GaussDB

M-Compatibility Developer Guide(Distributed)

Issue 01

Date 2025-10-23

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. 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

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.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 M-compatible Database System Overview 1
1.1 Query Request Handling Process 2
1.2 GaussDB Features That Are Not Supported by the M-compatible Database 2
2 Creating an M-compatible Database and User 3
3 Application Development Guide 4
3.1 Database Access Route 4
3.2 Development Specifications 4
3.3 Development Based on GaussDB JDBC 5
4 SQL Reference 6
4.1 Introduction to SQL 6
4.2 Keywords 7
4.3 Character Sets and Collations 38
4.3.1 Character Sets and Collations of the Client Connection 40
4.3.2 Database-level Character Sets and Collations 41
4.3.3 Table-level Character Sets and Collations 42
4.3.4 Column-level Character Sets and Collations 44
4.3.5 Character Sets and Collations of Expressions of the String Type 45
4.3.6 Rules for Combining Character Sets and Collations 46
4.4 SQL Syntax 47
4.4.1 Identifier Description 48
4.4.2 SQL Statements 48
4.4.2.1 SQL Syntax 49
4.4.2.2 DCL Syntax Overview 49
4.4.2.3 DDL Syntax Overview 49
4.4.2.4 DML Syntax Overview 55
4.4.2.5 Overview of Other Syntax 56
4.4.2.6 A 58
4.4.2.6.1 ALTER AUDIT POLICY 58
4.4.2.6.2 ALTER COORDINATOR 61
4.4.2.6.3 ALTER DATABASE 62
4.4.2.6.4 ALTER EXTENSION 64
4.4.2.6.5 ALTER GROUP 66

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.6.6 ALTER INDEX ... 68
4.4.2.6.7 ALTER NODE ... 70
4.4.2.6.8 ALTER NODE GROUP ... 73
4.4.2.6.9 ALTER RESOURCE LABEL ... 75
4.4.2.6.10 ALTER ROLE ... 76
4.4.2.6.11 ALTER SCHEMA ... 79
4.4.2.6.12 ALTER SEQUENCE ... 80
4.4.2.6.13 ALTER SESSION ... 82
4.4.2.6.14 ALTER SYSTEM KILL SESSION ... 84
4.4.2.6.15 ALTER TABLE ... 85
4.4.2.6.16 ALTER TABLE PARTITION ... 97
4.4.2.6.17 ALTER USER ... 107
4.4.2.6.18 ALTER VIEW ... 109
4.4.2.6.19 ANALYZE ... 111
4.4.2.7 B ... 114
4.4.2.7.1 BEGIN ... 114
4.4.2.8 C ... 115
4.4.2.8.1 CLEAN CONNECTION ... 115
4.4.2.8.2 CHECKPOINT ... 117
4.4.2.8.3 COMMENT ... 118
4.4.2.8.4 COMMIT ... 120
4.4.2.8.5 COMMIT PREPARED ... 121
4.4.2.8.6 COPY ... 122
4.4.2.8.7 CREATE AUDIT POLICY ... 129
4.4.2.8.8 CREATE DATABASE ... 132
4.4.2.8.9 CREATE EXTENSION ... 134
4.4.2.8.10 CREATE FUNCTION ... 135
4.4.2.8.11 CREATE GLOBAL INDEX ... 142
4.4.2.8.12 CREATE GROUP ... 148
4.4.2.8.13 CREATE INDEX ... 149
4.4.2.8.14 CREATE NODE ... 157
4.4.2.8.15 CREATE NODE GROUP ... 160
4.4.2.8.16 CREATE RESOURCE LABEL ... 161
4.4.2.8.17 CREATE ROLE ... 163
4.4.2.8.18 CREATE SCHEMA ... 168
4.4.2.8.19 CREATE SEQUENCE ... 169
4.4.2.8.20 CREATE TABLE ... 171
4.4.2.8.21 CREATE TABLE PARTITION ... 194
4.4.2.8.22 CREATE USER ... 208
4.4.2.8.23 CREATE VIEW ... 210
4.4.2.9 D ... 213
4.4.2.9.1 DEALLOCATE ... 213

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.9.2 DELETE 214
4.4.2.9.3 DO 218
4.4.2.9.4 DROP AUDIT POLICY 219
4.4.2.9.5 DROP DATABASE 219
4.4.2.9.6 DROP EXTENSION 220
4.4.2.9.7 DROP FUNCTION 221
4.4.2.9.8 DROP GROUP 223
4.4.2.9.9 DROP INDEX 224
4.4.2.9.10 DROP NODE 225
4.4.2.9.11 DROP NODE GROUP 226
4.4.2.9.12 DROP OWNED 226
4.4.2.9.13 DROP PREPARE 227
4.4.2.9.14 DROP RESOURCE LABEL 228
4.4.2.9.15 DROP ROLE 229
4.4.2.9.16 DROP SCHEMA 230
4.4.2.9.17 DROP SEQUENCE 231
4.4.2.9.18 DROP TABLE 232
4.4.2.9.19 DROP USER 233
4.4.2.9.20 DROP VIEW 234
4.4.2.10 E 235
4.4.2.10.1 EXECUTE 235
4.4.2.10.2 EXECUTE DIRECT 236
4.4.2.10.3 EXPLAIN 238
4.4.2.11 G 243
4.4.2.11.1 GRANT 243
4.4.2.12 I 251
4.4.2.12.1 INSERT 251
4.4.2.13 L 258
4.4.2.13.1 LOCK 258
4.4.2.13.2 LOAD DATA 261
4.4.2.14 P 265
4.4.2.14.1 PREPARE 265
4.4.2.14.2 PREPARE TRANSACTION 266
4.4.2.14.3 PURGE 267
4.4.2.15 R 269
4.4.2.15.1 REINDEX 269
4.4.2.15.2 RELEASE SAVEPOINT 273
4.4.2.15.3 REPLACE 274
4.4.2.15.4 RESET 278
4.4.2.15.5 REVOKE 278
4.4.2.15.6 ROLLBACK 281
4.4.2.15.7 ROLLBACK PREPARED 281

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.15.8 ROLLBACK TO SAVEPOINT....282
4.4.2.16 S....283
4.4.2.16.1 SAVEPOINT....283
4.4.2.16.2 SELECT....285
4.4.2.16.3 SELECT INTO....299
4.4.2.16.4 SET....302
4.4.2.16.5 SET ROLE....303
4.4.2.16.6 SET SESSION AUTHORIZATION....304
4.4.2.16.7 SET TRANSACTION....305
4.4.2.16.8 SHOW....307
4.4.2.16.9 START TRANSACTION....308
4.4.2.17 T....309
4.4.2.17.1 TRUNCATE....309
4.4.2.18 U....312
4.4.2.18.1 UPDATE....312
4.4.2.18.2 USE....317
4.4.2.19 V....318
4.4.2.19.1 VACUUM....318
4.5 Functions and Operators....321
4.5.1 Operator Overview....324
4.5.2 Logical Operators....327
4.5.3 Bitwise Operators....330
4.5.4 Pattern Matching Operators....331
4.5.5 Comparison Functions and Comparison Operators....339
4.5.6 Numeric Operation Functions and Arithmetic Operators....343
4.5.7 String Functions....353
4.5.8 Datetime Functions....368
4.5.9 Type Conversion Functions....397
4.5.10 Aggregate Functions....399
4.5.11 Encryption and Decryption Functions....407
4.5.12 Process Control Functions....411
4.5.13 Internal Functions....412
4.5.14 Other Functions....426
4.6 Data Types....427
4.6.1 Overview....427
4.6.2 Boolean Types....428
4.6.3 Binary Types....429
4.6.4 Character Types....431
4.6.5 Date and Time Types....433
4.6.6 Bit String Types....438
4.6.7 Numeric Types....440
4.6.7.1 Signed Integer Types....440

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.6.7.2 Unsigned Integer Types 442
4.6.7.3 Fixed-Point Types 444
4.6.7.4 Floating-Point Types 447
4.7 Type Conversion 450
4.7.1 Implicit Type Conversion 451
4.7.2 Explicit Type Conversion 452
4.7.3 UNION, CASE, and Related Constructs 453
4.8 Expressions 455
4.8.1 Simple Expressions 455
4.8.2 Condition Expressions 456
4.8.3 Subquery Expressions 459
4.8.4 Row Expressions 462
4.8.5 Time Interval Expressions 464
5 System Catalogs and System Views 467
5.1 System Catalogs 467
5.2 System Views 467
6 Schemas 476
6.1 Information Schemas 485
6.1.1 character_sets 486
6.1.2 collations 487
6.1.3 collation_character_set_applicability 488
6.1.4 columns 488
6.1.5 column_privileges 491
6.1.6 engines 491
6.1.7 events 492
6.1.8 files 494
6.1.9 global_status 496
6.1.10 global_variables 497
6.1.11 key_column_usage 497
6.1.12 optimizer_trace 499
6.1.13 parameters 500
6.1.14 partitions 501
6.1.15 plugins 503
6.1.16 processlist 504
6.1.17 profiling 505
6.1.18 referential_constraints 507
6.1.19 routines 508
6.1.20 schemata 511
6.1.21 schema_privileges 511
6.1.22 session_status 512
6.1.23 session_variables 512
6.1.24 statistics 513

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6.1.25 tables ... 515
6.1.26 table_constraints ... 517
6.1.27 table_privileges ... 517
6.1.28 triggers ... 518
6.1.29 user_privileges ... 520
6.1.30 views ... 521
6.2 m_schema ... 522
6.2.1 tables_priv ... 522
6.2.2 columns_priv ... 523
6.2.3 procs_priv ... 523
6.2.4 proc ... 524
6.2.5 func ... 526
7 Stored Procedures ... 527

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1 M-compatible Database System Overview

M-compatible is an enhanced version of GaussDB. It adds data types, syntax, and functions to meet enterprise customers' requirements for high performance, reliability, security, intelligence, scalability, and easy migration.

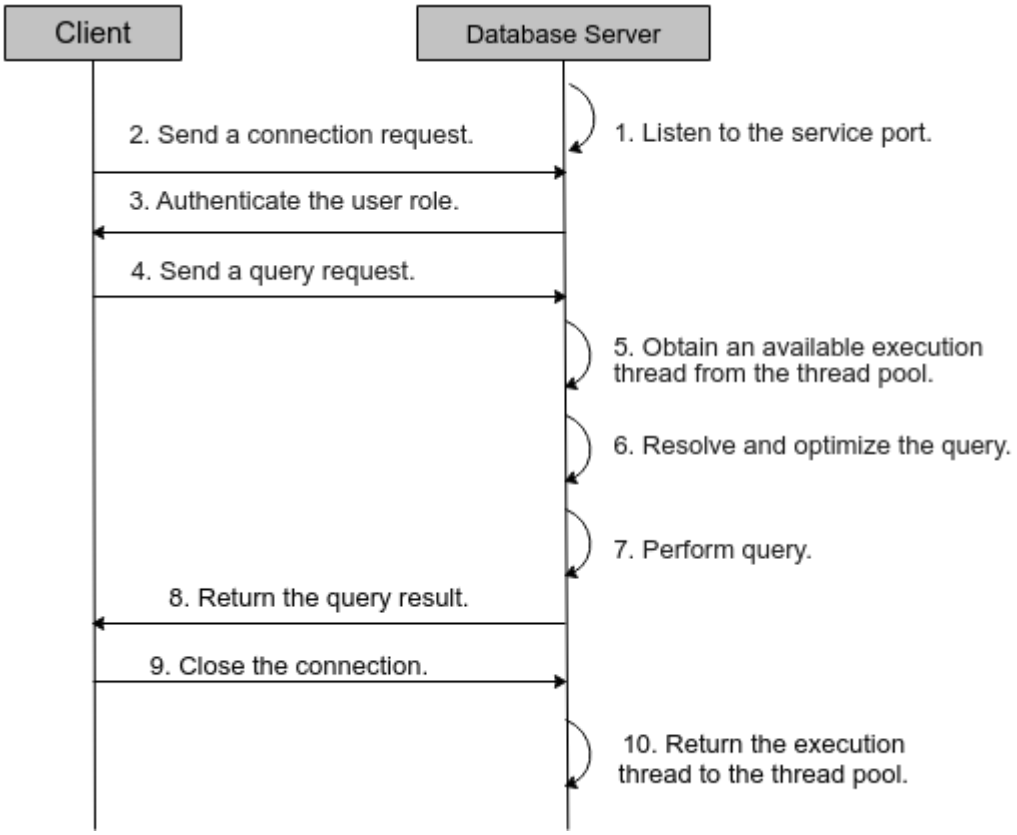
The logical architecture of the M-compatible database inherits that of GaussDB. For details, see "Database System Overview > Database Logical Architecture" in *Developer Guide*.

For details about how to design databases and schemas, see the relevant content in "MySQL Compatibility Description > Overview of MySQL Compatibility > Overview of M-compatible Mode" in *Compatibility Description*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1.1 Query Request Handling Process

Figure 1-1 GaussDB service response process



1.2 GaussDB Features That Are Not Supported by the M-compatible Database

The M-compatible database does not support the following features in *Feature Guide*:

1. Materialized views
2. Features related to fully-encrypted databases, including setting encrypted equality query and memory decryption emergency channel
3. SPM

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2 Creating an M-compatible Database and User

An M-compatible database must be created in a non-M-compatible database. Ensure that the **templatem** template database exists in the current environment.

```
gaussdb=# \l
                              List of databases
   Name    | Owner | Encoding |   Collate   |    Ctype    | Access privileges
-----------+-------+----------+-------------+-------------+-------------------
 postgres  | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/omm          +
           |       |          |             |             | omm=CTc/omm
 template1 | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/omm          +
           |       |          |             |             | omm=CTc/omm
 templatem | omm   | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/omm          +
           |       |          |             |             | omm=CTc/omm
(4 rows)
```

Note: After the upgrade from a version earlier than 505.2.0 to the latest version, if the **templatem** template database is not created in the basic package, the **templatem** template database will not be created after the upgrade. To use the M-compatible database, you need to create it by referring to the following example:

```
gaussdb=# CREATE DATABASE m_db dbcompatibility = 'M' template=templatem;
gaussdb=# \c m_db
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "m_db" as user "omm".
m_db=# CREATE USER omm_test password '********';
m_db=# GRANT ALL privileges to omm_test;
```

For details about other operations after the M-compatible database is created, see **SQL Reference**.

NOTICE

A GaussDB cluster can have only one M-compatible database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3 Application Development Guide

3.1 Database Access Route

For details about how to design databases and schemas, see the relevant content in "MySQL Compatibility Description > Overview of MySQL Compatibility > Overview of M-compatible Mode" in *Compatibility Description*.

Step 1 Create the database and user by referring to **Creating an M-compatible Database and User**.

Step 2 By default, schemas with the same name as **omm_test** and other default schemas are generated in the m_db database. These schemas are the workspace. You can use any of the following methods to access a data table:

1. Specify a database and schema to use the table name for access.
2. Specify a database and do not specify a schema. The system routes the request to the schema with the same name created by the connected user by default and uses **schema.table** to access the schema.
3. Specify a database and do not specify a schema. SQL statements or functions are used to switch the schema and directly use the table name for access.

----End

3.2 Development Specifications

If the connection pool mechanism is used during application development, comply with the following specifications before you return the connection to the connection pool:

- If GUC parameters are set in the connection, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status before you return the connection to the connection pool.
- If a temporary table is used, delete the temporary table.

If the status or temporary table is not cleared, the correctness of subsequent operations using the connection pool will be affected.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

When connecting to applications, ensure that SSL is used for secure TCP/IP connection or RSA is used for login password encryption to ensure user information security.

3.3 Development Based on GaussDB JDBC

Java Database Connectivity (JDBC) is a Java API for running SQL statements. JDBC provides a unified access API for different relational databases, based on which applications process data.

The M-compatible database can be developed based on GaussDB JDBC. For details, see "Application Development > Development Based on JDBC" in *Developer Guide*. Some functions of the M-compatible database are different from those of GaussDB. For details about the differences, see **Restrictions on Connecting GaussDB JDBC to the M-compatible Database**.

Restrictions on Connecting GaussDB JDBC to the M-compatible Database

1. When the DriverManager.getConnection(String url, Properties info) method is used to create a database connection, the **info** parameter does not support the **binaryTransfer**, **binaryTransferEnable**, and **binaryTransferDisable** attributes.
2. When the DriverManager.getConnection(String url, Properties info) method is used to create a database connection and the **info** parameter is set to **mCompatible**, some functions and behaviors are changed. The change is compatible with the MySQL JDBC 5.1 driver. For details, see "Application Development Guide > Development Based on JDBC > Development Procedure > Connecting to a Database" in *Developer Guide*.
3. For details about other GaussDB features that are not supported, see **GaussDB Features That Are Not Supported by the M-compatible Database**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4 SQL Reference

4.1 Introduction to SQL

What Is SQL?

SQL is a standard computer language used to control the access to databases and manage data in databases.

SQL provides different statements to enable you to:

- Query data.
- Insert, update, and delete rows.
- Create, replace, modify, and delete objects.
- Control the access to a database and its objects.
- Maintain the consistency and integrity of a database.

SQL consists of commands and functions that are used to manage databases and database objects. SQL can also forcibly implement the rules for data types, expressions, and texts. Therefore, **SQL Reference** describes data types, expressions, functions, and operators in addition to SQL syntax.

Development of SQL Standards

The development history of SQL standards is as follows:

- 1986: ANSI X3.135-1986, ISO/IEC 9075:1986, SQL-86
- 1989: ANSI X3.135-1989, ISO/IEC 9075:1989, SQL-89
- 1992: ANSI X3.135-1992, ISO/IEC 9075:1992, SQL-92
- 1999: ISO/IEC 9075:1999, SQL:1999
- 2003: ISO/IEC 9075:2003, SQL:2003
- 2011: ISO/IEC 9075:2011, SQL:2011
- 2016: ISO/IEC 9075:2016, SQL:2016
- 2019: ISO/IEC 9075:2019, SQL:2019

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQL Standards Supported by GaussDB

By default, GaussDB supports most features of SQL:2016.

4.2 Keywords

The SQL contains reserved keywords and non-reserved keywords. Standards require that reserved keywords not be used as other identifiers. Non-reserved keywords have special meanings only in a specific environment and can be used as identifiers in other environments. Based on reserved and non-reserved keywords, M-compatible databases further define the application scope of keywords as identifiers and provide the four more types of keywords.

- Reserved: Common reserved keywords cannot be used as identifiers of any database objects, such as tables, columns, functions, types, views, indexes, and variables.
- Non-reserved: Common non-reserved keywords can be used as identifiers of any database object.
- Reserved (which can be a function or type): Special reserved keywords can be used as identifiers of functions and variables.
- Non-reserved (which cannot be a function or type): Special non-reserved keywords cannot be used as identifiers of functions and variables.

Constraint differences:

- For detailed constraint differences in keyword implementation, see "MySQL Compatibility Description > M-compatible Mode > SQL > Keywords" in *Compatibility Description*.

NOTICE

1. Currently, the non-reserved keywords have the following restrictions when being used as the identifier of a database object:

   Keywords such as BEGIN, BY, CLOSE, CURSOR, DECLARE, DELETE, EXECUTE, FUNCTION, IF, IMMEDIATE, INSERT, LOOP, MOVE, OF, REF, RELEASE, RETURN, SAVEPOINT, STRICT, TYPE, and UPDATE without backquotes cannot be used as variable names.

2. Reserved keywords and the following keywords cannot be directly used as column aliases (that is, usage similar to SELECT 1 BIGINT may cause errors):

   BETWEEN, BIGINT, BLOB, CHAR, CHARACTER, CROSS, DEC, DECIMAL, DIV, DOUBLE, EXISTS, FLOAT, FLOAT4, FLOAT8, GROUPING, INNER, INOUT, INT, INT1, INT2, INT3, INT4, INT8, INTEGER, JOIN, LEFT, LIKE, LONGBLOB, LONGTEXT, MEDIUMBLOB, MEDIUMINT, MEDIUMTEXT, MOD, NATURAL, NUMERIC, OUT, OUTER, PRECISION, REAL, RIGHT, ROW, ROW_NUMBER, SIGNED, SMALLINT, SOUNDS, TINYBLOB, TINYINT, TINYTEXT, VALUES, VARCHAR, VARYING, and WITHOUT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

SQL Keywords

Table 4-1 SQL keywords

Keyword	Type
ABSOLUTE	Non-reserved
ACCESS	Non-reserved
ACCESSIBLE	Reserved
ACCOUNT	Non-reserved
ACTION	Non-reserved
ACTIVE	Non-reserved
ADD	Non-reserved
ADDDATE	Non-reserved
ADMIN	Non-reserved
AFTER	Non-reserved
AGAINST	Non-reserved
AGGREGATE	Non-reserved
ALGORITHM	Non-reserved
ALL	Reserved
ALTER	Non-reserved
ALWAYS	Non-reserved
ANALYZE	Reserved
AND	Reserved
ANY	Reserved
APPEND	Non-reserved
ARRAY	Reserved
AS	Reserved
ASC	Reserved
ASCII	Non-reserved
ASENSITIVE	Reserved
ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS	Non-reserved
AT	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
ATTRIBUTE	Non-reserved
AUTHENTICATION	Non-reserved
AUTHID	Non-reserved
AUTHORIZATION	Non-reserved
AUTO_INCREMENT	Non-reserved
AUTOEXTEND_SIZE	Non-reserved
AVG	Non-reserved
AVG_ROW_LENGTH	Non-reserved
BACKUP	Non-reserved
BEFORE	Non-reserved
BEGIN	Non-reserved
BEGIN_NON_ANOYBLOCK	Non-reserved
BETWEEN	Non-reserved (cannot be functions or types)
BIGINT	Non-reserved (cannot be functions or types)
BINARY	Reserved
BINLOG	Non-reserved
BIT	Non-reserved (cannot be functions or types)
BLOB	Non-reserved (cannot be functions or types)
BLOCK	Non-reserved
BOOL	Non-reserved (cannot be functions or types)
BOOLEAN	Non-reserved (cannot be functions or types)
BOTH	Reserved
BTREE	Non-reserved
BUCKETS	Non-reserved
BULK	Non-reserved
BY	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
BYTE	Non-reserved
CACHE	Non-reserved
CALL	Non-reserved
CALLED	Non-reserved
CANCELABLE	Non-reserved
CASCADE	Non-reserved
CASCADED	Non-reserved
CASE	Reserved
CAST	Non-reserved
CATALOG_NAME	Non-reserved
CHAIN	Non-reserved
CHALLENGE_RESPONSE	Non-reserved
CHANGE	Non-reserved
CHANGED	Non-reserved
CHANNEL	Non-reserved
CHAR	Non-reserved (cannot be functions or types)
CHARACTER	Non-reserved (cannot be functions or types)
CHARSET	Non-reserved
CHECK	Reserved
CHECKPOINT	Non-reserved
CHECKSUM	Non-reserved
CIPHER	Non-reserved
CLASS_ORIGIN	Non-reserved
CLEAN	Non-reserved
CLIENT	Non-reserved
CLONE	Non-reserved
CLOSE	Non-reserved
CLUSTER	Non-reserved
COALESCE	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
CODE	Non-reserved
COLLATE	Reserved
COLLATION	Non-reserved
COLUMN	Reserved
COLUMN_FORMAT	Non-reserved
COLUMN_NAME	Non-reserved
COLUMNS	Non-reserved
COMMENT	Non-reserved
COMMIT	Non-reserved
COMMITTED	Non-reserved
COMPACT	Reserved (functions and types allowed)
COMPLETION	Non-reserved
COMPONENT	Non-reserved
COMPRESSED	Non-reserved
COMPRESSION	Non-reserved
CONCURRENT	Non-reserved
CONCURRENTLY	Reserved (functions and types allowed)
CONDITION	Non-reserved
CONNECTION	Non-reserved
CONSISTENT	Non-reserved
CONSTRAINT	Reserved
CONSTRAINT_CATALOG	Non-reserved
CONSTRAINT_NAME	Non-reserved
CONSTRAINT_SCHEMA	Non-reserved
CONSTRAINTS	Non-reserved
CONTAINS	Non-reserved
CONTEXT	Non-reserved
CONTINUE	Non-reserved
CONVERT	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
COORDINATOR	Non-reserved
COORDINATORS	Non-reserved
COPY	Non-reserved
COST	Non-reserved
COUNT	Non-reserved
CPU	Non-reserved
CREATE	Reserved
CROSS	Reserved (functions and types allowed)
CSV	Non-reserved
CUBE	Non-reserved
CUME_DIST	Reserved
CURRENT	Non-reserved
CURRENT_DATE	Reserved
CURRENT_SCHEMA	Reserved (functions and types allowed)
CURRENT_TIME	Reserved
CURRENT_TIMESTAMP	Reserved
CURRENT_USER	Reserved
CURSOR	Non-reserved
CURSOR_NAME	Non-reserved
CURTIME	Non-reserved
CYCLE	Non-reserved
DATA	Non-reserved
DATABASE	Non-reserved
DATABASES	Reserved
DATAFILE	Non-reserved
DATANODE	Non-reserved
DATANODES	Non-reserved
DATE	Non-reserved (cannot be functions or types)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
DATE_ADD	Non-reserved
DATE_FORMAT	Non-reserved
DATE_SUB	Non-reserved
DATETIME	Non-reserved
DAY	Non-reserved
DAY_HOUR	Reserved
DAY_MICROSECOND	Reserved
DAY_MINUTE	Reserved
DAY_SECOND	Reserved
DEALLOCATE	Non-reserved
DEC	Non-reserved (cannot be functions or types)
DECIMAL	Non-reserved (cannot be functions or types)
DECLARE	Non-reserved
DECODE	Non-reserved
DEFAULT	Reserved
DEFAULTS	Non-reserved
DEFAULT_AUTH	Non-reserved
DEFINER	Non-reserved
DEFINITION	Non-reserved
DELAY_KEY_WRITE	Non-reserved
DELAYED	Reserved
DELETE	Non-reserved
DELIMITER	Non-reserved
DENSE_RANK	Reserved
DESC	Reserved
DESCRIBE	Reserved
DESCRIPTION	Non-reserved
DETERMINISTIC	Non-reserved
DIAGNOSTICS	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
DICTIONARY	Non-reserved
DIRECT	Non-reserved
DIRECTORY	Non-reserved
DISABLE	Non-reserved
DISCARD	Non-reserved
DISK	Non-reserved
DISTINCT	Reserved
DISTINCTROW	Reserved
DISTRIBUTE	Non-reserved
DISTRIBUTED	Non-reserved
DISTRIBUTION	Non-reserved
DIV	Reserved (functions and types allowed)
DO	Non-reserved
DOCUMENT	Non-reserved
DOUBLE	Non-reserved (cannot be functions or types)
DROP	Non-reserved
DUAL	Reserved
DUMPFILE	Non-reserved
DUPLICATE	Non-reserved
DYNAMIC	Non-reserved
EACH	Non-reserved
ELSE	Reserved
ELSEIF	Reserved
EMPTY	Reserved
ENABLE	Non-reserved
ENCLOSED	Non-reserved
ENCODING	Non-reserved
ENCRYPTION	Non-reserved
END	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
ENDS	Non-reserved
ENFORCED	Non-reserved
ENGINE	Non-reserved
ENGINE_ATTRIBUTE	Non-reserved
ENGINES	Non-reserved
ENUM	Non-reserved
ERROR	Non-reserved
ERRORS	Non-reserved
ESCAPE	Non-reserved
ESCAPED	Non-reserved
EVENT	Non-reserved
EVENTS	Non-reserved
EVERY	Non-reserved
EXCEPT	Reserved
EXCHANGE	Non-reserved
EXCLUDE	Non-reserved
EXLUDING	Non-reserved
EXCLUSIVE	Non-reserved
EXECUTE	Non-reserved
EXISTS	Non-reserved (cannot be functions or types)
EXIT	Reserved
EXPANSION	Non-reserved
EXPIRE	Non-reserved
EXPIRED	Non-reserved
EXPLAIN	Non-reserved
EXPORT	Non-reserved
EXTENDED	Non-reserved
EXTENSION	Non-reserved
EXTENT_SIZE	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
EXTERNAL	Non-reserved
EXTRACT	Non-reserved
FACTOR	Non-reserved
FAILED_LOGIN_ATTEMPTS	Non-reserved
FALSE	Reserved
FAST	Non-reserved
FAULTS	Non-reserved
FENCED	Non-reserved
FETCH	Reserved
FIELDS	Non-reserved
FILE	Non-reserved
FILE_BLOCK_SIZE	Non-reserved
FILTER	Non-reserved
FINISH	Non-reserved
FIRST	Non-reserved
FIRST_VALUE	Reserved
FIXED	Non-reserved (cannot be functions or types)
FLOAT	Non-reserved (cannot be functions or types)
FLOAT4	Non-reserved (cannot be functions or types)
FLOAT8	Non-reserved (cannot be functions or types)
FLUSH	Non-reserved
FOLLOWING	Non-reserved
FOLLOWS	Non-reserved
FOR	Reserved
FORCE	Reserved
FOREIGN	Reserved
FORMAT	Non-reserved
FORWARD	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
FOUND	Non-reserved
FREEZE	Reserved (functions and types allowed)
FROM	Reserved
FULL	Non-reserved
FULLTEXT	Reserved
FUNCTION	Non-reserved
GENERAL	Non-reserved
GENERATE	Non-reserved
GENERATED	Non-reserved
GEOMCOLLECTION	Non-reserved
GEOMETRY	Non-reserved
GEOMETRYCOLLECTION	Non-reserved
GET	Reserved
GET_FORMAT	Non-reserved
GET_MASTER_PUBLIC_KEY	Non-reserved
GET_SOURCE_PUBLIC_KEY	Non-reserved
GLOBAL	Non-reserved
GRANT	Reserved
GRANTED	Non-reserved
GRANTS	Non-reserved
GROUP	Reserved
GROUP_REPLICATION	Non-reserved
GROUPING	Non-reserved (cannot be functions or types)
GROUPS	Reserved
GTID_ONLY	Non-reserved
HANDLER	Non-reserved
HASH	Non-reserved
HAVING	Reserved
HELP	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
HIGH_PRIORITY	Reserved
HISTOGRAM	Non-reserved
HISTORY	Non-reserved
HOLD	Non-reserved
HOST	Non-reserved
HOSTS	Non-reserved
HOUR	Non-reserved
HOUR_MICROSECOND	Reserved
HOUR_MINUTE	Reserved
HOUR_SECOND	Reserved
IDENTIFIED	Non-reserved
IDENTITY	Non-reserved
IF	Reserved
IGNORE	Reserved
IGNORE_SERVER_IDS	Non-reserved
IMMUTABLE	Non-reserved
IMPORT	Non-reserved
IN	Reserved
INACTIVE	Non-reserved
INCLUDING	Non-reserved
INCREMENT	Non-reserved
INDEX	Reserved
INDEXES	Non-reserved
INFILE	Non-reserved
INITIAL	Non-reserved
INITIAL_SIZE	Non-reserved
INITIATE	Non-reserved
INNER	Reserved (functions and types allowed)
INOUT	Non-reserved (cannot be functions or types)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
INPUT	Non-reserved
INSENSITIVE	Non-reserved
INSERT	Non-reserved
INSERT_METHOD	Non-reserved
INSTALL	Non-reserved
INSTANCE	Non-reserved
INT	Non-reserved (cannot be functions or types)
INT1	Non-reserved (cannot be functions or types)
INT2	Non-reserved (cannot be functions or types)
INT3	Non-reserved (cannot be functions or types)
INT4	Non-reserved (cannot be functions or types)
INT8	Non-reserved (cannot be functions or types)
INTEGER	Non-reserved (cannot be functions or types)
INTERNAL	Non-reserved
INTERSECT	Reserved
INTERVAL	Reserved
INTO	Reserved
INVISIBLE	Non-reserved
INVOKER	Non-reserved
IO	Non-reserved
IO_AFTER_GTIDS	Reserved
IO_BEFORE_GTIDS	Reserved
IO_THREAD	Non-reserved
IPC	Non-reserved
IS	Reserved
ISNULL	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
ISOLATION	Non-reserved
ISSUER	Non-reserved
ITERATE	Reserved
JOIN	Reserved (functions and types allowed)
JSON	Non-reserved
JSON_TABLE	Reserved
JSON_VALUE	Non-reserved
KEY	Reserved
KEY_BLOCK_SIZE	Non-reserved
KEYRING	Non-reserved
KEYS	Reserved
KILL	Non-reserved
LAG	Reserved
LANGUAGE	Non-reserved
LAST	Non-reserved
LAST_VALUE	Reserved
LATERAL	Reserved
LC_COLLATE	Non-reserved
LEAD	Reserved
LEADING	Reserved
LEAKPROOF	Non-reserved
LEAVE	Reserved
LEAVES	Non-reserved
LEFT	Reserved (functions and types allowed)
LESS	Non-reserved
LEVEL	Non-reserved
LIKE	Reserved (functions and types allowed)
LIMIT	Reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
LINEAR	Reserved
LINES	Non-reserved
LINESTRING	Non-reserved
LIST	Non-reserved
LOAD	Non-reserved
LOAD_BAD	Non-reserved
LOAD_DISCARD	Non-reserved
LOCAL	Non-reserved
LOCALTIME	Reserved
LOCALTIMESTAMP	Reserved
LOCATION	Non-reserved
LOCK	Reserved
LOCKED	Non-reserved
LOCKS	Non-reserved
LOGFILE	Non-reserved
LOGS	Non-reserved
LONG	Reserved
LONGBLOB	Non-reserved (cannot be functions or types)
LONGTEXT	Non-reserved (cannot be functions or types)
LOOP	Non-reserved
LOW_PRIORITY	Reserved
MASTER	Non-reserved
MASTER_AUTO_POSITION	Non-reserved
MASTER_BIND	Reserved
MASTER_COMPRESSION_ALGORITHMS	Non-reserved
MASTER_CONNECT_RETRY	Non-reserved
MASTER_DELAY	Non-reserved
MASTER_HEARTBEAT_PERIOD	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
MASTER_HOST	Non-reserved
MASTER_LOG_FILE	Non-reserved
MASTER_LOG_POS	Non-reserved
MASTER_PASSWORD	Non-reserved
MASTER_PORT	Non-reserved
MASTER_PUBLIC_KEY_PATH	Non-reserved
MASTER_RETRY_COUNT	Non-reserved
MASTER_SSL	Non-reserved
MASTER_SSL_CA	Non-reserved
MASTER_SSL_CAPATH	Non-reserved
MASTER_SSL_CERT	Non-reserved
MASTER_SSL_CIPHER	Non-reserved
MASTER_SSL_CRL	Non-reserved
MASTER_SSL_CRLPATH	Non-reserved
MASTER_SSL_KEY	Non-reserved
MASTER_SSL_VERIFY_SERVER_CERT	Reserved
MASTER_TLS_CIPHERSUITES	Non-reserved
MASTER_TLS_VERSION	Non-reserved
MASTER_USER	Non-reserved
MASTER_ZSTD_COMPRESSION_LEVEL	Non-reserved
MATCH	Reserved
MAX_CONNECTIONS_PER_HOUR	Non-reserved
MAX_QUERIES_PER_HOUR	Non-reserved
MAX_ROWS	Non-reserved
MAX_SIZE	Non-reserved
MAX_UPDATES_PER_HOUR	Non-reserved
MAX_USER_CONNECTIONS	Non-reserved
MAXVALUE	Reserved
MEDIUM	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
MEDIUMBLOB	Non-reserved (cannot be functions or types)
MEDIUMINT	Non-reserved (cannot be functions or types)
MEDIUMTEXT	Non-reserved (cannot be functions or types)
MEMBER	Non-reserved
MEMORY	Non-reserved
MERGE	Non-reserved
MESSAGE_TEXT	Non-reserved
MICROSECOND	Non-reserved
MIDDLEINT	Reserved
MIGRATE	Non-reserved
MIN_ROWS	Non-reserved
MINUTE	Non-reserved
MINUTE_MICROSECOND	Reserved
MINUTE_SECOND	Reserved
MINVALUE	Non-reserved
MOD	Reserved (functions and types allowed)
MODE	Non-reserved
MODIFIES	Reserved
MODIFY	Non-reserved
MONTH	Non-reserved
MOVE	Non-reserved
MULTILINESTRING	Non-reserved
MULTIPOINT	Non-reserved
MULTIPOLYGON	Non-reserved
MUTEX	Non-reserved
MYSQL_ERRNO	Non-reserved
NAME	Non-reserved
NAMES	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
NATIONAL	Non-reserved (cannot be functions or types)
NATURAL	Reserved (functions and types allowed)
NCHAR	Non-reserved (cannot be functions or types)
NDB	Non-reserved
NDBCLUSTER	Non-reserved
NESTED	Non-reserved
NETWORK_NAMESPACE	Non-reserved
NEVER	Non-reserved
NEW	Non-reserved
NEXT	Non-reserved
NO	Non-reserved
NO_WAIT	Non-reserved
NO_WRITE_TO_BINLOG	Reserved
NOCYCLE	Non-reserved
NODE	Non-reserved
NODEGROUP	Non-reserved
NOMAXVALUE	Non-reserved
NOMINVALUE	Non-reserved
NONE	Non-reserved
NOT	Reserved
NOTHING	Non-reserved
NOW	Non-reserved
NOWAIT	Non-reserved
NTH_VALUE	Reserved
NTILE	Reserved
NULL	Reserved
NULLS	Non-reserved
NUMBER	Non-reserved (cannot be functions or types)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
NUMERIC	Non-reserved (cannot be functions or types)
NVARCHAR	Non-reserved (cannot be functions or types)
OF	Non-reserved
OFF	Non-reserved
OFFSET	Reserved
OIDS	Non-reserved
OJ	Non-reserved
OLD	Non-reserved
ON	Reserved
ONE	Non-reserved
ONLY	Reserved
OPEN	Non-reserved
OPERATOR	Non-reserved
OPTIMIZE	Reserved
OPTIMIZE_COSTS	Reserved
OPTION	Non-reserved
OPTIONAL	Non-reserved
OPTIONALLY	Non-reserved
OPTIONS	Non-reserved
OR	Reserved
ORDER	Reserved
ORDINALITY	Non-reserved
ORGANIZATION	Non-reserved
OTHERS	Non-reserved
OUT	Non-reserved (cannot be functions or types)
OUTER	Reserved (functions and types allowed)
OUTFILE	Non-reserved
OVER	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
OWNED	Non-reserved
OWNER	Non-reserved
PACK_KEYS	Non-reserved
PAGE	Non-reserved
PARSER	Non-reserved
PARTIAL	Non-reserved
PARTITION	Non-reserved
PARTITIONING	Non-reserved
PARTITIONS	Non-reserved
PASSWORD	Non-reserved
PASSWORD_LOCK_TIME	Non-reserved
PATH	Non-reserved
PERCENT_RANK	Reserved
PERSIST	Non-reserved
PERSIST_ONLY	Non-reserved
PHASE	Non-reserved
PLUGIN	Non-reserved
PLUGINS	Non-reserved
PLUGIN_DIR	Non-reserved
POINT	Non-reserved
POLYGON	Non-reserved
PORT	Non-reserved
PRECEDES	Non-reserved
PRECEDING	Non-reserved
PRECISION	Non-reserved (cannot be functions or types)
PREPARE	Non-reserved
PREPARED	Non-reserved
PRESERVE	Non-reserved
PREV	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
PRIMARY	Reserved
PRIVILEGES	Non-reserved
PRIVILEGE_CHECKS_USER	Non-reserved
PROCEDURE	Reserved
PROCESS	Non-reserved
PROCESSLIST	Non-reserved
PROFILE	Non-reserved
PROFILES	Non-reserved
PROXY	Non-reserved
PURGE	Non-reserved
QUARTER	Non-reserved
QUERY	Non-reserved
QUICK	Non-reserved
RANDOM	Non-reserved
RANGE	Non-reserved
RANK	Reserved
READ	Reserved
READS	Reserved
READ_ONLY	Non-reserved
READ_WRITE	Reserved
REAL	Non-reserved (cannot be functions or types)
REBUILD	Non-reserved
RECOVER	Non-reserved
RECURSIVE	Non-reserved
RECYCLEBIN	Non-reserved
REDO_BUFFER_SIZE	Non-reserved
REDUNDANT	Non-reserved
REFERENCE	Non-reserved
REFERENCES	Reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
REGEXP	Reserved
REGISTRATION	Non-reserved
REINDEX	Non-reserved
RELATIVE	Non-reserved
RELAY	Non-reserved
RELAYLOG	Non-reserved
RELAY_LOG_FILE	Non-reserved
RELAY_LOG_POS	Non-reserved
RELAY_THREAD	Non-reserved
RELEASE	Non-reserved
RELOAD	Non-reserved
RELOPTIONS	Non-reserved
REMOVE	Non-reserved
RENAME	Non-reserved
REORGANIZE	Non-reserved
REPAIR	Non-reserved
REPEAT	Reserved
REPEATABLE	Non-reserved
REPLACE	Non-reserved
REPLICA	Non-reserved
REPLICAS	Non-reserved
REPLICATE_DO_DB	Non-reserved
REPLICATE_DO_TABLE	Non-reserved
REPLICATE_IGNORE_DB	Non-reserved
REPLICATE_IGNORE_TABLE	Non-reserved
REPLICATE_REWRITE_DB	Non-reserved
REPLICATE_WILD_DO_TABLE	Non-reserved
REPLICATE_WILD_IGNORE_TABLE	Non-reserved
REPLICATION	Non-reserved
REQUIRE	Reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
REQUIRE_ROW_FORMAT	Non-reserved
REQUIRE_TABLE_PRIMARY_KEY_CHECK	Non-reserved
RESET	Non-reserved
RESIGNAL	Reserved
RESOURCE	Non-reserved
RESPECT	Non-reserved
RESTART	Non-reserved
RESTORE	Non-reserved
RESTRICT	Non-reserved
RESUME	Non-reserved
RETAIN	Non-reserved
RETURN	Non-reserved
RETURNED_SQLSTATE	Non-reserved
RETURNING	Reserved
RETURNS	Non-reserved
REUSE	Non-reserved
REVERSE	Non-reserved
REVOKE	Non-reserved
RIGHT	Reserved (functions and types allowed)
RLIKE	Reserved
ROLE	Non-reserved
ROLLBACK	Non-reserved
ROLLUP	Non-reserved
ROTATE	Non-reserved
ROUTINE	Non-reserved
ROW	Non-reserved (cannot be functions or types)
ROWS	Non-reserved
ROW_COUNT	Non-reserved
ROW_FORMAT	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
ROW_NUMBER	Reserved (functions and types allowed)
RTREE	Non-reserved
SAVEPOINT	Non-reserved
SCHEDULE	Non-reserved
SCHEMA	Non-reserved
SCHEMAS	Reserved
SCHEMA_NAME	Non-reserved
SCROLL	Non-reserved
SECOND	Non-reserved
SECOND_MICROSECOND	Reserved
SECONDARY	Non-reserved
SECONDARY_ENGINE	Non-reserved
SECONDARY_ENGINE_ATTRIBUTE	Non-reserved
SECONDARY_LOAD	Non-reserved
SECONDARY_UNLOAD	Non-reserved
SECURITY	Non-reserved
SELECT	Reserved
SENSITIVE	Reserved
SEPARATOR	Non-reserved
SEQUENCE	Non-reserved
SERIAL	Non-reserved
SERIALIZABLE	Non-reserved
SERVER	Non-reserved
SESSION	Non-reserved
SET	Non-reserved
SETOF	Non-reserved (cannot be functions or types)
SHARE	Non-reserved
SHIPPABLE	Non-reserved
SHOW	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
SHUTDOWN	Non-reserved
SIGNAL	Reserved
SIGNED	Non-reserved
SIMPLE	Non-reserved
SIZE	Non-reserved
SKIP	Non-reserved
SLAVE	Non-reserved
SLICE	Non-reserved
SLICEGROUP	Non-reserved
SLOW	Non-reserved
SMALLDATETIME	Non-reserved (cannot be functions or types)
SMALLDATETIME_FORMAT	Non-reserved
SMALLINT	Non-reserved (cannot be functions or types)
SNAPSHOT	Non-reserved
SOCKET	Non-reserved
SOME	Reserved
SONAME	Non-reserved
SOUNDS	Non-reserved
SOURCE	Non-reserved
SOURCE_AUTO_POSITION	Non-reserved
SOURCE_BIND	Non-reserved
SOURCE_COMPRESSION_ALGORITHMS	Non-reserved
SOURCE_CONNECTION_AUTO_FAILOVER	Non-reserved
SOURCE_CONNECT_RETRY	Non-reserved
SOURCE_DELAY	Non-reserved
SOURCE_HEARTBEAT_PERIOD	Non-reserved
SOURCE_HOST	Non-reserved
SOURCE_LOG_FILE	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
SOURCE_LOG_POS	Non-reserved
SOURCE_PASSWORD	Non-reserved
SOURCE_PORT	Non-reserved
SOURCE_PUBLIC_KEY_PATH	Non-reserved
SOURCE_RETRY_COUNT	Non-reserved
SOURCE_SSL	Non-reserved
SOURCE_SSL_CA	Non-reserved
SOURCE_SSL_CAPATH	Non-reserved
SOURCE_SSL_CERT	Non-reserved
SOURCE_SSL_CIPHER	Non-reserved
SOURCE_SSL_CRL	Non-reserved
SOURCE_SSL_CRLPATH	Non-reserved
SOURCE_SSL_KEY	Non-reserved
SOURCE_SSL_VERIFY_SERVER_CERT	Non-reserved
SOURCE_TLS_CIPHERSUITES	Non-reserved
SOURCE_TLS_VERSION	Non-reserved
SOURCE_USER	Non-reserved
SOURCE_ZSTD_COMPRESSION_LEVEL	Non-reserved
SPATIAL	Reserved
SPECIFIC	Reserved
SPLIT	Non-reserved
SQL	Non-reserved
SQLEXCEPTION	Reserved
SQLSTATE	Reserved
SQLWARNNING	Reserved
SQL_AFTER_GTIDS	Non-reserved
SQL_AFTER_MTS_GAPS	Non-reserved
SQL_BEFORE_GTIDS	Non-reserved
SQL_BIG_RESULT	Reserved
SQL_BUFFER_RESULT	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
SQL_CALC_FOUND_ROWS	Reserved
SQL_NO_CACHE	Reserved
SQL_SMALL_RESULT	Non-reserved
SQL_THREAD	Non-reserved
SQL_TSI_DAY	Non-reserved
SQL_TSI_HOUR	Non-reserved
SQL_TSI_MINUTE	Non-reserved
SQL_TSI_MONTH	Non-reserved
SQL_TSI_QUARTER	Non-reserved
SQL_TSI_SECOND	Non-reserved
SQL_TSI_WEEK	Non-reserved
SQL_TSI_YEAR	Non-reserved
SRID	Non-reserved
SSL	Reserved
STABLE	Non-reserved
STACKED	Non-reserved
START	Non-reserved
STARTING	Non-reserved
STARTS	Non-reserved
STATS_AUTO_RECALC	Non-reserved
STATS_PERSISTENT	Non-reserved
STATS_SAMPLE_PAGES	Non-reserved
STATUS	Non-reserved
STDIN	Non-reserved
STDOUT	Non-reserved
STOP	Non-reserved
STORAGE	Non-reserved
STORED	Non-reserved
STRAIGHT_JOIN	Reserved
STREAM	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
STRICT	Non-reserved
STRING	Non-reserved
SUBCLASS_ORIGIN	Non-reserved
SUBDATE	Non-reserved
SUBJECT	Non-reserved
SUBPARTITION	Non-reserved
SUBPARTITIONS	Non-reserved
SUBSTR	Non-reserved
SUBSTRING	Non-reserved
SUPER	Non-reserved
SUSPEND	Non-reserved
SWAPS	Non-reserved
SWITCHES	Non-reserved
SYSDATE	Non-reserved
SYSTEM	Non-reserved
TABLE	Reserved
TABLES	Non-reserved
TABLESPACE	Non-reserved
TABLE_CHECKSUM	Non-reserved
TABLE_NAME	Non-reserved
TEMPORARY	Non-reserved
TEMPTABLE	Non-reserved
TERMINATED	Non-reserved
TEXT	Non-reserved (cannot be functions or types)
THAN	Non-reserved
THEN	Reserved
THREAD_PRIORITY	Non-reserved
TIES	Non-reserved
TIME	Non-reserved (cannot be functions or types)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
TIMESTAMP	Non-reserved (cannot be functions or types)
TIMESTAMPADD	Non-reserved
TIMESTAMPDIFF	Non-reserved
TINYBLOB	Non-reserved (cannot be functions or types)
TINYINT	Non-reserved (cannot be functions or types)
TINYTEXT	Non-reserved (cannot be functions or types)
TLS	Non-reserved
TO	Reserved
TRAILING	Reserved
TRANSACTION	Non-reserved
TRIGGER	Reserved
TRIGGERS	Non-reserved
TRIM	Non-reserved
TRUE	Reserved
TRUNCATE	Non-reserved
TYPE	Non-reserved
TYPES	Non-reserved
UNBOUNDED	Non-reserved
UNCOMMITTED	Non-reserved
UNDEFINED	Non-reserved
UNDO	Reserved
UNDOFILE	Non-reserved
UNDO_BUFFER_SIZE	Non-reserved
UNICODE	Non-reserved
UNINSTALL	Non-reserved
UNION	Reserved
UNIQUE	Reserved
UNKNOWN	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
UNLOCK	Non-reserved
UNLOGGED	Non-reserved
UNREGISTER	Non-reserved
UNSIGNED	Reserved
UNTIL	Non-reserved
UNUSABLE	Non-reserved
UPDATE	Non-reserved
UPGRADE	Non-reserved
URL	Non-reserved
USAGE	Non-reserved
USE	Reserved
USER	Reserved
USER_RESOURCES	Non-reserved
USE_FRM	Non-reserved
USING	Reserved
UTC_DATE	Reserved
UTC_TIME	Reserved
UTC_TIMESTAMP	Reserved
VACUUM	Non-reserved
VALID	Non-reserved
VALIDATION	Non-reserved
VALUE	Non-reserved
VALUES	Non-reserved (cannot be functions or types)
VARBINARY	Reserved
VARCHAR	Non-reserved (cannot be functions or types)
VARCHAR2	Non-reserved (cannot be functions or types)
VARCHARACTER	Non-reserved (cannot be functions or types)
VARIABLES	Non-reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
VARYING	Non-reserved
VCPU	Non-reserved
VERBOSE	Reserved (functions and types allowed)
VERSION	Non-reserved
VIEW	Non-reserved
VIRTUAL	Reserved
VISIBLE	Non-reserved
VOLATILE	Non-reserved
WAIT	Non-reserved
WARNINGS	Non-reserved
WEEK	Non-reserved
WEIGHT_STRING	Non-reserved
WHEN	Reserved
WHERE	Reserved
WHILE	Reserved
WINDOW	Reserved
WITH	Reserved
WITHIN	Non-reserved
WITHOUT	Non-reserved
WORK	Non-reserved
WRAPPER	Non-reserved
WRITE	Reserved
X509	Non-reserved
XA	Non-reserved
XID	Non-reserved
XML	Non-reserved
XOR	Reserved
YEAR	Non-reserved (cannot be functions or types)
YEAR_MONTH	Reserved

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Keyword	Type
ZEROFILL	Reserved
ZONE	Non-reserved

4.3 Character Sets and Collations

A character set provides character encoding rules, and a collation provides character sorting rules. This section describes the character sets and collations in M-compatible databases. The following character sets, collation rules, and syntax are supported only in M-compatible databases.

- The character sets and collations of M-compatible databases must meet the following requirements:
  - Each character set has one or more collations and has only one default collation.
  - Each collation has only one associated character set.
  - The sorting results of the same data using different collations may be different.
  - utf8mb4 and utf8 are the same character set.
  - You are advised to select the same character set for table columns and server_encoding to avoid performance loss caused by transcoding.
- The character sets and collations of M-compatible databases support the following functions:
  - Multiple character sets can be used to store character strings.
  - Collations can be used to compare character strings.
  - Database-level, schema-level, table-level, and column-level character sets and collations are supported.

NOTE

Character sets other than the BINARY character set and the character sets of the current database cannot be used in the same database.

Supported Character Sets

The following character sets are supported by M-compatible databases.

Table 4-2 Character sets supported by M-compatible databases

Character Set	Description	Default Collation
utf8	Variable-length Unicode character code. The length of the character code ranges from 1 to 4 bytes.	utf8mb4_general_ci

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Character Set	Description	Default Collation
utf8mb4	The character set is the same as that of utf8.	utf8mb4_general_ci
gbk	Extended character set of Chinese characters in the GB standard.	gbk_chinese_ci
gb18030	Character set of Chinese characters in the GB standard.	gb18030_chinese_ci
binary	Binary pseudo-character set.	binary

NOTE

- Currently, the database-level, schema-level, table-level, and column-level syntax supports only character sets specified in **Table 4-2**.
- The BINARY character set is implemented by using the existing character set SQL_ASCII.
- The conversion logic between character sets in GaussDB is different from that in MySQL. Therefore, some special characters that can be converted in M-compatible databases may fail to be converted in MySQL. You are advised not to use these uncommon characters.
- Currently, GaussDB does not perform strict encoding logic verification on invalid characters that do not belong to the current character set. As a result, such invalid characters may be successfully entered.

Supported Collations

The following collations are supported by M-compatible databases.

Table 4-3 Collations supported by M-compatible databases

Collation	Character Set	Description	Blank Filling
utf8_bin	utf8	The binary collation is used.	Supported.
utf8_general_ci	utf8	The general collation is used.	Supported.
utf8_unicode_ci	utf8	The Unicode Collation Algorithm (UCA)-based collation is used.	Supported.
utf8mb4_bin	utf8mb4	Same as utf8_bin.	Supported.
utf8mb4_general_ci	utfbmb4	Same as utf8_general_ci.	Supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Collation	Character Set	Description	Blank Filling
utf8mb4_unicode_ci	utf8mb4	Same as utf8_unicode_ci.	Supported.
utf8mb4_0900_ai_ci	utf8mb4	The Unicode Collation Algorithm (UCA)-based collation is used.	Supported.
gbk_bin	gbk	The binary collation is used.	Supported.
gbk_chinese_ci	gbk	The Chinese (pinyin) collation is used.	Supported.
gb18030_bin	gb18030	The binary collation is used.	Supported.
gb18030_chinese_ci	gb18030	The Chinese (pinyin) collation is used.	Supported.
binary	binary	The binary collation is used.	Not supported.

NOTE

- Collation names start with the names of the character sets associated with them and are usually followed by one or more suffixes that indicate other character order features. For example, **_bin** indicates binary sorting rules, and **_ci** indicates case-insensitive.
- If a collation supports blank filling, spaces at the end of character strings are ignored during comparison, for example, **:'A' = 'A'**.

4.3.1 Character Sets and Collations of the Client Connection

When data containing character set attributes is transmitted between the server and client, coding conversion is automatically performed. After receiving an SQL statement from a client, the server converts the character set of the SQL statement from the client character set client_encoding to the database character set server_encoding. Before the query result data is sent to the client, the data is encoded and converted to the character_set_results character set of the client.

System parameters are described as follows:

- server_encoding

  Character set specified when an M-compatible database is created. For details, see **Creating an M-compatible Database and User**.
- client_encoding

  Character set of the client. You can run the **SET NAMES** statement to change the value. For details, see **SET**.
- character_set_connection

  Default character set applied when no character set is specified for character string constants in SQL statements.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- collation_connection

  Default collation applied when no collation is specified for character string constants in SQL statements.

- character_set_results

  Character set of the returned result.

  NOTE

  - For an expression that converts a non-character type object to a character type, the result character set and collation are **character_set_connection** and **collation_connection**.
  - By default, the character set and collation of a bound parameter of the character type are the values of **character_set_connection** and **collation_connection**, respectively.
  - During character data conversion, the character code is verified. If the character code does not meet the requirements, the "invalid byte sequence for encoding xxx" error is reported.

4.3.2 Database-level Character Sets and Collations

Create a database or schema and specify the default character set and collation.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_option] ...
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Modify the default character set and collation attributes of the database or schema.

```
ALTER {DATABASE | SCHEMA} db_name
    [create_option] ...
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Syntax description:

- **db_name**

  Specifies the database or schema name.

  Value range: a string. It must comply with the identifier naming convention.

- **default_charset**

  Specifies the default character set of a database or schema. If you specify a character set separately, the default collation of the database or schema is set to the default collation of the specified character set.

- **default_collation**

  Specifies the default collation of a database or schema. If you specify a collation separately, the default character set of the database or schema is set to the character set corresponding to the specified collation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

GaussDB selects a character set and collation of a database or schema in the following ways:

- If both **default_charset** and **default_collation** are set, the character set **default_charset** and collation **default_collation** are used. In addition, **default_charset** and **default_collation** must correspond to each other. Otherwise, an error is reported.
- If only **default_charset** is set, the character set **default_charset** and its default collation are used.
- If only **default_collation** is set, the collation **default_collation** and its corresponding character set are used.
- If neither **default_charset** nor **default_collation** is set, the character set of the database or scheme is the same as server_encoding, and the collation is the default collation of the character set.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in **Supported Collations** support **default_collation**. If another collation is specified, an error is reported.
- Except BINARY character sets, the character set of a new library or schema must be server_encoding of the database.

Examples:

```
-- Set only the character set. The collation is the default collation of the character set.
m_db=# CREATE SCHEMA test CHARSET utf8;

-- Set only the collation. The character set is the character set associated with the collation.
m_db=# CREATE SCHEMA test COLLATE utf8_bin;

-- Set both the character set and collation. The character set and collation must correspond to each other.
m_db=# CREATE SCHEMA test CHARSET utf8 COLLATE utf8_bin;

-- Change the default character set of test to utf8mb4 and the default collation to utf8mb4_bin.
m_db=# ALTER SCHEMA test CHARSET utf8mb4 COLLATE utf8mb4_bin;

-- Drop the table.
m_db=# DROP SCHEMA test;
```

4.3.3 Table-level Character Sets and Collations

Set the default character set and default collation for a table.

```
CREATE TABLE table_name (column_list)
    [ [DEFAULT] {CHARACTER SET | CHAR SET | CHARSET}[ = ] default_charset ]
    [ [DEFAULT] COLLATE [ = ] default_collation ]
```

Modify the default character set and collation of a table. The modification does not affect the existing columns in the table.

```
ALTER TABLE table_name
    [ [DEFAULT] {CHARACTER SET | CHAR SET | CHARSET}[ = ] default_charset ]
    [ [DEFAULT] COLLATE [ = ] default_collation ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

Currently, you cannot modify the default character set and collation of a table to the specified values, set the character set and collation of all columns with character type to the specified value, or convert the data in the column to new character set encoding. The syntax is as follows:

```
ALTER TABLE table_name
  CONVERT TO CHARACTER SET | CHARSET charset [ COLLATE collation ]
```

Parameter description

- **table_name**

  Specifies the name of the table.

- **default_charset**

  Specifies the default character set of a table. If you specify a character set separately, the default collation of the table is set to the default collation of the specified character set.

- **default_collation**

  Specifies the default collation of a table. If you specify a collation separately, the default character set of the table is set to the character set corresponding to the specified collation.

GaussDB selects a character set and collation of a table in the following ways:

- If both **default_charset** and **default_collation** are set, the character set **default_charset** and collation **default_collation** are used. In addition, **default_charset** and **default_collation** must correspond to each other. Otherwise, an error is reported.
- If only **default_charset** is set, the character set **default_charset** and its default collation are used.
- If only **default_collation** is set, the collation **default_collation** and its corresponding character set are used.
- If neither **default_charset** nor **default_collation** is set, the default character set and collation of the schema where the table is located are used as the default character set and collation of the table.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in **Supported Collations** support **default_collation**. If another collation is specified, an error is reported.
- If the default collation of a table is BINARY, the text types whose collation is not specified in the table are converted to the corresponding binary type, and the collation is set to BINARY.
- Except BINARY character sets and collations, the default character set of a table cannot be different from the database-level character set and collation.
- Currently, the data of columns of the character type in the table cannot be converted into encoding.
- Do not create a foreign key when the collation of the foreign key is different from that of the column referenced by the foreign key.
- If a table or column has a foreign key or foreign key reference column, the character set or collation cannot be converted or modified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
-- Set only the character set. The collation is the default collation of the character set.
m_db=# CREATE TABLE test(c1 text) CHARSET utf8;

-- Set only the collation. The character set is the character set associated with the collation.
m_db=# CREATE TABLE test(c1 text) COLLATE utf8_bin;

-- Set both the character set and collation. The character set and collation must correspond to each other.
m_db=# CREATE TABLE test(c1 text)  CHARSET utf8 COLLATE utf8_bin;

-- Modify the default character set of the table to utf8mb4 and the default collation to utf8mb4_bin.
m_db=# ALTER TABLE test CHARSET utf8mb4 COLLATE utf8mb4_bin;

-- Drop the table.
m_db=# DROP TABLE test;
```

4.3.4 Column-level Character Sets and Collations

You can set the character set and collation for each column of the string type (CHAR, VARCHAR, or TEXT).

```
CREATE TABLE table_name (
    column_name data_type
    [ {CHARACTER SET | CHAR SET | CHARSET} charset ]
    [ COLLATE collation ]);
```

Syntax description:

- **table_name**

  Table name.

- **data_type**

  Data type of a column. The value can be character set and collation syntax.

- **CHARACTER SET | CHAR SET | CHARSET charset**

  Character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

- **COLLATE collation**

  The COLLATE clause specifies the collation of a column (the data type of the column must support collation). If no collation is specified, the default collation is used.

Setting Method

M-compatible selects character sets and collations of a table column in the following ways:

- If both **charset** and **collation** are specified, **charset** and **collation** are used. **charset** and **collation** must correspond to each other. Otherwise, an error is reported.
- If only **charset** is specified, the character set **charset** and its default collation are used.
- If only **collation** is specified, the character set associated with collation and the specified collation are used.
- If neither **charset** nor **collation** is specified, the default character set and collation of the table are used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in **Supported Collations** support **default_collation**. If another collation is specified, an error is reported.
- If the collation of a column is BINARY, the text types whose collation is not specified in the table are converted to the corresponding binary type, and the collation is set to BINARY.
- Except BINARY character sets and collations, the default character set of a column cannot be different from the table-level character set and collation.
- The character set of the partition key of the partitioned table must be the same as that of the database.

Examples:

```
-- Set only the character set. The collation is the default collation of the character set.
m_db=# CREATE TABLE test(c1 text CHARSET utf8);

-- Set only the collation. The character set is the character set associated with the collation.
m_db=# CREATE TABLE test(c1 text COLLATE utf8_bin);

-- Set both the character set and collation. The character set and collation must correspond to each other.
m_db=# CREATE TABLE test(c1 text CHARSET utf8 COLLATE utf8_bin);

-- Drop the table.
m_db=# DROP TABLE test;
```

4.3.5 Character Sets and Collations of Expressions of the String Type

Each expression of the string type contains character sets and collation attributes. The default character sets and collations of string constants are determined by the system parameters **character_set_connection** and **collation_connection**.

Character set syntax:

```
[_charset_name]'string'
```

Currently, the character set syntax for specifying string constants is not supported in GaussDB.

Collation syntax:

You can also specify the collation for expressions of other character string types.

```
expression [COLLATE collation_name]
```

Syntax description:

COLLATE collation_name

Collation name, which is the collation attribute of the string.

- The data type of the expression must be a data type that supports collations.
- The specified collation must be supported by the character set of the expression.

Examples:

```
-- Use the collate statement to specify the collation.
m_db=# SELECT 'a' COLLATE utf8mb4_general_ci = 'A';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.3.6 Rules for Combining Character Sets and Collations

Expressions with different character sets and collations are processed based on certain priorities to determine the collations used for character string comparison and the character sets of the expressions.

Collation Priority

The priorities of different expressions in descending order are as follows:

- The COLLATE syntax has the highest priority.
- Expressions with collation conflicts (for example, two character strings with different collations).
- Columns of data types that support collation, user-defined variables, stored procedure parameters, and CASE expressions.
- Specific system functions (such as version() and opengauss_version() function expressions).
- String constants and bound parameters.
- NULL expression.
- If a data type of an expression does not support collation, this expression has the lowest priority.

When the collations of two expressions are different, the collation of the expression with the highest priority is used.

Examples:

```
m_db=# CREATE TABLE t_utf8(c1 varchar(16) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin);
m_db=# INSERT INTO t_utf8 VALUES('STRING');

-- If the utf8mb4_bin collation is used for comparison, the result is false.
m_db=#  SELECT c1 = 'string' AS result FROM t_utf8;
 result
--------
 f
(1 row)

-- If the utf8mb4_general_ci collation is used for comparison, the result is true.
m_db=# SELECT c1 = 'string' COLLATE utf8mb4_general_ci AS result FROM t_utf8;
 result
--------
 t
(1 row)

-- The CASE expression is at the same level as the c1 column. Even if the expression contains an explicit collation, the collation of the c1 column is still used for comparison. The two collations are different. "same level" is displayed.
m_db=# SELECT CASE 'string' COLLATE utf8mb4_general_ci WHEN c1 THEN 'different level' ELSE 'same level' END AS result FROM t_utf8;
   result
------------
 same level
(1 row)

-- The IN subquery has the same level as the c1 column. Even if the expression contains an explicit collation, the collation of c1 is still used for comparison. The two collations are different.
m_db=# SELECT c1 FROM t_utf8 WHERE c1 in (SELECT 'string' COLLATE utf8mb4_general_ci);
 c1
----
(0 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
m_db=# DROP TABLE t_utf8;
```

If the collations of two expressions with the same priority are different, the following processing method is used:

- If the two character sets are the same, the collation suffixed with **_bin** is preferred.
- If the two character sets are the same, the default collation is not used.
- If the preceding conditions are not met, the two expressions are marked as a collation conflict, and the collations are marked as invalid.
  - If a conflict occurs because the COLLATE syntax specifies different collations for the same character set, an exception is displayed.
  - If the two conflicting collations are supported by M-compatible, an exception message is displayed.
  - If an invalid collation is used for sorting operations (such as > and <), an exception is displayed.
  - During equality comparison of character strings, if the collation is invalid, the character strings are compared as binary values.

Examples:

```
m_db=# CREATE TABLE t_utf8mb4_charset(
    c_utf8_bin varchar(16) character set utf8mb4 collate utf8mb4_bin,
    c_utf8_uni varchar(16) character set utf8mb4 collate utf8mb4_unicode_ci,
    c_utf8_gen varchar(16) character set utf8mb4 collate utf8mb4_general_ci);
m_db=# INSERT INTO t_utf8mb4_charset VALUES('STRING', 'String', 'string');

-- The utf8mb4_bin collation is preferentially used for comparison. The result is false.
m_db=# SELECT c_utf8_bin = c_utf8_uni FROM t_utf8mb4_charset;
 ?column?
----------
 f
(1 row)

-- If the collation conflicts, an exception message is displayed.
m_db=# SELECT c_utf8_uni = c_utf8_gen FROM t_utf8mb4_charset;
ERROR:  Collation mismatch between collations "utf8mb4_unicode_ci" and "utf8mb4_general_ci".
LINE 1: SELECT c_utf8_uni = c_utf8_gen FROM t_utf8mb4_charset;
                          ^
-- If the explicitly specified collation conflicts, an exception message is displayed.
m_db=# SELECT c_utf8_uni COLLATE utf8mb4_unicode_ci = c_utf8_gen COLLATE utf8mb4_general_ci FROM
t_utf8mb4_charset;
ERROR:  collation mismatch between explicit collations "utf8mb4_unicode_ci" and "utf8mb4_general_ci"
LINE 1: ..._utf8_uni COLLATE utf8mb4_unicode_ci = c_utf8_gen COLLATE ut...
                                                  ^
-- Drop the table.
m_db=# DROP TABLE t_utf8mb4_charset;
```

NOTE

- Only the character sets of objects and expressions of the string type (excluding char and name) can be different from those of the database.
- According to the rules for combining character sets and collations, the character sets corresponding to C, POSIX, and DEFAULT collations are server_encoding.

4.4 SQL Syntax

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.1 Identifier Description

The M-compatible naming rules for identifiers are as follows:

- Identifiers support backquotes (``) by default.
  - When **sql_mode** is set to **ANSI_QUOTES**, identifiers support backquotes (``) and double quotation marks ("").
  - In special cases, quotation marks can be used to avoid special character errors.
- Characters allowed in unquoted identifiers are as follows:
  - ASCII: letters, digits, underscores (_), dollar signs ($), and number signs (#)
  - Extended characters: U+0080 to U+00FF
  - Only letters, digits, underscores (_), and extended characters from U +0080 to U+00FF are allowed at the beginning.
- Characters allowed in quoted identifiers are as follows:
  - ASCII: U+0001 to U+007F
  - Extended characters: U+0080 to U+FFFF
- Identifiers cannot be U+0000 characters or characters greater than U+10000.
- An identifier can start with a digit but cannot consist of only digits unless it is enclosed in quotation marks.

The case sensitivity of M-compatible identifiers is as follows:

- When the GUC parameter **lower_case_table_names** is set to **0**, case-sensitive identifiers include databases, schemas, tables, and views. These identifiers must be case-sensitive during creation and use.
- When the GUC parameter **lower_case_table_names** is set to **1**, databases, schemas, tables (including the sequence names generated by the **auto_increment** column), and views are case-insensitive during creation and use. In addition, when backquotes or double quotation marks are used to qualify these objects, the objects are still case-insensitive.
- Column names are case-sensitive when they are stored and are case-insensitive when they are used for comparison. Column names are stored as they are in the system and are case-insensitive when they are used.
- Other identifiers:
  - They are lowercase in the system. If an identifier is enclosed in double quotation marks (**SQL_MODE** must be set to **ANSI_QUOTES**) or backquotes, it is case-sensitive.
  - If a schema or table is specified and the GUC parameter **lower_case_table_names** is set to **0**, the schema and table names are case-sensitive. If the GUC parameter **lower_case_table_names** is set to **1**, the schema and table names are case-insensitive.

4.4.2 SQL Statements

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.1 SQL Syntax

Table 4-4 SQL syntax

Format	Description
[ ]	The part enclosed in brackets ([]) is optional.
...	Preceding elements can appear repeatedly.
[x | y | ...]	One item is selected from two or more options or no item is selected.
{ x | y | ... }	One item is selected from two or more options.
[x | y | ...] [...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with spaces.
[x | y | ...] [,...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with commas (,).
{ x | y | ... } [...]	At least one parameter can be selected. If multiple parameters are selected, separate them with spaces.
{ x | y | ... } [,...]	At least one parameter can be selected. If multiple parameters are selected, separate them with commas (,).

4.4.2.2 DCL Syntax Overview

Data control language (DCL) is used to set or change the permissions of database users or roles.

Granting Rights

M compatibility provides a statement for granting rights to data objects and roles. For details, see **14.2.5.2.12.1-GRANT**.

Revoking Rights

M compatibility provides a statement for revoking rights. For details, see **14.2.5.2.17.9-REVOKE**.

4.4.2.3 DDL Syntax Overview

Data definition language (DDL) is used to define or change an object in a database, such as a table, an index, or a view.

NOTE

M compatibility does not support DDL operations when the CN is incomplete. For example, if a CN in the cluster is faulty, creating a database or a table will fail.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Defining a Role

A role is used to manage permissions. For database security, management and operation permissions can be granted to different roles. **Table 4-5** lists the involved SQL statements.

Table 4-5 SQL statements for defining a role

Function	SQL Statement
Creating a role	**CREATE ROLE**
Altering role attributes	**ALTER ROLE**
Dropping a role	**DROP ROLE**
Deleting the database objects owned by a database role	**DROP OWNED**

Defining a User

A user is used to log in to a database. Different permissions can be granted to users for managing data accesses and operations of the users. **Table 4-6** lists the involved SQL statements.

Table 4-6 SQL statements for defining a user

Function	SQL Statement
Creating a user	**CREATE USER**
Altering user attributes	**ALTER USER**
Dropping a user	**DROP USER**

Defining a Database or Schema

In M compatibility, database and schema are synonyms. A database or schema is the warehouse for organizing, storing, and managing data. Defining a database includes: creating a database, altering the database attributes, and dropping the database. **Table 4-7** lists the involved SQL statements.

Table 4-7 SQL statements for defining a database

Function	SQL Statement
Creating a database or schema	**CREATE DATABASE** and **CREATE SCHEMA**
Modifying a database or schema attribute	**ALTER DATABASE** and **ALTER SCHEMA**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	SQL Statement
Deleting a database or schema	**DROP DATABASE** and **DROP SCHEMA**
Specifying the current mode	**USE**

Defining a Table

A table is a special data structure in a database and is used to store data objects and relationship between data objects. **Table 4-8** lists the involved SQL statements.

Table 4-8 SQL statements for defining a table

Function	SQL Statement
Creating a table	**CREATE TABLE**
Altering table attributes	**ALTER TABLE**
Dropping a table	**DROP TABLE**

Defining a Partitioned Table

A partitioned table is a logical table used to improve query performance and does not store data (data is stored in ordinary tables). **Table 4-9** lists the involved SQL statements.

Table 4-9 SQL statements for defining a partitioned table

Function	SQL Statement
Creating a partitioned table	**CREATE TABLE PARTITION**
Create a partition	**ALTER TABLE PARTITION**
Altering partitioned table attributes	**ALTER TABLE PARTITION**
Dropping a partition	**ALTER TABLE PARTITION**
Dropping a partitioned table	**DROP TABLE**

Defining an Index

An index indicates the sequence of values in one or more columns in a database table. It is a data structure that improves the speed of data access to specific information in a database table. **Table 4-10** lists the involved SQL statements.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-10 SQL statements for defining an index

Function	SQL Statement
Creating an index	**CREATE INDEX**
Altering index attributes	**ALTER INDEX**
Dropping an index	**DROP INDEX**
Rebuilding an index	**REINDEX**

Defining a Plug-in Extension

This feature is for internal use only. You are advised not to use it.

Table 4-11 SQL statements for defining a plug-in extension

Function	SQL Statement
Creating a plug-in extension	**CREATE EXTENSION**
Modifying a plug-in extension	**ALTER EXTENSION**
Dropping a plug-in extension	**DROP EXTENSION**

Defining a User Group

Table 4-12 SQL statements for defining a user group

Function	SQL Statement
Creating a user group	**CREATE GROUP**
Modifying attributes of a user group	**ALTER GROUP**
Deleting a user group	**DROP GROUP**

Defining a Node

Table 4-13 SQL statements for defining a node

Function	SQL Statement
Creating a cluster node	**CREATE NODE**
Modifying the definition of an existing node	**ALTER NODE**
Deleting a node	**DROP NODE**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Defining a Node Group

Table 4-14 SQL statements for defining a node group

Function	SQL Statement
Creating a cluster node group	CREATE NODE GROUP
Modifying information about an existing node group	ALTER NODE GROUP
Deleting a node group	DROP NODE GROUP

Defining a Sequence

Table 4-15 SQL statements for defining a sequence

Function	SQL Statement
Adding a sequence to the current database	CREATE SEQUENCE
Modifying parameters of an existing sequence	ALTER SEQUENCE
Deleting a sequence from the current database	DROP SEQUENCE

Defining a Resource Label

Table 4-16 SQL statements for defining a resource label

Function	SQL Statement
Creating a resource label	CREATE RESOURCE LABEL
Altering a resource label	ALTER RESOURCE LABEL
Dropping a resource label	DROP RESOURCE LABEL

Defining an Audit Policy

Table 4-17 SQL statements for defining an audit policy

Function	SQL Statement
Creating a unified audit policy	CREATE AUDIT POLICY

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Function	SQL Statement
Altering a unified audit policy	ALTER AUDIT POLICY
Dropping an audit policy	DROP AUDIT POLICY

Defining a View

Table 4-18 SQL statements for defining a view

Function	SQL Statement
Creating a view	CREATE VIEW
Modifying the auxiliary attributes of a view	ALTER VIEW
Forcibly deleting a view from the database	DROP VIEW

Defining a Function

Table 4-19 SQL statements for defining a function

Function	SQL Statement
Creating a function	CREATE FUNCTION
Deleting a function	DROP FUNCTION

Setting the nodeis_active Column of a Specified Node

ALTER COORDINATOR describes how to change the value of the **nodeis_active** column of a specified node in the pgxc_node system catalog.

Collecting Statistics

For details about how to collect statistics related to the contents of ordinary tables in the database, see ANALYZE.

Clearing a Recycle Bin

M compatibility provides statements for clearing a recycle bin. For details, see PURGE.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Defining an Object Comment

M compatibility supports statements for defining or modifying an object comment. For details, see **COMMENT**.

Creating a Table Based on Query Results

M compatibility supports statements for creating a table based on query results and inserting queried data into the target table. For details, see **SELECT INTO**.

Clearing Table Data

M compatibility supports statements for quickly deleting all rows from a table. For details, see **TRUNCATE**.

Recycling Storage Space

M compatibility supports statements for recycling storage space occupied by deleted rows in a table or B-tree index. For details, see **VACUUM**.

4.4.2.4 DML Syntax Overview

Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.

Inserting Data

Inserting data refers to adding one or multiple records to a database table. For details, see **INSERT**.

Updating Data

Updating data refers to modifying one or multiple records in a database table. For details, see **UPDATE**.

Querying Data

The database query statement SELECT is used to search required information in a database. For details, see **SELECT**.

Deleting Data

M compatibility provides statements for deleting data that meets specified conditions from a table. For details, see **DELETE**.

Copying Data

M compatibility provides a statement for copying data between tables and files. For details, see **COPY**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Locking

M compatibility provides multiple lock modes to control concurrent accesses to table data. For details, see **LOCK**.

Session Management

A session is a connection established between the user and the database. For details about how to modify a session, see **ALTER SESSION**.

Executing an Anonymous Code Block

M compatibility provides statements for executing an anonymous code block. For details, see **DO**.

Inserting or Replacing Data

M compatibility provides statements for inserting or replacing data in a table. For details, see **REPLACE**.

Importing Data

GaussDB provides a statement for importing data from a file to a specified table in the database. For details, see **LOAD DATA**.

4.4.2.5 Overview of Other Syntax

In addition to DCL, DDL, and DML syntax, M-compatible also provides syntax for other functions.

Showing an Execution Plan of an SQL Statement

M compatibility provides statements for showing an execution plan of an SQL statement. For details, see **EXPLAIN**.

Collecting Table Content Statistics

M compatibility provides statements for collecting statistics about the table content in the database. For details, see **ANALYZE**.

Clearing the Storage Space

M compatibility provides statements for recycling storage space occupied by deleted rows in a table or B-Tree index. For details, see **VACUUM**.

Creating a Prepared Statement

M compatibility provides statements for creating a prepared statement. For details, see **PREPARE**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Prepared Statements

Table 4-20 SQL statements related to prepared statements

Function	SQL Statement
Executing a prepared statement	**EXECUTE**
Deallocating a prepared statement	**DEALLOCATE** and **DROP PREPARE**

SQL Statements Related to Transactions

Table 4-21 SQL statements related to transactions

Function	SQL Statement
Rolling back the current transaction and canceling the changes in the transaction	**ROLLBACK**
Starting a transaction	**BEGIN**
	SET TRANSACTION
	START TRANSACTION
Preparing the current transaction for two-phase commit	**PREPARE TRANSACTION**
Committing the current transaction	**COMMIT**
Committing a prepared two-phase transaction	**COMMIT PREPARED**
Deleting a savepoint previously defined for the current transaction	**RELEASE SAVEPOINT**
Rolling back to a savepoint	**ROLLBACK TO SAVEPOINT**
Canceling a transaction ready for two-phase commit	**ROLLBACK PREPARED**
Creating a savepoint in the current transaction	**SAVEPOINT**
Setting a checkpoint	**CHECKPOINT**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Modifying, Displaying, and Restoring a GUC Parameter

Table 4-22 SQL statements for modifying, displaying, and restoring a GUC parameter

Function	SQL Statement
Modifying a GUC parameter	**SET**
Showing the current value of a GUC parameter	**SHOW**
Restoring a GUC parameter to the default value	**RESET**

Setting a User Identifier

Table 4-23 SQL statements for setting a user identifier

Function	SQL Statement
Setting the current user identifier of the current session	**SET ROLE**
Setting the session user identifier and the current user identifier of the current session to a specified user	**SET SESSION AUTHORIZATION**

4.4.2.6 A

4.4.2.6.1 ALTER AUDIT POLICY

Description

Alters a unified audit policy. Currently, the distributed mode is not supported.

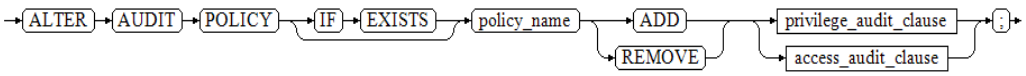
Precautions

- Only the user with the POLADMIN or SYSADMIN permission, or initial user has the permission to maintain audit policies.
- The unified audit policy takes effect only after **enable_security_policy** is set to **on**.

Syntax

Add or delete an operation type in the audit policy.

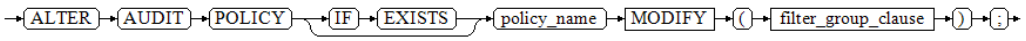
```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]
[ access_audit_clause ] };
```



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

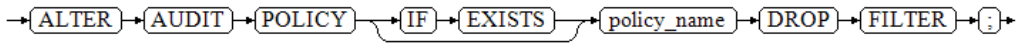
Modify the filter criteria in the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
```



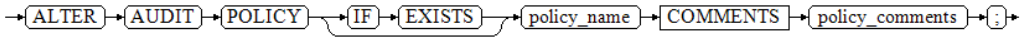
Delete the filter criteria from the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```



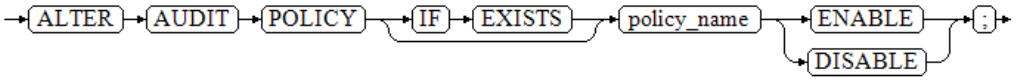
Change the description of the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```



Enable or disable the audit policy.

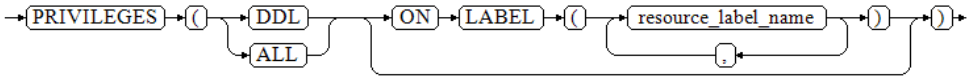
```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```



- privilege_audit_clause

  DDL operation type and target resource label in the audit policy.

  ```
  PRIVILEGES ({ DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```

- access_audit_clause

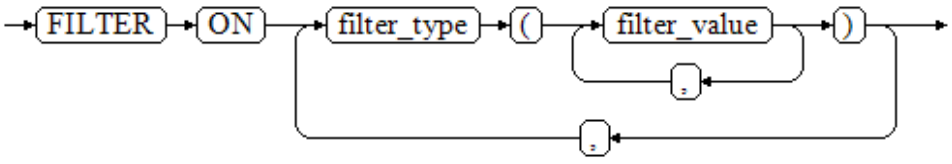
  DML operation type and target resource label in the audit policy.

  ```
  ACCESS ({ DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ])
  ```


- filter_group_clause

  Filter criteria in the audit policy.

  ```
  FILTER ON { filter_type ( filter_value [, ... ] ) } [, ... ]
  ```

- DDL

  Perform audit when the following operations are performed on the database.

  ```
  { ( ALTER | ANALYZE | COMMENT | CREATE | DROP | GRANT | REVOKE | SET | SHOW ) }
  ```

- DML

  Perform audit when the following operations are performed on the database.

  ```
  { ( COPY | DEALLOCATE | DELETE | EXECUTE | REINDEX | INSERT | PREPARE | SELECT | TRUNCATE |
  UPDATE ) }
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the **naming conventions**.

- **resource_label_name**

  Specifies the resource label name.

- **DDL**

  Specifies the operations that will be audited within the database: ALTER, ANALYZE, COMMENT, CREATE, DROP, GRANT, REVOKE, SET, and SHOW.

- **DML**

  Specifies the operations that are audited within the database: COPY, DEALLOCATE, DELETE_P, EXECUTE, REINDEX, INSERT, PREPARE, SELECT, TRUNCATE, and UPDATE.

- **ALL**

  Specifies all operations supported by the specified DDL or DML statements in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.

- **filter_type**

  Specifies the types of information to be filtered by the policy: **IP**, **ROLES**, and **APP**.

- **filter_value**

  Specifies the detailed information to be filtered.

- **policy_comments**

  Records description information of the audit policy.

- **ENABLE|DISABLE**

  Enables or disables the unified audit policy.

Examples

- Add or delete an operation type in the audit policy.

```
-- Create audit policy adt1 for executing CREATE on the database.
m_db=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- Add DROP to the adt1 audit policy.
m_db=# ALTER AUDIT POLICY adt1 ADD PRIVILEGES (DROP);

-- Delete DROP from the adt1 audit policy.
m_db=# ALTER AUDIT POLICY adt1 REMOVE PRIVILEGES (DROP);
```

- Change the comments of the audit policy.

```
-- Change the comment of the adt1 audit policy to adt1_comments.
m_db=#  ALTER AUDIT POLICY adt1 COMMENTS 'adt1_comments';
```

- Change the filter information of the audit policy.

```
-- Create a user bob_audit.
m_db=#  CREATE USER bob_audit PASSWORD '********';

-- Change the filtering user of the adt1 audit policy to bob_audit.
m_db=#  ALTER AUDIT POLICY adt1 MODIFY (FILTER ON (ROLES(bob_audit)));

-- Delete user bob_audit.
m_db=# DROP USER bob_audit;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Delete the filter criteria from the audit policy.

```
-- Delete the filter criteria in the adt1 audit policy.
m_db=#  ALTER AUDIT POLICY adt1  DROP FILTER;
```

- Disable the audit policy.

```
-- Disable the adt1 audit policy.
m_db=#  ALTER AUDIT POLICY adt1  DISABLE;

-- Delete the adt1 audit policy.
m_db=# DROP AUDIT POLICY adt1;
```

Helpful Links

CREATE AUDIT POLICY and DROP AUDIT POLICY

4.4.2.6.2 ALTER COORDINATOR

Description

Changes the CN status. You can run the SQL statement on any normal CN in the cluster to change the value of the **nodeis_active** field of a specified node in the pgxc_node system catalog. You can also specify the nodes on which the system catalog is modified.

Precautions

- ALTER COORDINATOR is a statement used to modify the system catalog. Only the administrator and users in internal maintenance mode (for example, CM administrator) can execute this statement. This statement is dedicated for the CN removal feature and must be used together with other operations. You are advised not to run it by yourself.
- After this statement is executed, **SELECT reload_active_coordinator()** needs to be called to update the connection pool information of the node on which the system catalog is modified.

Syntax

```
ALTER COORDINATOR nodename SET status
    WITH (nodename1[, nodename2, nodename3 ...]);
```

Parameters

- **nodename**

  Specifies the CN name corresponding to a row of records in the pgxc_node system catalog. After the node name is specified, the value of **nodeis_active** in the record is changed.

  Value range: a string. Only CNs are supported. Ensure that the node name has a corresponding record in the pgxc_node system catalog.

- **status**

  Specifies the CN status, that is the updated value of **nodeis_active** in the pgxc_node system catalog.

  Value range:

  - FALSE

  - TRUE

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- ON
- OFF

- **nodename1[, nodename2, nodename3 ...]**

  Specifies the range of nodes on which the SQL statement is executed. When **ALTER COORDINATOR** is executed, the SQL statement is automatically delivered to all nodes in the range. The current execution node must be included.

  Value range: a string. Only CNs are supported. Ensure that the node name has a corresponding record in the pgxc_node system catalog and the node is normal. Otherwise, the SQL statement fails to be executed.

Examples

- Remove a faulty CN.

```
-- The cluster has three CNs: cn_5001, cn_5002, and cn_5003, which are working properly. If cn_5001 is faulty and the removal time meets the requirement, cn_5001 needs to be removed from the cluster.
gaussdb=# ALTER COORDINATOR cn_5001 SET False WITH (cn_5002,cn_5003);

-- Refresh the pgxc_node system catalog. On the cn_5002 and cn_5003 nodes, run the following SQL statement to update the value of nodeis_active corresponding to the cn_5001 record in the pgxc_node system catalog to false:
gaussdb=# SELECT reload_active_coordinator();

-- Check the CN status. The value of nodeis_active changes to f (false).
gaussdb=#  SELECT nodeis_active,node_name FROM pgxc_node where node_name='cn_5001';
 nodeis_active | node_name
---------------+-----------
 f             | cn_5001
(1 row)
```

- Restore the CN whose fault has been rectified.

```
-- After the cn_5001 fault is rectified, add cn_5001 back to the cluster.
gaussdb=#  ALTER COORDINATOR cn_5001 SET True WITH (cn_5002,cn_5003);

-- Refresh the pgxc_node system catalog. On the cn_5002 and cn_5003 nodes, run the following SQL statement to update the value of nodeis_active corresponding to the cn_5001 record in the pgxc_node system catalog to true:
gaussdb=# SELECT reload_active_coordinator();

-- Check the CN status. The value of nodeis_active changes to t (true).
gaussdb=# SELECT nodeis_active,node_name FROM pgxc_node where node_name='cn_5001';
 nodeis_active | node_name
---------------+-----------
 t             | cn_5001
(1 row)
```

4.4.2.6.3 ALTER DATABASE

Description

In the M-compatible database, DATABASE and SCHEMA are synonyms that allow you to modify the character set, collation, and owner of a database or schema.

Precautions

- Only the owner of a schema or a user granted the ALTER permission for a schema has the permission to run the **ALTER DATABASE** command. When separation of duties is disabled, a system administrator has this permission by default. To change the owner of a schema, you must be the owner of the schema or system administrator, and also a member of the new owner role.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- For system schemas other than public, such as pg_catalog and sys, only the initial user is allowed to change the owner of a schema. Changing the names of the built-in system schemas may make some functions unavailable or even affect the normal running of the database. By default, the names of the built-in system schemas cannot be changed. To ensure forward compatibility, you can change the names of the built-in system schemas only when the system is being started or upgraded or when **allow_system_table_mods** is set to **on**.
- Except the initial user, other users cannot change the owner of a schema to an O&M administrator.

Syntax

- Change the database owner.

```
ALTER DATABASE database_name
    OWNER TO new_owner;
```

- Modify the character set and collation of the database.

```
ALTER {DATABASE | SCHEMA} database_name
    [create_option] ...
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Parameters

- **database_name**

  Specifies the name of the database whose attributes are to be modified.

  Value range: a string complying with **Identifier Description**.

- **new_owner**

  Specifies the new owner of a database.

  Value range: a string indicating a valid username. For details about the username requirements, see **•user_name**.

- **COLLATE [=] default_collation**

  (Optional) Specifies the character set used by the database. For example, set this parameter by using **collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the character set of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

- **{CHAR SET | CHARSET | CHARACTER SET} [=] default_charset**

  (Optional) Specifies the character class used by the database. For example, set this parameter by using **CHARSET = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

Examples

- Modify the character set and collation of the database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a database test_db1.
m_db=# CREATE DATABASE test_db1;
-- Change the character set of test_db1 to utf8.
m_db=# ALTER DATABASE test_db1 CHARSET utf8;
-- Change the collation of test_db1 to utf8mb4_bin.
m_db=# ALTER DATABASE test_db1 COLLATE utf8mb4_bin;
```

- Change the database owner.

```
-- Create user scott.
m_db=# CREATE USER scott PASSWORD '********';
-- Change the owner of test_db1 to scott.
m_db=# ALTER DATABASE test_db1 OWNER TO scott;
```

Helpful Links

CREATE DATABASE and DROP DATABASE

4.4.2.6.4 ALTER EXTENSION

Description

Modifies plug-in extensions. Currently, this syntax is not supported. This feature is for internal use only. You are advised not to use it.

Precautions

ALTER EXTENSION modifies the definition of an installed extension. To use this API, set **support_extended_features** to **true**. Methods are as follows:

- UPDATE

  The extension is updated to a target version. The extension must be applicable to an update script (or a series of scripts) so that the current installation version can be modified to a required version.

- SET SCHEMA

  The extended object is moved to another schema. This extension must be relocatable to make the command successful.

- ADD member_object

  An existing object is added to the extension. This is mainly useful for extension update scripts. This object is then treated as a member of the extension. Obviously, the object can only be canceled by canceling the extension.

- DROP member_object

  A member object is removed from the extension. This method is useful in extension update scripts. The object is not canceled, but is simply separated from the extension.

  You must have an extension before using ALTER EXTENSION. You must have the permission on adding or deleting an object before using the ADD or DROP statement.

Syntax

- Modify the version of an extension.

```
ALTER EXTENSION name UPDATE [ TO new_version ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Modify the schema of the extension.

```
ALTER EXTENSION name SET SCHEMA new_schema;
```

- Add or delete member objects of the extension.

```
ALTER EXTENSION name { ADD | DROP } member_object;
```

The member object **member_object** is written as follows:

```
{
  CAST (source_type AS target_type)
  COLLATION object_name |
  FUNCTION function_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLE object_name |
  VIEW object_name}
```

Parameters

- **name**

  Specifies the name of an installed extension.

- **new_version**

  Specifies the latest version of the extension, which can be overridden by identifiers and string literals. If a target version of the extension is not specified, ALTER EXTENSION UPDATE updates to the default version shown in the extension's control file.

- **new_schema**

  Specifies the new schema of the extension.

- **object_name**

  agg_name

  function_name

  operator_name

  Specifies the names of objects that are added or removed from the extension, including names of tables, aggregations, domains, external linked lists, functions, operators, operator classes, operator families, sequences, text search objects, types, and views that can be schema-qualified.

- **agg_type**

  Specifies the input data type of the aggregate function. To reference a zero-parameter aggregate function, use ***** to replace the input data type list.

- **source_type**

  Specifies the name of the source data type to be forcibly converted.

- **target_type**

  Specifies the name of the target data type to be forcibly converted.

- **argmode**

  Specifies the model of the function parameter. The value can be **IN**, **OUT**, **INOUT**, or **VARIADIC**. The default value is **IN**. ALTER EXTENSION does not correlate to **OUT**, because you only need to enter parameters to confirm the consistency of functions. Therefore, **IN**, **INOUT**, and **VARIADIC** are enough.

- **argname**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the parameter name of the function. ALTER EXTENSION does not correlate to the parameter name. Only the parameter data type is required to confirm the consistency of the function.

- **argtype**

  Specifies the data type (optionally schema-qualified) of a function parameter.

- **left_type**

  right_type

  Specifies the data type (optionally schema-qualified) of an operator parameter. **NONE** is written for a missing parameter of a prefix or suffix operator.

Examples

Update the PL/pgSQL extension to version 2.0.

```
m_db=# ALTER EXTENSION plpgsql UPDATE TO '2.0';
```

Update the PL/pgSQL extension mode to utils.

```
m_db=# ALTER EXTENSION plpgsql SET SCHEMA utils;
```

Helpful Links

CREATE EXTENSION and **DROP EXTENSION**

4.4.2.6.5 ALTER GROUP

Description

Changes the role name or membership.

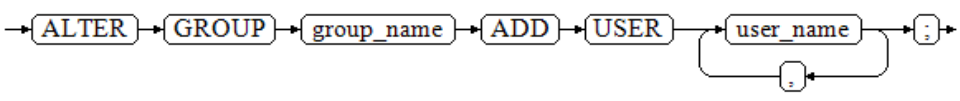
Precautions

- ALTER GROUP is not a standard SQL statement and therefore is not recommended.
- The ADD USER and DROP USER clauses are used to add users to or delete users from a user group. Any user can be considered a user group. The two clauses are equivalent to granting and recycling memberships respectively. Therefore, you are advised to use the GRANT or REVOKE statement.
- The RENAME TO clause changes the user group name, which is equivalent to naming roles using ALTER USER.

Syntax

- Add users to a group.

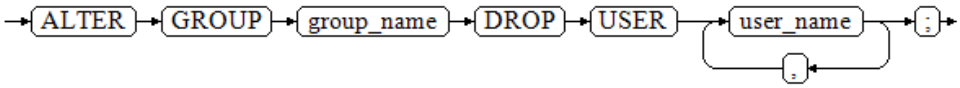
```
ALTER GROUP group_name
    ADD USER user_name [, ... ];
```



- Remove users from a group.

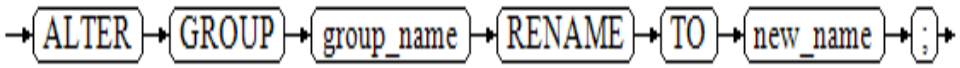
```
ALTER GROUP group_name
    DROP USER user_name [, ... ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.



- Change the name of the group.

```
ALTER GROUP group_name
    RENAME TO new_name;
```



Parameters

- **user_name**

  Specifies a role name.

  Value range: an existing role name. For details, see **•role_name**.

- **group_name**

  Specifies the ame of an existing user group.

  Value range: an existing role name. For details, see **•role_name**.

- **new_name**

  Specifies the name of a new role.

  Value range: a string complying with **Identifier Description**. For details about the role name requirements, see **•role_name**.

Examples

- Rename a user group.

```
-- Create a user group test.
m_db=# CREATE GROUP test PASSWORD '********';

-- Changes the username, which is equivalent to the role of ALTER GROUP RENAME.
m_db=# ALTER GROUP test RENAME TO tu_a1;
```

- Add users to or delete users from a user group.

```
-- Create users tu_a2 and tu_a3.
m_db=# CREATE ROLE tu_a2 PASSWORD '********';
m_db=# CREATE ROLE tu_a3 PASSWORD '********';

-- Add user tu_a2 to user group tu_a1.
m_db=# ALTER GROUP tu_a1 ADD USER tu_a2;

-- The preceding SQL statement is equivalent to the GRANT statement.
m_db=# GRANT tu_a1 TO tu_a3;

-- Query.
m_db=# SELECT groname, grolist FROM pg_group WHERE groname = 'tu_a1';
 groname |    grolist
---------+---------------
 tu_a1   | {25590,25593}
(1 row)
m_db=# SELECT rolname, oid FROM pg_roles WHERE oid IN (25590,25593);
 rolname |  oid
---------+-------
 tu_a2   | 25590
 tu_a3   | 25593
(2 rows)

-- Delete.
m_db=# DROP ROLE tu_a1,tu_a2,tu_a3;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Helpful Links

CREATE GROUP, **DROP GROUP**, and **ALTER ROLE**

4.4.2.6.6 ALTER INDEX

Description

Modifies the definition of an existing index.

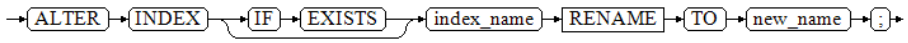
Precautions

- Only the index owner, a user who has the INDEX permission on the table where the index resides, or a user who has the ALTER ANY INDEX permission can run the **ALTER INDEX** command. When separation of duties is disabled, a system administrator has this permission by default.
- Do not keep a large number of invisible indexes on the same base table. Otherwise, the performance of DML operations such as INSERT, UPDATE, and DELETE may be affected.

Syntax

- Rename a table index.

```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME TO new_name;
```

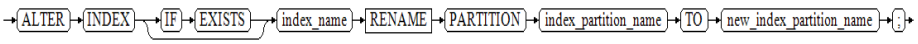


- Set the table index to be unavailable.

```
ALTER INDEX [ IF EXISTS ] index_name UNUSABLE;
```

- Rename an index partition.

```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME PARTITION index_partition_name TO new_index_partition_name;
```



- Change the storage parameter of a table index.

```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```

- Reset the storage parameter of a table index.

```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( {storage_parameter = value} [, ... ] );
```

Parameters

- **index_name**

  Specifies the name to be modified.

- **IF EXISTS**

  Sends a notice instead of an error if the specified index does not exist.

- **RENAME TO new_name**

  Changes only the name of the index. The stored data is not affected.

  new_name

  Specifies the new name of the index.

  Value range: a string complying with **Identifier Description**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **RENAME PARTITION index_partition_name TO new_index_partition_name**

  Renames an index partition.

- **new_index_partition_name**

  Specifies the new name of the index partition.

- **index_partition_name**

  Specifies the name of an index partition.

- **SET ({storage_parameter = value} [, ...])**

  Changes one or more index-method-specific storage parameters. Note that the index content will not be modified immediately by this statement. You may need to use REINDEX to rebuild the index based on different parameters to achieve the expected effect.

  – **storage_parameter**

    Specifies the name of an index-method-specific parameter. **ACTIVE_PAGES** indicates the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter because it is invalid in distributed mode. When ALTER is executed on GSI, the **INTERNAL_MASK**, **APPEND_MODE_INTERNAL**, **WAIT_CLEAN_GPI**, **PARTITION_DDL_FLAG**, **COLLATE**, **CROSSBUCKET**, and **DEDUPLICATION** parameters are not supported. The behavior is the same as that of UB-tree. The **FILLFACTOR**, and **INDEXSPLIT** parameters are supported. **ACTIVE_PAGES** and **WAIT_CLEAN_CBI** are not applicable to GSI.

  – **value**

    Specifies the new value for an index-method-specific storage parameter. This might be a number or a word depending on the parameter.

- **RESET ({ storage_parameter } [, ...])**

  Reset one or more index-method-specific storage parameters to the default value. Similar to the SET statement, REINDEX may be used to completely update the index.

Examples

- Rename an index.

```
-- Create the test1 table and create an index for it.
m_db=# CREATE TABLE test1(col1 INT, col2 INT);
m_db=# CREATE INDEX aa ON test1(col1);

-- Rename index aa to idx_test1_col1.
m_db=# ALTER INDEX aa RENAME TO idx_test1_col1;

-- Query the index information in the test1 table.
m_db=# SELECT tablename,indexname,tablespace FROM pg_indexes WHERE tablename = 'test1';
 tablename |   indexname    | tablespace
-----------+----------------+------------
 test1     | idx_test1_col1 |
(1 row)
```

- Rename an index partition.

```
-- Create the partitioned table test2.
m_db=# CREATE TABLE test2(col1 int, col2 int) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (100),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    PARTITION p2 VALUES LESS THAN (200)
);

-- Create a partitioned index.
m_db=# CREATE INDEX idx_test2_col1 ON test2(col1) LOCAL(
    PARTITION p1,
    PARTITION p2
);
-- Rename the index partition.
m_db=# ALTER INDEX idx_test2_col1 RENAME PARTITION p1 TO p1_test2_idx;
m_db=# ALTER INDEX idx_test2_col1 RENAME PARTITION p2 TO p2_test2_idx;

-- Query the partition name of the idx_test2_col1 index.
m_db=# SELECT relname FROM pg_partition WHERE parentid = 'idx_test2_col1'::regclass;
   relname
--------------
 p1_test2_idx
 p2_test2_idx
(2 rows)
```

Helpful Links

CREATE INDEX, **DROP INDEX**, and **REINDEX**

4.4.2.6.7 ALTER NODE

Description

Changes the definition of an existing cluster node.

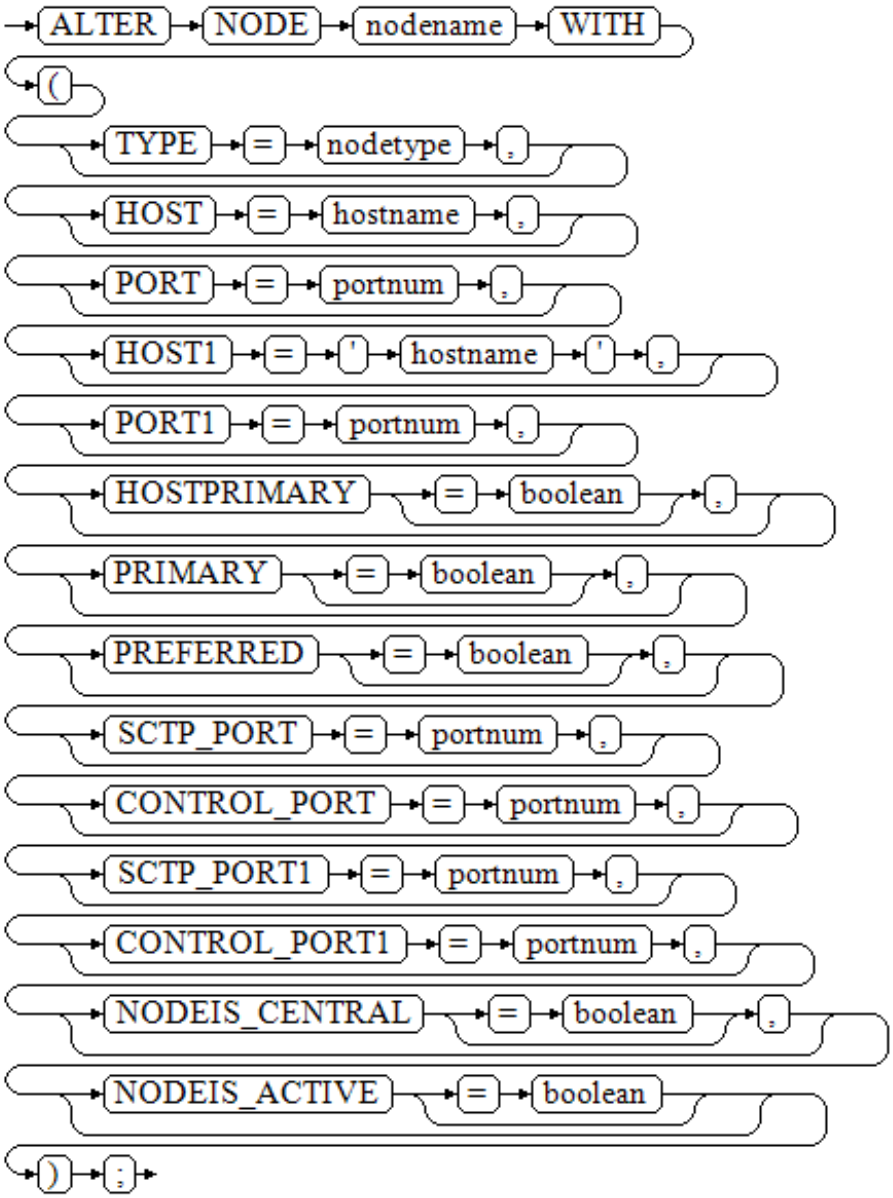
Precautions

ALTER NODE is an API of the cluster management tool and is used to manage clusters. Only administrators have the permission to use this API. You are advised not to use this API, because doing so affects the cluster.

Syntax

```
ALTER NODE nodename WITH
  (
    [ TYPE = nodetype,]
    [ HOST = hostname,]
    [ PORT = portnum,]
    [ HOST1 = 'hostname',]
    [ PORT1 = portnum,]
    [ HOSTPRIMARY [ = boolean ],]
    [ PRIMARY [ = boolean ],]
    [ PREFERRED [ = boolean ],]
    [ SCTP_PORT = portnum,]
    [ CONTROL_PORT = portnum,]
    [ SCTP_PORT1 = portnum,]
    [ CONTROL_PORT1 = portnum, ]
    [ NODEIS_CENTRAL [= boolean], ]
    [ NODEIS_ACTIVE [= boolean] ]
  );
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.



NOTE

The port whose number is specified by **PORT** is used for internal communications between nodes. Unlike the port connecting to an external client, it can be queried in the pgxc_node table.

Parameters

- **nodename**

  Specifies the node name.

  Value range: a string complying with **Identifier Description**.

- **TYPE = nodetype**

  Specifies the type of a node.

  Value range:

  – 'coordinator'

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - 'datanode'
- **HOST = hostname**

  Specifies the host name or IP address of a node.
- **PORT = portnum**

  Specifies the port number of the primary node to which a node is bound.
- **HOST1 = hostname**

  Specifies the name or IP address of the standby node corresponding to a node.
- **PORT1 = portnum**

  Specifies the port number of the standby node to which a node is bound.
- **HOSTPRIMARY**

  Specifies whether a host name or IP address is the primary one.

  Value range:

  - **true**
  - **false** (default value)
- **PRIMARY = boolean**

  Specifies whether the node is a primary node or not. A primary node allows read/write operations. A non-primary node allows only read operations.

  Value range:

  - **true**
  - **false** (default value)
- **PREFERRED = boolean**

  Specifies whether the node is a preferred node for read operations.

  Value range:

  - **true**
  - **false** (default value)
- **SCTP_PORT = portnum**

  Specifies the port used by the TCP proxy communications library of the primary node to listen on the data transmission channel. TCP is used to listen on connections.
- **CONTROL_PORT = portnum**

  Specifies the port used by the TCP proxy communications library of the primary node to listen on the control transmission channel. The TCP protocol is used to listen on connections.
- **SCTP_PORT1 = portnum**

  Specifies the port used by the TCP proxy communications library of the standby node to listen on the data transmission channel. TCP is used to listen on connections.
- **CONTROL_PORT 1= portnum**

  Specifies the port used by the TCP proxy communications library of the standby node to listen on the control transmission channel. The TCP protocol is used to listen on connections.
- **NODEIS_CENTRAL**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies whether the current node is a central control node. It is used only for CNs and is invalid for DNs.

- **true**
- **false** (default value)

- **NODEIS_ACTIVE**

  Specifies whether the current node is normal. It is used to mark whether the CN is removed and is invalid for DNs.

  - **true**
  - **false** (default value)

Examples

```
-- Create cluster nodes and execute them on all CNs.
gaussdb=# CREATE NODE datanode1 WITH(
   TYPE = datanode,
   PREFERRED = false
);

-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 datanode1 | f
 datanode2 | f
(2 rows)

-- Set datanode1 as the preferred DN.
gaussdb=# ALTER NODE datanode1 WITH(preferred = true);

-- Query the cluster DN status after the change.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 datanode1 | t
 datanode2 | f
(2 rows)

-- Connect to only one CN to delete cluster nodes.
gaussdb=# DROP NODE datanode1;
```

Helpful Links

CREATE NODE and **DROP NODE**

4.4.2.6.8 ALTER NODE GROUP

Description

Changes the information about a cluster node group.

Precautions

- Only the system administrator or a user who has been granted the ALTER permission for the node group can modify the node group information.
- Node group modification is an internal operation of the system. Except SET DEFAULT, other operations must be performed in maintenance mode (by calling **SET xc_maintenance_mode=on;**).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- ALTER NODE GROUP can be used only within a database. To avoid data inconsistency in DBMS, do not manually run this SQL statement.

Syntax

```
ALTER NODE GROUP groupname
{ SET DEFAULT
   | RENAME TO new_group_name
   | SET TABLE GROUP new_group_name
   | COPY BUCKETS FROM src_group_name
   | ADD NODE ( nodename [, ... ] )
   | DELETE NODE ( nodename [, ... ] )
   | SET SEQUENCE ( TO LOCAL | TO ALL NODE )
};
```

Parameters

- **groupname**

  Specifies the name of the node group to be modified.

  Value range: a string complying with **Identifier Description**.

- **SET DEFAULT**

  Sets **in_redistribution** to **'y'** for all node groups excluding the one specified by **groupname**. To be compatible with earlier versions, this syntax is retained and does not need to be executed in maintenance mode.

- **RENAME TO new_group_name**

  Changes the name of the node group specified by **groupname** to **new_group_name**.

- **SET TABLE GROUP new_group_name**

  Changes all the **group_names** in the **pgroup** columns of the **pgxc_class** tables on all CNs to **new_group_name**.

- **COPY BUCKETS FROM src_group_name**

  Copies the contents of the **group_members** and **group_buckets** columns from the node group specified by **src_group_name** to the node group specified by **groupname**.

  The current version does not support the COPY BUCKETS FROM option.

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

  Adds nodes from the node group specified by **groupname**. After the statement execution, the new nodes are registered with the **PGXC_NODE** system catalog. This statement only modifies the system catalog and does not add nodes or redistribute data. Do not call this statement. You can observe the impact of the statement in the **PGXC_GROUP** system catalog.

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

  Deletes nodes from the node group specified by **groupname**. The deleted nodes still exist in the **PGXC_NODE** system catalog. This statement only modifies the system catalog and does not delete nodes or redistribute data. Do not call this statement. You can observe the impact of the statement in the **PGXC_GROUP** system catalog.

- **SET SEQUENCE (TO LOCAL | TO ALL NODE)**

  Configures the sequence of the cluster to the local node or all nodes.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 dn_6001_6002_6003 | f
 dn_6004_6005_6006 | f
 dn_6007_6008_6009 | f
(3 rows)

-- Create a node group and replace dn_6001_6002_6003 with the actual node name obtained in the previous step.
gaussdb=# CREATE NODE GROUP test_group WITH ( dn_6001_6002_6003 );

-- Changes the name of the created node group to test_group_new. This must be implemented in the maintenance mode (by calling SET xc_maintenance_mode=on;).
gaussdb=# set xc_maintenance_mode=on;
gaussdb=# ALTER NODE GROUP test_group RENAME TO test_group_new;

-- Query the node group.
gaussdb=# SELECT group_name, group_members FROM pgxc_group;
   group_name   |   group_members
----------------+-------------------
 group_version1 | 16384 16388 16394
 test_group_new | 16384
(2 rows)

-- Delete the node group and disable the maintenance mode.
gaussdb=# DROP NODE GROUP test_group_new;
gaussdb=# SET xc_maintenance_mode=off;
```

Helpful Links

CREATE NODE GROUP and **DROP NODE GROUP**

4.4.2.6.9 ALTER RESOURCE LABEL

Description

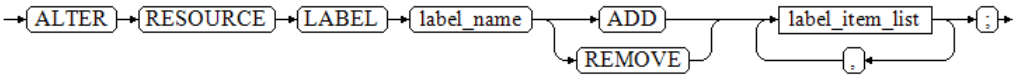
ALTER RESOURCE LABEL modifies a resource label. Currently, the distributed mode is not supported.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

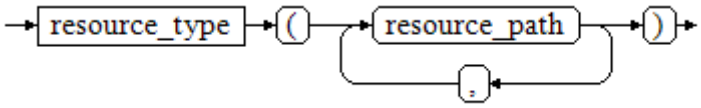
Syntax

```
ALTER RESOURCE LABEL label_name {ADD|REMOVE}
  label_item_list[, ...];
```



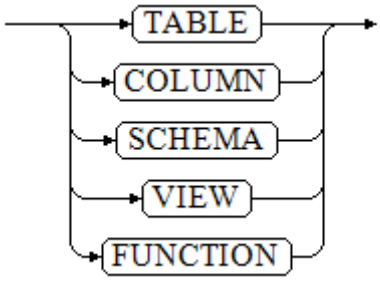
- label_item_list

  ```
  resource_type(resource_path[, ...])
  ```



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- resource_type
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION



Parameters

- **label_name**
  Specifies the resource label name.
  Value range: a string. It must comply with the **naming conventions**.
- **resource_type**
  Specifies the type of database resources to be labeled.
- **resource_path**
  Specifies the path of database resources.

Examples

```
-- Create basic table table_for_label.
m_db=# CREATE TABLE table_for_label(col1 int, col2 text);

-- Create resource label table_label.
m_db=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

-- Attach resource label table_label to col2.
m_db=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2);

-- Remove an item from table_label.
m_db=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

-- Delete the resource label table_label.
m_db=# DROP RESOURCE LABEL table_label;

-- Delete the base table table_for_label.
m_db=# DROP TABLE table_for_label;
```

Helpful Links

CREATE RESOURCE LABEL and **DROP RESOURCE LABEL**

4.4.2.6.10 ALTER ROLE

Description

Modifies role attributes.

Precautions

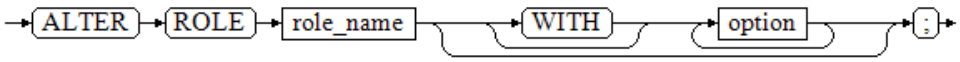
None

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

- Modify the permissions of a role.

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```

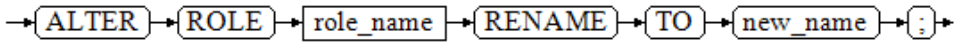


  - The **option** clause for granting permissions is as follows.

```
{CREATEDB | NOCREATEDB}
  | {CREATEROLE | NOCREATEROLE}
  | {AUDITADMIN | NOAUDITADMIN}
  | {SYSADMIN | NOSYSADMIN}
  | {MONADMIN | NOMONADMIN}
  | {OPRADMIN | NOOPRADMIN}
  | {POLADMIN | NOPOLADMIN}
  | {USEFT | NOUSEFT}
  | {INHERIT | NOINHERIT}
  | {LOGIN | NOLOGIN}
  | {PERSISTENCE | NOPERSISTENCE}
  | CONNECTION LIMIT connlimit
  | PASSWORD { 'password' [EXPIRED] | DISABLE }
  | IDENTIFIED BY { 'password' [ REPLACE 'old_password' | EXPIRED ] | DISABLE }
  | VALID BEGIN 'timestamp'
  | VALID UNTIL 'timestamp'
  | USER GROUP 'groupuser'
  | NODE GROUP logic_cluster_name
  | ACCOUNT { LOCK | UNLOCK }
```

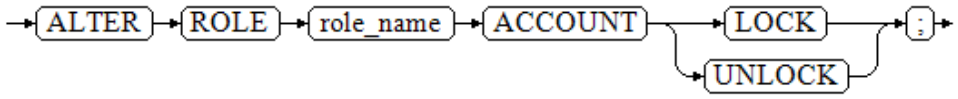
- Rename a role.

```
ALTER ROLE role_name
  RENAME TO new_name;
```



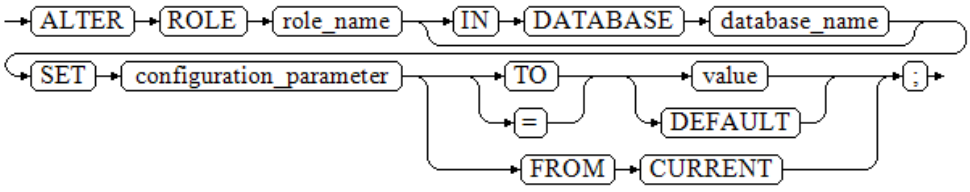
- Lock or unlock.

```
ALTER ROLE role_name
  ACCOUNT { LOCK | UNLOCK };
```



- Set parameters for a role.

```
ALTER ROLE role_name [ IN DATABASE database_name ]
  SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```



- Reset all configuration parameters of a role.

```
ALTER ROLE role_name
  [ IN DATABASE database_name ] RESET ALL;
```

Parameters

- **role_name**

  Specifies a role name.

  Value range: an existing role name. For details, see **•role_name**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **IN DATABASE database_name**

  Modifies the parameters of a role in a specified database.

- **SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT}**

  Sets parameters for a role. The session parameters modified using ALTER ROLE is only for a specific role and is valid in the next session triggered by the role.

  Value range:

  For details about the values of **configuration_parameter** and **value**, see **SET**.

  DEFAULT: clears the value of **configuration_parameter**. **configuration_parameter** will inherit the default value of the new session generated for the role.

  FROM CURRENT: uses the value of **configuration_parameter** of the current session.

- **RESET ALL**

  Restores all values of **configuration_parameter** to default values.

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - **ACCOUNT LOCK**: locks an account to forbid login to databases.
  - **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.

- **PGUSER**

  In the current version, the **PGUSER** attribute of a role cannot be modified.

- **{PASSWORD|IDENTIFIED BY} 'password'**

  Resets or changes the user password. Except the initial user, other administrators and common users need to enter the correct old password when changing their own passwords. Only the initial user, the system administrators (sysadmin) when separation of duties is disabled, or users who have the permission (CREATEROLE) to create users can reset the password of a common user without entering the old password. The initial user can reset passwords of system administrators. A system administrator cannot reset passwords of other system administrators. The user password should be enclosed in single quotation marks (').

- **EXPIRED**

  Invalidates the password. Only the initial user, the system administrators (sysadmin), or users who have the permission to create a user (CREATEROLE) can invalidate the user password. A system administrator can invalidate the password of itself or other system administrators only when the separation of duties is disabled. The password of the initial user cannot be invalidated.

  The user whose password is invalid can log in to the database but cannot perform the query operation. The query operation can be performed only after the password is changed or the administrator resets the password.

For details about other parameters, see **Parameters** in section "CREATE ROLE."

Examples

See **Examples** in section "CREATE ROLE."

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Helpful Links

CREATE ROLE, **DROP ROLE**, and **SET ROLE**

4.4.2.6.11 ALTER SCHEMA

Description

In the M-compatible database, DATABASE and SCHEMA are synonyms that allow you to modify the character set, collation, and owner of a database or schema.

Precautions

- Only the owner of a schema or users granted the ALTER permission on the schema can run the **ALTER SCHEMA** command. When separation of duties is disabled, system administrators have this permission by default. To change the owner of a schema, you must be the owner of the schema or system administrator and a member of the new owner role.
- For system schemas other than public, such as pg_catalog and sys, only the initial user is allowed to change the owner of a schema. Changing the names of the built-in system schemas may make some functions unavailable or even affect the normal running of the database. By default, the names of the built-in system schemas cannot be changed. To ensure forward compatibility, you can change the names of the built-in system schemas only when the system is being started or upgraded or when **allow_system_table_mods** is set to **on**.
- M compatibility does not allow to rename other schemas as **templatem** or rename the **templatem** schema as others.
- Except the initial user, other users cannot change the owner of a schema to an O&M administrator.

Syntax

- Modify the database owner.

```
ALTER SCHEMA schema_name
    OWNER TO new_owner;
```

- Modify the character set and collation of the database.

```
ALTER {DATABASE | SCHEMA} schema_name
    [create_option] ...
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Parameters

- **schema_name**

  Specifies the name of the database whose attributes are to be modified.

  Value range: a string complying with **Identifier Description**.

- **new_owner**

  Specifies the new owner of a database.

  Value range: a string indicating a valid username. For details about the username requirements, see **•user_name**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **COLLATE [=] default_collation**

  (Optional) Specifies the character set used by the database. For example, set this parameter by using **collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the sorting order of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

- **{CHAR SET | CHARSET | CHARACTER SET} [=] default_charset**

  (Optional) Specifies the character class used by the database. For example, set this parameter by using **CHARSET = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

Examples

- Modify the character set and collation of the database.

```
-- Create the test_db1 database.
m_db=# CREATE SCHEMA test_db1;
-- Change the character set of test_db1 to utf8.
m_db=# ALTER SCHEMA test_db1 CHARSET utf8;
-- Change the collation of test_db1 to utf8mb4_bin.
m_db=# ALTER SCHEMA test_db1 COLLATE utf8mb4_bin;
```

- Modify the database owner.

```
-- Create user scott.
m_db=# CREATE USER scott PASSWORD '********';
-- Change the owner of test_db1 to scott.
m_db=# ALTER SCHEMA test_db1 OWNER TO scott;
```

Helpful Links

CREATE SCHEMA and **DROP SCHEMA**

4.4.2.6.12 ALTER SEQUENCE

Description

Modifies the parameters of an existing sequence.

Precautions

- Only the owner of a sequence, a user granted the ALTER permission on a sequence, or a user granted the ALTER ANY SEQUENCE permission on a sequence can run the **ALTER SEQUENCE** command. When separation of duties is disabled, a system administrator has this permission by default. To modify a sequence owner, you must be the sequence owner or system administrator and a member of the new owner role.
- In the current version, you can modify only the owner, owning column, and maximum value. To modify other parameters, delete the sequence and create it again. Then, use the **Setval** function to restore parameter values.
- ALTER SEQUENCE MAXVALUE cannot be used in transactions, functions, and stored procedures.

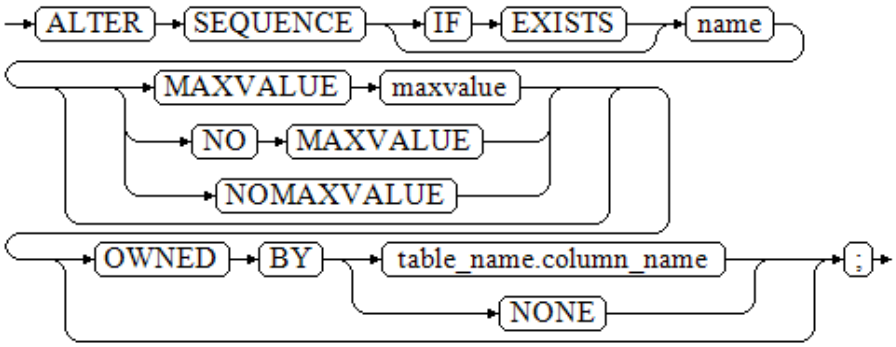
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- After the maximum value of a sequence is changed, the cache of the sequence in all sessions is cleared.
- The ALTER SEQUENCE statement blocks the calling of nextval, setval, currval, and lastval.

Syntax

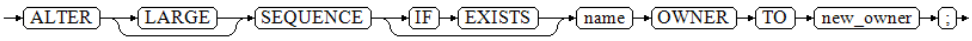
- Change the maximum sequence value and owning column.

```
ALTER SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```



- Change the owner of a sequence.

```
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```



Parameters

- **name**

  Specifies the name of the sequence to be modified.

- **IF EXISTS**

  This option is used when the sequence does not exist. ERROR is not displayed. Instead, a NOTICE message is returned.

- **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**

  Specifies the maximum value of the sequence. The new maximum value must be greater than **last_value**. If no new value is specified, the old maximum value is retained.

  Value range: (last_value, 2^{63} – 1]. If LARGE is used, the range is (last_value, 2^{127} – 1].

- **OWNED BY**

  Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to.

  If the sequence has been associated with another table before you use this option, the new association will overwrite the old one.

  The associated table and sequence must be owned by the same user and in the same schema.

  If OWNED BY NONE is used, all existing associations will be deleted.

- **new_owner**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the username of the new owner of the sequence. To change the owner, you must also be a direct or indirect member of the new role, and this role must have the CREATE permission on the sequence's schema. For details about the username requirements, see **•user_name**.

Examples

```
-- Create an ascending sequence named serial, which starts from 101.
m_db=# CREATE SEQUENCE serial START 101;

-- Query the next value of serial.
m_db=# SELECT pg_catalog.nextval('serial');
 nextval
---------
     101
(1 row)

-- Change the maximum value of serial to 101.
m_db=# ALTER SEQUENCE serial MAXVALUE 101;
ERROR:  Can not alter sequence uuid 1000011 maxval from 9223372036854775807 to 101, because new maxval 101 should be greater than currval 101 in gtm.

-- Delete the sequence.
m_db=# DROP SEQUENCE serial CASCADE;
```

Helpful Links

CREATE SEQUENCE and **DROP SEQUENCE**

4.4.2.6.13 ALTER SESSION

Description

ALTER SESSION defines or modifies the conditions or parameters that affect the current session. Modified session parameters are kept until the current session is disconnected.

Precautions

- If the START TRANSACTION statement is not executed before the SET TRANSACTION statement, the transaction is ended instantly and the statement does not take effect.
- You can use the transaction_mode(s) method declared in the START TRANSACTION statement to avoid using the SET TRANSACTION statement. For details, see **START TRANSACTION**.

Syntax

- Set transaction parameters of a session.

```
ALTER SESSION SET TRANSACTION
    { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | REPEATABLE READ |
SERIALIZABLE} } | { READ ONLY  | READ WRITE } } [, ...] ;
```

- Set other GUC parameters of a session.

```
ALTER SESSION SET
    {{config_parameter { { TO  | =  }  { value | DEFAULT } | FROM CURRENT }}
      | TIME ZONE time_zone
      | CURRENT_SCHEMA schema
      | SCHEMA 'schema'
      | NAMES encoding_name
      | NAMES encoding_name COLLATE collate_name
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  | ROLE role_name PASSWORD 'password'
  | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
} ;
```

Parameters

- **config_parameter**

  Specifies the name of a configurable GUC parameter. You can use SHOW ALL to view available GUC parameters.

  - **value**

    Specifies the new value of **config_parameter**. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. **DEFAULT** is used to set default values for parameters.

    - **DEFAULT**
    - **OFF**
    - **RESET**
    - User-specified value: The value must meet the restriction of the modified parameter.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **TIME ZONE timezone**

  Specifies the local time zone for the current session.

  Value range: a valid local time zone. The corresponding GUC parameter is **TimeZone**. The default value is **PRC**.

- **CURRENT_SCHEMA schema**

  Specifies the current schema.

  Value range: an existing schema name. If the schema name does not exist, the value of **CURRENT_SCHEMA** will be empty.

- **SCHEMA 'schema'**

  Specifies the current schema. The schema here is a string.

- **NAMES encoding_name**

  Specifies the client character encoding. This statement is equivalent to **set client_encoding to encoding_name**.

  Value range: a valid character encoding name. The GUC parameter corresponding to this option is **client_encoding**. The default encoding is **UTF8**.

- **ROLE role_name**

  Specifies the value range of **role_name**. For details about the role name requirements, see **•role_name**.

- **PASSWORD 'password'**

  Specifies the password for the role. It must comply with the password convention. Encrypted passwords are not supported.

- **SESSION AUTHORIZATION**

  Sets the session user identifier of the current session.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **ISOLATION LEVEL**

  Specifies a transaction isolation level, which determines what data is visible to a transaction if there are other concurrent transactions.

  Value range:

  - **READ COMMITTED**: Only committed data can be read. This is the default value.
  - **READ UNCOMMITTED**: After this isolation level is set, its behavior is the same as that of **READ COMMITTED**.

Examples

```
-- Create the ds schema.
m_db=# CREATE SCHEMA ds;

-- Set the search path of a schema.
m_db=# SET SEARCH_PATH TO ds, public;

-- Set the time/date type to the traditional postgres format (date before month).
m_db=# SET DATESTYLE TO postgres, dmy;

-- Set the character code of the current session to UTF8.
m_db=# ALTER SESSION SET NAMES 'UTF8';

-- Set the time zone to Berkeley of California.
m_db=# SET TIME ZONE 'PST8PDT';

-- Set the time zone to Italy.
m_db=# SET TIME ZONE 'Europe/Rome';

-- Set the current schema.
m_db=# ALTER SESSION SET CURRENT_SCHEMA TO ds;
-- Delete the ds schema.
m_db=# DROP SCHEMA ds;
-- Start a transaction and set the transaction level.
m_db=# START TRANSACTION;
m_db=# ALTER SESSION SET TRANSACTION READ ONLY;
m_db=# ROLLBACK;
```

Helpful Links

SET

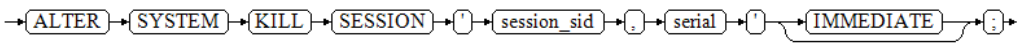
4.4.2.6.14 ALTER SYSTEM KILL SESSION

Description

ALTER SYSTEM KILL SESSION ends a session.

Syntax

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```



Parameters

- **session_sid, serial**

  Specifies **SID** and **SERIAL** of a session (see examples for format). You can use the pg_stat_activity system catalog to query the current active threads (see

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the examples). However, when you run the **ALTER SYSTEM KILL SESSION** command, the threads may have ended.

Value range: SIDs and SERIALs of all sessions that can be queried from the system catalog dv_sessions.

- **IMMEDIATE**

  Specifies that a session will be ended instantly after the statement is executed.

Examples

```
-- Start two sessions, and create a table and start a transaction to insert data in the first session.
gaussdb=# CREATE TABLE tbl_test(id int);
gaussdb=# BEGIN;
gaussdb=# INSERT INTO tbl_test VALUES (1);

-- Query the session information in the second session. If the value of state is idle in transaction, the
transaction is waiting to be committed.
gaussdb=# SELECT t1.datname,
       t1.usename,
       t1.pid,
       t2.serial#,
       t1.state
FROM pg_stat_activity t1,
     dv_sessions t2
WHERE t1.query LIKE 'INSERT INTO tbl_test%'
  AND t1.sessionid = t2.sid;
 datname  | usename |       pid       | serial# |        state
----------+---------+-----------------+---------+---------------------
 postgres | omm     | 139802072635136 |       0 | idle in transaction
(1 row)

-- End a session. If the IMMEDIATE parameter is not specified, the session and transactions in the session
are forcibly ended.
gaussdb=# ALTER SYSTEM KILL SESSION '139802072635136,0';
 pg_terminate_backend
----------------------
 t
(1 row)

-- Perform the reconnection and query the tbl_test table. The transaction is forcibly ended and the data is
rolled back.
gaussdb=# SELECT * FROM tbl_test;
 id
----
(0 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
```

4.4.2.6.15 ALTER TABLE

Description

Modifies tables, including modifying table definitions, renaming tables, renaming specified columns in tables, setting table schemas, and adding or updating multiple columns.

Precautions

- The owner of a table, users granted the ALTER permission on the table, or users granted the ALTER ANY TABLE permission can run the **ALTER TABLE** command. System administrators have the permission to run the command

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

by default. To modify the owner or schema of a table, you must be the table owner or system administrator and a member of the new owner role.

- The tablespace of a partitioned table cannot be modified, but the tablespace of a partition can be modified.
- 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.
- A column whose **DEFAULT** value contains the nextval() expression cannot be added.
- When you delete a PRIMARY KEY constraint by constraint name, the NOT NULL constraint is not deleted. If necessary, manually delete the NOT NULL constraint.
- When JDBC is used, the **DEFAULT** value can be set through PrepareStatement.
- If a column is added using ADD COLUMN, all existing rows in the table are initialized with the column's default value (**NULL** if no DEFAULT clause is specified).
  - If no **DEFAULT** value is specified for the new column, **NULL** is used, and no full table update is triggered.
  - If the new column has the **DEFAULT** value, the column must meet all the following requirements. Otherwise, the entire table will be updated, affecting online services.
    - The data type must be TINYINT, SMALLINT, BIGINT, INTEGER, NUMERIC, DECIMAL, BOOL, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, TIMESTAMP, DATE, or TIME.
    - The length of the **DEFAULT** value of the new column cannot exceed 128 bytes.
    - The **DEFAULT** value of the new column does not contain the volatile function.
    - The **DEFAULT** value is required and cannot be **NULL**.
    - If you are not sure whether the **DEFAULT** value of the new column does not contain volatile functions, check whether the **provolatile** attribute of the function in the **pg_rpoc** system catalog is **'v'**.

Syntax

- Modify the definition of a table.

```
ALTER TABLE { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
```

There are several clauses of **action**:

```
column_clause
    | ADD [CONSTRAINT] table_constraint
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | UPDATE SLICE LIKE table_name
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    | COMMENT [ = ] 'string'
    | [ [ DEFAULT ] {CHARACTER SET | CHAR SET | CHARSET} [ = ] charset_name] [ [ DEFAULT ]
COLLATE [ = ] collation_name]
    | ADD index_clause
    | DROP PRIMARY KEY
    | REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- **ADD [CONSTRAINT] table_constraint**

  Adds a table constraint.

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  Deletes a table constraint.

- **SET ({storage_parameter = value} [, ...])**

  Changes one or more storage parameters for the table. If the value of **table_name** is an index name, **ACTIVE_PAGES** specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimizer optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM (including AUTOVACUUM) and ANALYZE. You are advised not to manually set this parameter because it is invalid in distributed mode.

- **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 a table, sequence, or view to a specified user. For details about the username requirements, see **•user_name**.

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

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

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

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

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

- **UPDATE SLICE LIKE table_name**

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

- **[[DEFAULT] {CHARACTER SET | CHAR SET | CHARSET} [=] default_charset] [[DEFAULT] COLLATE [=] default_collation]**

  Changes the default character set and collation of a table to the specified values. The modification does not affect the existing columns in the table.

- **DROP PRIMARY KEY**

  Deletes a primary key constraint from a table.

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  Specifies the record level of old tuples in UPDATE and DELETE statements on a table in logical replication scenarios.

  - **DEFAULT** records the old value of the primary key column. If there is no primary key, **DEFAULT** does not record the old value.
  - **USING INDEX** records the old values of columns covered by the named indexes. These values must be unique, non-local, and non-deferrable, and contain the values of columns marked **NOT NULL**.
  - **FULL** records the old values of all columns in the row.
  - **NOTHING** does not record information in old rows.

  In logical replication scenarios, when the UPDATE and DELETE statements of a table are parsed, the parsed old tuples consist of the information recorded in this

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

method. For tables with a primary key, this option can be set to **DEFAULT** or **FULL**. For a table without a primary key, set this parameter to **FULL**. Otherwise, the old tuple will be parsed as empty during decoding. Generally, you are advised not to set this parameter to **NOTHING** because old tuples are always parsed as empty.

For Ustore tables, the **NOTHING** option is invalid, and the actual effect is the same as that of **FULL**. If **DEFAULT** does not have a primary key, all columns in the row are recorded.

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

```
ADD [ COLUMN ] column_name data_type [ {CHARACTER SET | CHAR SET | CHARSET}
charset ][ COLLATE collation ] [ column_constraint [ ... ] ]
| ADD ({ column_name data_type [ compress_mode ]} [, ...] )
| DROP [ COLUMN ]  column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name { SET DEFAULT default_expr | DROP DEFAULT }
```

NOTE

- **ADD [COLUMN] column_name data_type [{CHARACTER SET | CHAR SET | CHARSET} charset] [COLLATE collation] [column_constraint [...]]**

  Adds a column to a table. If a column is added using 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 multiple columns to a table.

- **DROP [COLUMN] column_name [RESTRICT | CASCADE]**
  - Drops a column from a table. Indexes and constraints related to the column are automatically dropped. The **CASCADE** option is supported only as syntax but does not take effect when the version compatibility control parameter **m_format_dev_version** of an M-compatible database is set to **'s1'** or later.
  - The DROP COLUMN statement 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 tablespace on the disks, because the space occupied by the deleted column is not recycled. The space will be recycled when VACUUM is executed.

- **ALTER [COLUMN] column_name { SET DEFAULT default_expr | DROP DEFAULT }**

  Sets or removes the default value for a column. The default values only apply to subsequent INSERT operations; 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.

  For details about the value range of **default_expr**, see **•DEFAULT default_expr**.

- **column_constraint** is as follows:

```
COMMENT 'string'
|[ CONSTRAINT constraint_name ]
   { NOT NULL |
     NULL |
     CHECK ( expression ) |
     DEFAULT default_expr  |
     ON UPDATE update_expr |
     UNIQUE [KEY] index_parameters |
     [PRIMARY] KEY index_parameters
 }
```

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

```
[ DICTIONARY ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **update_expr** is as follows:
    ```
    { CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
    ```

- index_clause
  ```
  {INDEX | KEY} [ [schema_name.] index_name ] [ USING method ] ({column_name [(length)] | (expr) [ASC | DESC]}[,...]) [[COMMENT 'string' | USING method][...]]
  ```
  For details about the parameters, see **CREATE INDEX**.
- **table_constraint** is as follows:
  ```
  [ CONSTRAINT [ constraint_name ] ]
     { CHECK ( expression ) |
       UNIQUE [INDEX|KEY][index_name] [USING access_method] ( { { column_name [ ( length ) ] | ( expression ) } [ ASC | DESC ] } [, ... ] ) index_parameters [USING access_method] [comment 'string'] [ BY GLOBAL INDEX ]|
       PRIMARY KEY [ USING access_method ] [index_name] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters [USING access_method] [comment 'string']
  }
        }
  ```
  The value of **access_method** is as follows:
  ```
  BTREE
  ```
  index_parameters is as follows:
  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  ```
  Specifies whether to create a global secondary index.
  ```
  [BY GLONAL INDEX]
  ```

- Rename a table. The renaming does not affect stored data.
  ```
  ALTER TABLE table_name
    RENAME [TO | AS | = ] new_table_name;
  ```
- Rename a table index.
  ```
  ALTER TABLE table_name
    RENAME {INDEX | KEY} old_index_name TO new_index_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 the CREATE permission 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 change 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 the user can only rebuild and delete the table. However, a system administrator can modify all permissions on any table in any way.
  - 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 tables need be made.
  - Adding a CHECK or NOT NULL constraint will scan the table to validate that existing rows meet the constraint.
  - Adding a column with a non-null default or changing the type of an existing column will rewrite the entire table. Rewriting a large table may take much time and temporarily needs doubled disk space.

- Add columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

Parameters

- **IF EXISTS**

  Sends a notice instead of an error if no tables have identical names. The notice prompts that the table you are querying does not exist.

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name is the name of the table to be modified.

  If **ONLY** is specified, only the table is modified. If **ONLY** is not specified, the table and all subtables are modified. You can add the asterisk (*) option following the table name to specify that all subtables are scanned, which is the default operation. Currently, the syntax of **ONLY** with the ***** option is reserved, but the function is not supported.

- **constraint_name**
  - Specifies the name of an existing constraint to be dropped in the DROP CONSTRAINT operation.
  - Specifies the name of a new constraint in the ADD CONSTRAINT operation.

- **index_name**

  Specifies the index name.

  NOTICE

  In the ADD CONSTRAINT operation, if both **constraint_name** and **index_name** are set for the PRIMARY/UNIQUE KEY constraint, the index name is specified by **index_name**.

- **USING method**

  Specifies the method of creating an index.

  For details about the value range, see "USING method" in **Parameters**.

  NOTICE

  In the ADD CONSTRAINT operation:
  - If USING method is not specified, the default index method is btree for ASTORE or UB-tree for USTORE.
  - If the storage mode of a table is USTORE and the constraint in the SQL statement is specified as USING BTREE, the underlying layer automatically creates the constraint as USING UBTREE.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

- **expression**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies an expression index constraint based on one or more columns of the table. It must be written in parentheses.

- **storage_parameter**

  Specifies the name of a storage parameter.

  TDE options:

  – enable_tde (Boolean)

    Specifies whether to set a table as an encrypted table. This parameter supports only row-store tables, segment-page tables, temporary tables, and unlogged tables. When setting **enable_tde** to **on**, ensure that the TDE function has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*.

    Value range: **on** and **off**.

    - **on**: Transparent data encryption is enabled.
    - **off**: Transparent data encryption is disabled.

      NOTE

      - After the value is changed from **on** to **off**, the inserted or updated data is still encrypted when being written to the old page, and is not encrypted when being written to the new page generated after the switchover.
      - After the value is changed from **off** to **on**, the inserted or updated data is not encrypted when being written to the old page, and is automatically encrypted when being written to the new page generated after the switchover.

      That is, the encryption status on the new data page is the same as that after the encryption switchover, and the encryption status on the old data page is the same as that before the switchover. You are advised to manually perform VACUUM FULL on the table after the encryption switchover to ensure that the encryption status of all data pages is consistent.

    Default value: **off**

  – **encrypt_algo** (string type)

    Specifies the encryption algorithm of the encryption table.

    Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

    Default value: If **enable_tde** is set to **on** and the subparameter **table_algorithm** in the GUC parameter **tde_encrypt_config** is not empty, the default value is the same as that of **table_algorithm**. If **table_algorithm** is empty, the default value is **AES_128_CTR**. If **enable_tde** is set to **off**, the default value is null.

  The following option is added for creating an index:

  – **parallel_workers** (int type)

    Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

    Value range: [0,32]. The value **0** indicates that concurrent index creation is disabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Default value: If this parameter is not set, the concurrent index creation function is disabled.

- **hasuids** (Boolean type)

  Default value: **off**

  If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

- **new_owner**

  Specifies the name of the new table owner. For details about the username requirements, see **•user_name**.

- **column_name**, **column_1_name**, **column_2_name**

  Specifies the name of a new or existing column.

- **data_type**

  Specifies the type of a new column or a new type of an existing column.

- **compress_mode**

  Specifies whether to compress a table column. The clause specifies the compression algorithm preferentially used by the column. The current option cannot be used because row-store tables do not support compression.

- **charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

- **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. You can run the **select * from pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

  The utf8mb4_bin, utf8mb4_general_ci, utf8mb4_unicode_ci, binary, gbk_chinese_ci, gbk_bin, gb18030_chinese_ci, and gb18030_bin collations are also supported. For details, see **Table-level Character Sets and Collations**.

  NOTE

  - Only the character type supports the specified character set. If the BINARY character set or collation is specified, the character type is converted to the corresponding binary type. If the type mapping does not exist, an error is reported. Currently, only the mapping from the TEXT type to the BLOB type is available.
  - Except the BINARY character set and collation, only the character set that is the same as the database encoding can be specified.
  - If the character set or collation of a column is not explicitly specified and the default character set or collation of the table is specified, the character set or collation of the column is inherited from the table.
  - If the modified character set or the character set corresponding to the collation is different from the character set of the current column, the data in the column is converted to the specified character set for encoding.

- **NOT NULL | NULL**

  Sets whether the column allows null values.

- **ENABLE**

  Specifies that the constraint is enabled. By default, the constraint is enabled.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **CHECK (expression)**

  New rows or rows to be updated must satisfy for an expression to be true. If any row produces a false result, an error is raised and the database is not modified.

  Currently, the CHECK expression cannot contain subqueries.

- **DEFAULT default_expr**
  - The DEFAULT clause is used to specify a default expression for a 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**.
  - The following content can be specified: constants, numeric constants with positive and negative signs, and **update_expr**.
  - The value of **update_expr** can be used as the default value only for columns of the TIMESTAMP or DATETIME type, and the precision of the columns must be the same as that of **update_expr**.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - In terms of syntax, **update_expr** supports four keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, LOCALTIME, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The four types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified on columns of the TIMESTAMP and DATETIME types.
  - The precision specified by this attribute must be the same as that specified by the type in the corresponding column. Otherwise, an error is reported. For example, **CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(6));**. If the precision is inconsistent, **ERROR: Invalid ON UPDATE clause for "col1"** is reported.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.
  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details, see **•COMMENT [=] 'string'**.

- **Column-level unique constraint: UNIQUE [KEY] index_parameters**

  UNIQUE KEY has the same semantics as UNIQUE.

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

  NOTE

  If the source table is hash distributed table, creating a column-level unique key constraint will fail. You can use CREATE TABLE ADD to create a table-level unique constraint. The table-level unique constraint supports the **BY GLOBAL INDEX** option.

- **Table-level unique constraint: UNIQUE [INDEX | KEY] [index_name] [USING method]({{ column_name [(length)] | (expression) } [ASC | DESC] }[, ...]) index_parameters [BY GLOBAL INDEX]**

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

  column_name (length) indicates the prefix key. For details, see **•column_name (length)**.

  index_name indicates the index name.

  NOTICE

  For a unique key constraint, if both **constraint_name** and **index_name** are specified, the value of **index_name** is used as the index name.

- **Column-level primary key constraint: [PRIMARY] KEY index_parameters**

  Table-level primary key constraint: PRIMARY KEY [index_name] [USING method] ({ column_name [ASC | DESC] } [, ...]) index_parameters

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

- **WITH ({storage_parameter = value} [, ...])**

  Specifies an optional storage parameter for an index.

- **new_table_name**

  Specifies the new table name.

- **new_column_name**

  Specifies the new name of a specific column in a table.

- **new_index_name**

  Specifies the new index name in a table.

- **new_schema**

  Specifies the new schema name.

- **CASCADE | RESTRICT**

  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.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> This attribute is supported only as syntax but does not take effect when the version compatibility control parameter **m_format_dev_version** of an M-compatible database is set to **'s1'** or later.

- **schema_name**

  Specifies the schema name of a table.

Examples of Modifying a Table

- Rename a table.

```
m_db=# CREATE TABLE aa(c1 int, c2 int);
m_db=# ALTER TABLE aa RENAME TO test_alt1;
```

- Modify the schema of a table.

```
-- Create the test_schema schema.
m_db=# CREATE SCHEMA test_schema;
-- Change the schema of the test_alt1 table to test_schema.
m_db=# ALTER TABLE test_alt1 SET SCHEMA test_schema;
-- Query table information.
m_db=# SELECT schemaname,tablename FROM pg_tables WHERE tablename = 'test_alt1';
 schemaname  | tablename
-------------+-----------
 test_schema | test_alt1
(1 row)
```

- Change the owner of a table.

```
-- Create user test_user.
m_db=# CREATE USER test_user PASSWORD 'XXXXXXXXXX';
-- Change the owner of the test_alt1 table to test_user.
m_db=# ALTER TABLE test_schema.test_alt1 OWNER TO test_user;
-- Query.
m_db=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename =
'test_alt1';
 tablename | schemaname  | tableowner
-----------+-------------+------------
 test_alt1 | test_schema | test_user
(1 row)
```

Examples of Modifying a Column

- Add columns.

```
-- Create a table.
m_db=# CREATE TABLE test_alt2(id INT,areaid INT);
-- Add a column to the test_alt2 table.
m_db=# ALTER TABLE test_alt2 ADD COLUMN name VARCHAR(20);
-- Query.
m_db=# \d test_alt2
          Table "public.test_alt1"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 areacode | integer               |
 name     | varchar(20)           | collate utf8mb4_general_ci
```

- Delete a column.

```
-- Delete the areaid column from test_alt2.
m_db=# ALTER TABLE test_alt2 DROP COLUMN areaid;
-- Query.
m_db=# \d test_alt2
         Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 name   | varchar(50)           |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples of Modifying a Constraint

- Modify the default value of a column.

```
-- Create a table.
m_db=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));
-- Modify the default value of id in the test_alt1 table.
m_db=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';
-- Query.
m_db=# \d test_alt3
                Table "public.test_alt3"
 Column |         Type          |        Modifiers
--------+-----------------------+-------------------------
 pid    | integer               |
 areaid | char(5)               | default '00000'::bpchar
 name   | varchar(20)           |
-- Delete the default value of id.
m_db=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;
-- Query.
m_db=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | char(5)               |
 name   | varchar(20)           |
```

- Add a table-level constraint.

  Directly add a constraint.

```
-- Add a primary key constraint to the table.
m_db=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);
-- Query.
m_db=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               | not null
 areaid | integer               |
 name   | varchar(20)           |
Indexes:
    "pk_test3_pid" PRIMARY KEY, btree (pid) TABLESPACE pg_default
```

Helpful Links

CREATE TABLE and **DROP TABLE**

4.4.2.6.16 ALTER TABLE PARTITION

Description

Modifies table partitions, including adding, deleting, splitting, merging, clearing, exchanging, and renaming partitions, moving partition tablespaces, enabling and disabling the automatic partitioning function, and modifying partition attributes.

Precautions

- Only the partitioned table owner or a user granted with the ALTER permission can run the **ALTER TABLE PARTITION** command. When separation of duties is disabled, system administrators have this permission by default.
- The tablespace of the added partition cannot be **pg_global**.
- The name of the added partition must be different from names of existing partitions in the partitioned table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition.
- Hash partitions cannot be added.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.
- If a partitioned table has only one partition, the partition cannot be deleted.
- Use PARTITION FOR() to choose partitions. The number of specified values in the brackets should be the same as the number of columns specified when you define a partition, and they must be consistent.
- Hash partitioned tables do not support splitting, combination, addition, or deletion of partitions.
- Deleting, splitting, merging, clearing, and exchanging partitions will invalidate global indexes. The UPDATE GLOBAL INDEX clause can be used to update the indexes synchronously.
- If the UPDATE GLOBAL INDEX clause is not used when you delete, split, merge, clear, or exchange partitions, concurrent DML services may report errors due to invalidated indexes.
- If **enable_gpi_auto_update** is set to **on**, the global index is automatically updated even if the UPDATE GLOBAL INDEX clause is not declared.

Syntax

Modifying a partition in a partitioned table includes modifying the main syntax of the table partition and modifying the syntax of the table partition name.

- Modify the syntax of the table partition.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
```

action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, M-compatible databases execute DROP PARTITION and ADD PARTITION first, and then the rest clauses in sequence.

```
    move_clause |
    exchange_clause |
    merge_clause |
    split_clause |
    add_clause |
    drop_clause |
    truncate_clause |
```

  - The move_clause syntax is used to move the partition to a new tablespace.

```
MOVE PARTITION { partition_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

  - The exchange_clause syntax is used to move the data from an ordinary table to a specified partition.

```
EXCHANGE PARTITION {  partition_name | FOR ( partition_value [, ... ] ) }
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
[ { WITH | WITHOUT } VALIDATION ]  [ UPDATE GLOBAL INDEX ] [ [NO] UPDATE DISTRIBUTED GLOBAL INDEX]
```

The ordinary table and partition whose data is to be exchanged must meet the following requirements:

- The number of columns in the ordinary table is the same as that of the partition, and their information should be consistent, including: column name, data type, constraint, collation information, storage parameter, and compression information.
- The compression information of the ordinary table and partition should be consistent.
- The number of ordinary table indexes is the same as that of local indexes of the partition, and the index information is the same.
- The number and information of constraints of the ordinary table and partition should be consistent.
- An ordinary table cannot be a temporary table. A partitioned table can only be a range partitioned table, list partitioned table, or hash partitioned table.

NOTICE

- When the exchange is done, the data and tablespace of the ordinary table and partition are exchanged. The statistics about ordinary tables and partitions become unreliable, and they should be analyzed again.
- A non-partition key cannot be used to create a local unique index. Therefore, if an ordinary table contains a unique index, data cannot be exchanged.

  To exchange data, you can create an intermediate table. You need to insert the partition data into the intermediate table, truncate the partition, insert the ordinary table data into the partitioned table, drop the ordinary table, and rename the intermediate table to complete data exchange.
- If the DROP COLUMN operation is performed on an ordinary or partitioned table, the deleted column still exists physically. Therefore, you need to ensure that the deleted column of the ordinary table is strictly aligned with that of the partition. Otherwise, the exchange will fail.
- EXCHANGE PARTITION *partition_name*:
  - If *partition_name* is the name of a level-1 partition, the level-1 partition and ordinary table are exchanged. If *partition_name* is the name of a level-2 partition, the level-2 partition and ordinary table are exchanged.
  - Level-1 partitions and ordinary tables cannot be exchanged in level-2 partitioned tables.
  - After partitions are exchanged, auto-increment columns are not reset.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The merge_clause syntax is used to merge partitions into one. The maximum number of source partitions that can be merged in a command is 300.

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
    [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

NOTICE

- For range partitioning, the ranges of the source partitions must increase continuously, and the partition name after MERGE can be the same as the name of the last source partition. For list partitioning, there is no such range requirement on the source partitions, and the partition name after MERGE can be the same as that of any source partition. If the partition name after MERGE is the same as that of a source partition, they are considered as the same partition.
- Ustore tables do not support ALTER TABLE MERGE PARTITIONS in transaction blocks and stored procedures.

- The split_clause syntax is used to split one partition into multiple ones.

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause |
no_split_point_clause } [ UPDATE GLOBAL INDEX ]
```

NOTICE

- The partition name after SPLIT can be the same as the source partition name, but they are regarded as different partitions.
- If the GUC parameter **enable_ilm** is not enabled and the split_clause syntax is used to split a partition with the ILM policy into multiple partitions, the new partitions do not inherit the ILM policy.

  - The split_point_clause syntax for specifying the split point for a range partitioned table is as follows:

```
AT ( partition_value ) INTO ( PARTITION partition_name [ TABLESPACE
tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

    NOTICE

    The size of the split point should be in the range of partition keys of the partition to be split. The split point can only split one partition into two new partitions.

  - The no_split_point_clause syntax without specifying a split point for a range partitioned table is as follows:

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- The first new partition key specified by **partition_less_than_item** should be greater than that of the previously split partition (if any), and the last partition key specified by **partition_less_than_item** should equal that of the partition being split.
- The first new partition key specified by **partition_start_end_item** should equal that of the former partition (if any), and the last partition key specified by **partition_start_end_item** should equal that of the partition being split. no_split_point_clause
- **partition_less_than_item** supports a maximum of 16 partition keys, while **partition_start_end_item** supports only one partition key. For details about the supported data types, see **PARTITION BY RANGE [COLU...**.
- **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement. There is no restriction on different SPLIT statements.

- The syntax of partition_less_than_item is as follows (the range of the last partition is not defined, that is, the VALUES LESS THAN (partition_value) part is not defined; by default, the last partition inherits the upper boundary value of the range defined for the source partition):

```
PARTITION partition_name VALUES LESS THAN {( { partition_value | MAXVALUE }  [, ...] ) | MAXVALUE }
   [ TABLESPACE tablespacename ]
```

NOTICE

During range partitioning, the MAXVALUE keyword can be used without parentheses. It cannot be used in subpartitions of level-2 partitions or in scenarios where the partition field contains multiple columns.

- The partition_start_end_item syntax is as follows: (For details about the constraints, see **START END**.)

```
PARTITION partition_name {
      {START(partition_value) END ({partition_value | MAXVALUE})} |
      {START(partition_value)} |
      {END({partition_value | MAXVALUE})}
}  [TABLESPACE tablespace_name]
```

- The split_point_clause syntax for specifying the split point for a list partitioned table is as follows:

```
VALUES ( partition_value_list ) INTO ( PARTITION partition_name[ TABLESPACE tablespacename ] , PARTITION partition_name [ TABLESPACE tablespacename ] )
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

The split point must be a non-empty true subset of the source partition. Specifying a split point can only split one partition into two partitions.

- The no_split_point_clause syntax for not specifying the split point for a list partitioned table is as follows:

```
INTO ( PARTITION partition_name VALUES (partition_value_list) [ TABLESPACE tablespacename ][, ...] )
```

NOTICE

- The range of the last partition is not defined; the partition range is equal to the remaining set of the source partition excluding other level-2 partitions.
- If no split point is specified, each new partition must be a non-empty true subset of the source partition and does not overlap with each other.

– The add_clause syntax is used to add one or more partitions to a specified partitioned table.

```
ADD PARTITION({partition_less_than_item | partition_start_end_item | partition_list_item}[, ...])
```

The partition_list_item syntax is as follows:

```
PARTITION partition_name VALUES [IN] (list_values_clause)
    [ TABLESPACE tablespacename ]
```

NOTICE

- **partition_list_item** supports a maximum of 16 partition keys. For details about the supported data types, see **PARTITIONBYLIST[COLUMNS](partition_key)**.
- Interval and hash partitioned tables do not support partition addition.
- IN cannot be used in subpartitions of level-2 partitions.

– The drop_clause syntax is used to remove a partition from a specified partitioned table.

```
DROP PARTITION partition_name
```

NOTICE

- Hash partitioned table does not support partition deletion.
- If a partitioned table has only one partition, the partition cannot be deleted.

– The truncate_clause syntax is used to remove a specified partition from a partitioned table.

```
TRUNCATE PARTITION  { { ALL | partition_name [, ...] } | FOR (  partition_value [, ...] )  }
[ UPDATE GLOBAL INDEX ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The syntax for modifying the name of a partition is as follows:

```
ALTER TABLE [ IF EXISTS ]  table_name
    RENAME PARTITION { partition_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

Parameters

- **table_name**

  Specifies the name of a partitioned table.

  Value range: an existing table name

- **partition_name**

  Specifies the name of a partition.

  Value range: an existing partition name

- **tablespacename**

  Specifies which tablespace the partition moves to.

  Value range: an existing tablespace name

  Note: You need to create or delete tablespaces in a non-M-compatible database.

- **partition_value**

  Specifies the key value of a partition.

  The value specified by PARTITION FOR (partition_value [, ...]) can uniquely identify a partition.

  Value range: partition keys for the partition to be operated.

- **ordinary_table_name**

  Specifies the name of the ordinary table whose data is to be migrated.

  Value range: an existing ordinary table name.

- **{ WITH | WITHOUT } VALIDATION**

  Checks whether the ordinary table data meets the specified partition key range of the partition to be migrated.

  Value range:

  - **WITH**: checks whether the ordinary table data meets the partition key range of the partition to be migrated. If any data does not meet the required range, an error is reported.
  - **WITHOUT**: does not check whether the ordinary table data meets the partition key range of the partition to be migrated.

  The default value is **WITH**.

  The check is time consuming, especially when the data volume is large. Therefore, use **WITHOUT** when you are sure that the current ordinary table data meets the partition key range of the partition to be migrated.

- **partition_new_name**

  Specifies the new name of a partition.

  Value range: a string complying with **Identifier Description**.

- **{ ALL | partition_name [, ...] }**

  ALL: clears all partition data.

  partition_name: specifies a partition in the target partitioned table. Level-1 and level-2 partition names are supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: an existing partition name

- **UPDATE GLOBAL INDEX**

  Updates all global indexes in the partitioned table to ensure that correct data can be queried using global indexes. This parameter requires that the table be a segment page table and a hash bucket table. If this parameter is not used, all global indexes in the partitioned table become invalid.

- **UPDATE DISTRIBUTED GLOBAL INDEX**

  Updates all global indexes in the partitioned table to ensure that correct data can be queried using global indexes. If this parameter is not used, all global indexes in the partitioned table become invalid. This parameter is not used when **NO** is added in front of this parameter.

Examples

- Change the name of a table partition:

```
-- Create a pre-partitioned table.
m_db=# CREATE TABLE test_p1 (col1 INT, col2 INT) PARTITION BY RANGE (col1)
(
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN (20),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

-- Change the partition name.
m_db=# ALTER TABLE test_p1 RENAME PARTITION p3 TO pmax;

-- Query partition information.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p1'::regclass AND
parttype <> 'r';
 relname | boundaries |  oid
---------+------------+-------
 p1      | {10}       | 17066
 p2      | {20}       | 17067
 pmax    | {NULL}     | 17068
(3 rows)
```

- Swap partitions:

```
-- Create an ordinary table and insert data into the table.
m_db=# CREATE TABLE test_ep1(col1 INT,col2 INT);
m_db=# INSERT INTO test_ep1 VALUES (GENERATE_SERIES(1,9), 1000);

-- Migrate ordinary table data to a specified partition.
m_db=# ALTER TABLE test_p1 EXCHANGE PARTITION p1 WITH TABLE test_ep1;

-- Query.
m_db=# SELECT COUNT(*) FROM test_p1 PARTITION (p1);
 count
-------
     9
(1 row)

-- Delete the test_ep1 table.
m_db=# DROP TABLE test_ep1;
```

- Merge partitions:

```
-- Merge the p2 and pmax partitions in the test_p1 table into pmax.
m_db=# ALTER TABLE test_p1 MERGE PARTITIONS p2,pmax INTO PARTITION pmax;

-- View a partition.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p1'::regclass AND
parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1      | {10}       | 17066
 pmax    | {NULL}     | 17070
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(2 rows)

-- Delete tables and tablespaces.
m_db=# DROP TABLE test_p1;
```

- Split partitions:

```
-- Create a table.
m_db=# CREATE TABLE test_r1 (col1 INT,col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1   VALUES LESS THAN (10),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Split partitions.
m_db=# ALTER TABLE test_r1 SPLIT PARTITION pmax AT (20) INTO (PARTITION p2, PARTITION
pmax);
m_db=# ALTER TABLE test_r1 SPLIT PARTITION pmax INTO (
    PARTITION p3   VALUES LESS THAN (30),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Query.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_r1'::regclass AND
parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1      | {10}       | 17088
 p2      | {20}       | 17090
 p3      | {30}       | 17092
 pmax    | {NULL}     | 17093
(4 rows)

-- Delete the test_r1 table.
m_db=# DROP TABLE test_r1;
-- Create a table.
m_db=# CREATE TABLE test_r2(col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1   START(1) END(10),
    PARTITION p2   START(10) END(20),
    PARTITION pmax START(20) END(MAXVALUE)
);

-- Split partitions.
m_db=# ALTER TABLE test_r2 SPLIT PARTITION pmax INTO (
    PARTITION p3 START(20) END(30),
    PARTITION pmax START(30) END (MAXVALUE)
);

-- Query.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_r2'::regclass AND
parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1_0    | {1}        | 17112
 p1_1    | {10}       | 17113
 p2      | {20}       | 17114
 p3      | {30}       | 17116
 pmax    | {NULL}     | 17117
(5 rows)

-- Delete the test_r2 table.
m_db=# DROP TABLE test_r2;
-- Create a table.
m_db=# CREATE TABLE test_l1(col1 INT, col2 INT) PARTITION BY LIST(col1)(
    PARTITION p1   VALUES (10,20),
    PARTITION p2 VALUES (30,40)
);

-- Split partitions.
m_db=# ALTER TABLE test_l1 SPLIT PARTITION p1 VALUES (10) INTO (PARTITION p1_1, PARTITION
p1_2);
m_db=# ALTER TABLE test_l1 SPLIT PARTITION p2 INTO (PARTITION p3_1 VALUES(30), PARTITION
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
p3_2);

-- Query.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_l1'::regclass AND
parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1_1    | {10}       | 17132
 p1_2    | {20}       | 17133
 p3_1    | {30}       | 17134
 p3_2    | {40}       | 17135
(4 rows)

-- Delete the test_l1 table.
m_db=# DROP TABLE test_l1;
```

- Add partitions:

```
-- Create a table.
m_db=# CREATE TABLE test_p2 (col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN (20)
);

-- Add a partition.
m_db=# ALTER TABLE test_p2 ADD PARTITION (PARTITION p3 VALUES LESS THAN (30));

-- Delete the test_p2 table.
m_db=# DROP TABLE test_p2;
-- Create a table.
m_db=# CREATE TABLE test_p3 (col1 INT, col2 INT) PARTITION BY LIST(col1)(
    PARTITION p1 VALUES (1),
    PARTITION p2 VALUES (2)
);

-- Add a partition.
m_db=# ALTER TABLE test_p3 ADD PARTITION (PARTITION p3 VALUES (3));

-- Delete the test_p3 table.
m_db=# DROP TABLE test_p3;

-- Enter the M mode.
-- Create a table.
m_db=# CREATE TABLE addpart1
(
   month_code int,
   num        int,
   name       varchar(30)
)PARTITION BY RANGE COLUMNS(month_code)
(
   PARTITION p_202301 VALUES LESS THAN(202302),
   PARTITION p_202302 VALUES LESS THAN (202303)
);
-- Add a partition.
m_db=# ALTER TABLE addpart1 ADD PARTITION (PARTITION p_202303 VALUES LESS THAN(202304));
-- Drop the table.
m_db=# DROP TABLE addpart1;
```

- Drop partitions:

```
-- Create a table.
m_db=# CREATE TABLE test_p4 (col1 INT, col2 INT) PARTITION BY LIST(col1)(PARTITION p1 VALUES
(1),PARTITION p2 VALUES (2));

-- Delete the p2 partition from the test_p3 table.
m_db=# ALTER TABLE test_p4 DROP PARTITION p2;

-- Query.
m_db=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p4'::regclass;
 relname | boundaries |  oid
---------+------------+-------
 test_p4 |            | 17187
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 p1     | {1}       | 17188
(2 rows)

-- Delete the test_p4 table.
m_db=# DROP TABLE test_p4;
```

- Truncate partitions:

```
-- Create a table.
m_db=# CREATE TABLE test_p5 (col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (5),
    PARTITION p2 VALUES LESS THAN (10)
);

-- Insert data.
m_db=# INSERT INTO test_p5 VALUES (GENERATE_SERIES(1,9), 100);

-- View data in the p2 partition.
m_db=# SELECT * FROM test_p5 PARTITION (p2);
 col1 | col2
------+------
    5 |  100
    6 |  100
    7 |  100
    8 |  100
    9 |  100
(5 rows)

-- Clear the data in the p2 partition.
m_db=# ALTER TABLE test_p5 TRUNCATE PARTITION p2;

-- Clear data in the p1 and p2 partitions.
m_db=# ALTER TABLE test_p5 TRUNCATE PARTITION p1,p2;

-- Clear all partition data.
m_db=# ALTER TABLE test_p5 TRUNCATE PARTITION ALL;

-- View data in the p2 partition.
m_db=# SELECT * FROM test_p5 PARTITION (p2);
 col1 | col2
------+------
(0 rows)

-- Delete the test_p5 table.
m_db=# DROP TABLE test_p5;
```

Helpful Links

CREATE TABLE PARTITION and DROP TABLE

4.4.2.6.17 ALTER USER

Description

Modifies the attributes of a database user.

Precautions

Session parameters modified by ALTER USER apply to a specified user and take effect in the next session.

Syntax

- Modify user permissions or other information.

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
ALTER USER user_name [ IN DATABASE database_name ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    SET configuration_parameter {{ TO | = } { value | DEFAULT }|FROM CURRENT};
ALTER USER user_name
    [ IN DATABASE database_name ] RESET ALL;
```

The **option** clause is as follows:

```
{ CREATEDB | NOCREATEDB }
    | { CREATEROLE | NOCREATEROLE }
    | { AUDITADMIN | NOAUDITADMIN }
    | { SYSADMIN | NOSYSADMIN }
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | { USEFT | NOUSEFT }
    | {INHERIT | NOINHERIT}
    | { LOGIN | NOLOGIN }
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | NODE GROUP logic_cluster_name
    | ACCOUNT { LOCK | UNLOCK }
```

- Change the value of a specified parameter associated with the user.

```
ALTER USER user_name [ IN DATABASE database_name ]
    SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

- Reset the values of all parameter associated with the user.

```
ALTER USER user_name
    [ IN DATABASE database_name ] RESET ALL;
```

Parameters

- **user_name**

  Specifies the current username.

  Value range: an existing username. For details about the username requirements, see **•user_name**.

- **new_password**

  Specifies a new password.

  The new password must:

  – Differ from the old password.

  – Contain at least eight characters. This is the default length.

  – Differ from the username or the username spelled backward.

  – Contain at least three types of the following four types of characters: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and special characters, including: ~!@#$%^&*()-_=+\|[{}];:,<.>/?

  – Be enclosed by single quotation marks.

  Value range: a string.

- **old_password**

  Specifies the old password.

- **ACCOUNT { LOCK | UNLOCK }**

  – **ACCOUNT LOCK**: locks an account to forbid login to databases.

  – **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.

- **PGUSER**

  In the current version, the **PGUSER** attribute of a user cannot be modified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about other parameters, see "Parameters" in **CREATE ROLE** and **ALTER ROLE**.

Examples

See **Examples** in section "CREATE USER."

Helpful Links

CREATE ROLE, **CREATE USER**, and **DROP USER**

4.4.2.6.18 ALTER VIEW

Description

Modifies the auxiliary attributes of a view. (To modify the query definition of a view, use CREATE OR REPLACE VIEW.)

Precautions

Only the view owner and users with the ALTER permission on the current view can run the **ALTER VIEW** command. System administrators automatically have this permission when separation of duties is disabled. The following are permission constraints depending on the attributes to be modified:

- To modify the schema of a view, you need to be either the view owner or a system administrator and have the CREATE permission on the new schema. However, when separation of duties is enabled, system administrators are restricted from making this modification.
- To modify the owner of a view, you need to be either the view owner or a system administrator. Additionally, you must be a member of the new owner's role and have the CREATE permission on the schema of the view. However, when separation of duties is enabled, system administrators are restricted from making this modification.
- Do not change the type of a column in a view.

Syntax

- Set the default value of a view column.
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name SET DEFAULT expression;
```
- Remove the default value of a view column.
```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name DROP DEFAULT;
```
- Modify the owner of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    OWNER TO new_owner;
```
- Rename a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    RENAME TO new_name;
```
- Set the schema of a view.
```
ALTER VIEW [ IF EXISTS ] view_name
    SET SCHEMA new_schema;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **IF EXISTS**

  If this option is used, no error is generated when the view does not exist, and only a message is displayed.

- **view_name**

  Specifies the view name, which can be schema-qualified.

  Value range: existing view names (character string).

- **column_name**

  Specifies the column name.

  Value range: existing column names (character string).

- **SET/DROP DEFAULT**

  Sets or deletes the default value of a column. This parameter does not take effect.

- **new_owner**

  Sets a username for the new owner of a view. The username must meet the requirements outlined in **•user_name**.

- **new_name**

  Specifies the new view name.

- **new_schema**

  Specifies the new schema of the view.

- **expression**

  Specifies constants, functions, or SQL expressions.

Example

- Rename a view.

```
-- Create the test_tbl table.
m_db=# CREATE TABLE test_tb1(col1 INT,col2 INT);
-- Create a view.
m_db=# CREATE VIEW abc AS SELECT * FROM test_tb1;
-- Rename the view.
m_db=# ALTER VIEW IF EXISTS abc RENAME TO test_v1;
-- Query a view.
m_db=# \dv
              List of relations
 Schema |  Name   | Type | Owner | Storage
--------+---------+------+-------+---------
 public | test_v1 | view | omm   |
(1 row)
```

- Modify the owner of a view.

```
-- Create a user.
m_db=# CREATE ROLE role_test PASSWORD '********';
-- Modify the owner of a view.
m_db=# ALTER VIEW IF EXISTS test_v1 OWNER TO role_test;
-- Query the view information.
m_db=# \dv
                List of relations
 Schema |  Name   | Type |   Owner   | Storage
--------+---------+------+-----------+---------
 public | test_v1 | view | role_test |
(1 row)
```

- Set the schema of a view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a schema.
m_db=# CREATE SCHEMA tcpds;
-- Modify the schema of a view.
m_db=# ALTER VIEW test_v1 SET SCHEMA tcpds;
-- Query the view information.
m_db=# \dv tcpds.test_v1;
              List of relations
 Schema |  Name   | Type |   Owner   | Storage
--------+---------+------+-----------+---------
 tcpds  | test_v1 | view | role_test |
(1 row)
```

Helpful Links

CREATE VIEW and **DROP VIEW**

4.4.2.6.19 ANALYZE

Description

ANALYZE collects statistics about ordinary tables in a database and stores the results in the pg_statistic and pg_statistic_ext system catalogs. After you run the **ANALYZE** command, you can query the collected statistics in the system catalogs or in the system views pg_stats and pg_ext_stats. The execution plan generator uses these statistics to generate the most effective execution plan.

If no parameters are specified, ANALYZE analyzes each table and partitioned table in the database. You can also specify **table_name**, **column_name**, and **partition_name** to limit the analysis to a specified table, column, or partitioned table.

Statistics collected each time are stored in historical statistics tables (**gs_statistic_history**, **gs_statistic_ext_history**, and **gs_statistic_history**). The number of historical tables and the retention period of statistics are specified by the GUC parameters **stats_history_record_limit** and **stats_history_retention_time**.

Precautions

- Non-temporary tables cannot be analyzed in an anonymous block, transaction block, function, or stored procedure. Temporary tables in a stored procedure can be analyzed but their statistics updates cannot be rolled back.
- The ANALYZE VERIFY operation is used to detect abnormal scenarios. The RELEASE version is required. Remote read is not triggered in the ANALYZE VERIFY scenario. Therefore, the remote read parameter does not take effect. If the system detects that a page is damaged due to an error in a key system catalog, the system directly reports an error and halts the detection.
- With no parameter specified, ANALYZE processes all the tables on which the current user has permission to analyze in the current database. With tables specified, ANALYZE processes only the specified tables.
- To analyze a table, you must be the owner of the table or a user granted the ANALYZE permission for the specified table. When separation of duties is disabled, a system administrator has this permission by default. However, database owners are allowed to analyze all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

database-wide analyze operation can only be executed by the system administrator). ANALYZE skips tables on which users do not have permissions.

- ANALYZE does not collect columns for which comparison or equivalent operations cannot be performed, for example, columns of the cursor type.

Syntax

- Collect statistics information about a table.

```
ANALYZE [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ] ;
```

- Collect partition statistics information about a partitioned table.

```
ANALYZE [ VERBOSE ]
    table_name [ ( column_name [, ...] ) ] PARTITION ( partition_name ) ;

Collect table index information.
ANALYZE [ VERBOSE ] GLOBAL INDEX index_name FOR TABLE table_name;
```

  NOTE

  If the keyword PARTITION is used, **partition_name** must be the name of a level-1 partition.

- Collect statistics about multiple columns manually.

```
ANALYZE [ VERBOSE ]
    table_name (( column_1_name, column_2_name [, ...] ));
```

  NOTE

  - If the GUC parameter **enable_functional_dependency** is disabled, the statistics about a maximum of 32 columns can be collected at a time. If **enable_functional_dependency** is enabled, the statistics about a maximum of 4 columns can be collected at a time.
  - You are not allowed to collect statistics about multiple columns in system catalogs and global temporary tables.

- Collect statistics about multiple columns automatically.

  After the **auto_statistic_ext_columns** parameter is enabled and the ANALYZE statement is executed, multi-column statistics are automatically created based on the index prefix of the table. The number of columns in the multi-column statistics cannot exceed the value of **auto_statistic_ext_columns**.

  For example, if index (a, b, c, d) exists in table **t** and **auto_statistic_ext_columns** is set to **4**, multi-column statistics about (a, b), (a, b, c), and (a, b, c, d) are created after table **t** is analyzed.

```
ANALYZE [ VERBOSE ] table_name;
```

Parameters

- **VERBOSE**

  Enables the display of progress messages.

  NOTE

  If **VERBOSE** is specified, ANALYZE displays the progress information, indicating the table that is being processed. Statistics about tables are also displayed.

- **table_name**

  Specifies the name (possibly schema-qualified) of a specific table to analyze. If omitted, all regular tables (but not foreign tables) in the current database are analyzed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Currently, you can use ANALYZE to collect statistics only from row-store tables.

Value range: an existing table name.

- **column_name**, **column_1_name**, **column_2_name**

  Specifies the name of a specific column to analyze. All columns are analyzed by default.

  Value range: an existing column name.

- **partition_name**

  Assumes the table is a partitioned table. You can specify **partition_name** following the keyword PARTITION to analyze the statistics of this table.

  Value range: a partition name of a table.

Examples

-- Create a table.

```
m_db=# CREATE TABLE customer_info
(
WR_RETURNED_DATE_SK       INTEGER                   ,
WR_RETURNED_TIME_SK       INTEGER                   ,
WR_ITEM_SK                INTEGER             NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
);
```

-- Create a partitioned table.

```
m_db=# CREATE TABLE customer_par
(
WR_RETURNED_DATE_SK       INTEGER                   ,
WR_RETURNED_TIME_SK       INTEGER                   ,
WR_ITEM_SK                INTEGER             NOT NULL,
WR_REFUNDED_CUSTOMER_SK   INTEGER
)
PARTITION BY RANGE(WR_RETURNED_DATE_SK)
(
PARTITION P1 VALUES LESS THAN(2452275),
PARTITION P2 VALUES LESS THAN(2452640),
PARTITION P3 VALUES LESS THAN(2453000),
PARTITION P4 VALUES LESS THAN(MAXVALUE)
);
```

-- Run the ANALYZE statement to update statistics.

```
m_db=# ANALYZE customer_info;
m_db=# ANALYZE customer_par;
```

-- Run the ANALYZE VERBOSE statement to update statistics and display table information.

```
m_db=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "public.customer_info"(cn_5002 pid=53078)
INFO:  analyzing "public.customer_info" inheritance tree(cn_5002 pid=53078)
ANALYZE
```

NOTE

If any environment-related fault occurs, check logs of the primary database node.

-- Drop the table.

```
m_db=# DROP TABLE customer_info;
m_db=# DROP TABLE customer_par;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.7 B

4.4.2.7.1 BEGIN

Description

Starts a transaction using BEGIN. If an isolation level and a read/write mode are specified, a new transaction will have those characteristics. You can also specify them using **SET TRANSACTION**.

Precautions

None

Syntax

Start a transaction.

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
   } [, ...]
 ];
```

Parameters

- **WORK | TRANSACTION**

  Specifies an optional keyword in the BEGIN format and does not have any actual effect.

- **ISOLATION LEVEL**

  Specifies a transaction isolation level, which determines what data is visible to a transaction if there are other concurrent transactions.

  NOTE

  The isolation level cannot be changed after data is modified using SELECT, INSERT, DELETE, UPDATE, or COPY in the transaction.

  Value range:

  - **READ COMMITTED**: Only committed data can be read. This is the default value.
  - **READ UNCOMMITTED**: After this isolation level is set, its behavior is the same as that of **READ COMMITTED**.
  - **REPEATABLE READ**: Only the data committed before the transaction starts can be read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported in an M-compatible database. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read-only).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
m_db=# CREATE TABLE tpcds.reason (c1 int, c2 int);

-- Start a transaction in default mode.
m_db=# START TRANSACTION;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Start a transaction in default mode.
m_db=# BEGIN;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ WRITE:
m_db=# BEGIN ISOLATION LEVEL READ COMMITTED READ WRITE;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Drop the tpcds.reason table.
m_db=# DROP TABLE tpcds.reason;

-- Delete a schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

START TRANSACTION

4.4.2.8 C

4.4.2.8.1 CLEAN CONNECTION

Description

Clears idle or invalid network connections between the current CN and other specified CNs or DNs. This statement is used to clear the specified database and idle or invalid connections of a specified user cached in the current CN on a specified CN.

Precautions

- In non-force mode, this function only clears connections between database cluster nodes (CNs/DNs) and does not affect client connections.
- This function clears only idle and invalid connections cached on the CN. Normal connections that are being used are not cleared.
- This function takes effect only on CNs and does not take effect on DNs.
- You can query the **PG_STAT_GET_POOLER_STATUS()** function to check the cache connection and verify the clearing effect.
- You are advised to perform this operation only when the network connection of the database is abnormal.
- If residual connections cannot be cleared on distributed DNs, O&M personnel are advised to use the pg_terminate_session function to clear invalid sessions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
CLEAN CONNECTION
   TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ] [ FORCE ] }
   { FOR DATABASE dbname | TO USER username | FOR DATABASE dbname TO USER username };
```

Parameters

- **CHECK**

  This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will check whether a database is accessed by other sessions before its connections are cleared. If any sessions are detected before **DROP DATABASE** is executed, an error will be reported and the database will not be deleted.

- **FORCE**

  This parameter can be specified only when the node list is TO ALL. If this parameter is specified, all threads related to the specified dbname and username in the current CN receive the SIGTERM signal, the corresponding session is forcibly closed, the transaction is terminated, and the network connection is cleared.

- **COORDINATOR (nodename ,nodename ... }) | NODE (nodename , nodename ...) | ALL**

  This command is used to delete the idle or invalid connections between the current CN node and a specified node. There are three scenarios:

  – **COORDINATOR**: Delete the idle or invalid connections from the current CN to a specified CN.

  – **NODE**: Delete the idle or invalid connections from the current CN to a specified DN.

  – **ALL**: Delete the idle or invalid connections from the current CN to all nodes, including CNs and DNs.

  Value range: **nodename** is an existing node name.

- **dbname**

  Deletes connections to a specified database from the current CN. If this parameter is not specified, connections to all databases will be deleted.

  Value range: an existing database name

- **username**

  Deletes connections to a specified user from the current CN. If this parameter is not specified, connections of all users will be deleted.

  Value range: an existing username

Examples

```
-- Create the test_clean_connection database.
m_db=# CREATE DATABASE test_clean_connection;

-- Create user jack.
m_db=# CREATE USER jack PASSWORD '********';

-- Log in to the database as the user jack in another session, and query the connection information in the view.
m_db=# SELECT datname,usename,application_name,waiting,state
        FROM pg_stat_activity
        WHERE datname = 'test_clean_connection';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
       datname        | usename | application_name | waiting | state
----------------------+---------+------------------+---------+-------
 test_clean_connection | jack    | gsql             | f       | idle
(1 row)

-- If you directly delete the test_clean_connection database, the following error information is displayed:
m_db=# DROP DATABASE test_clean_connection;
ERROR:  Database "test_clean_connection" is being accessed by other users. You can stop all connections by command: "clean connection to all force for database XXXX;" or wait for the sessions to end by querying view: "pg_stat_activity".
DETAIL:  There is 1 other session using the database.

-- Delete the connections to all nodes for logging in to the test_clean_connection database.
-- If the FORCE parameter is not used, connections whose stat is not in other states cannot be deleted.
m_db=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

-- Query the connection for logging in to the test_clean_connection database.
m_db=# SELECT datname,usename,application_name,waiting,state
         FROM pg_stat_activity
         WHERE datname = 'test_clean_connection';
 datname | usename | application_name | waiting | state
---------+---------+------------------+---------+-------
(0 rows)


-- Delete the test_clean_connection database.
m_db=# DROP DATABASE test_clean_connection;

-- Delete user jack.
m_db=# DROP USER jack;
```

4.4.2.8.2 CHECKPOINT

Description

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to a disk.

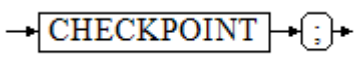
It sets transaction log checkpoints. By default, WALs periodically specify checkpoints in a transaction log. You may use gs_guc to specify GUC parameters **checkpoint_segments**, **checkpoint_timeout**, and **incremental_checkpoint_timeout** to adjust the atomized checkpoint intervals.

Precautions

- Only the system administrator and O&M administrator can call CHECKPOINT.
- CHECKPOINT forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system.

Syntax

```
CHECKPOINT;
```



Parameters

None

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Set a checkpoint.
m_db=# CHECKPOINT;
```

4.4.2.8.3 COMMENT

Description

Defines or changes the comment of an object.

Precautions

- Each object stores only one comment. Therefore, you need to modify a comment and issue a new **COMMENT** command to the same object. To delete the comment, write **NULL** at the position of the text string. When an object is deleted, the comment is automatically deleted.
- Currently, there is no security protection for viewing comments. Any user connected to a database can view all the comments for objects in the database. For shared objects such as databases, roles, and tablespaces, comments are stored globally so any user connected to any database in the cluster can see all the comments for shared objects. Therefore, do not put security-critical information in comments.
- To comment objects, you must be an object owner or user granted the COMMENT permission. By default, system administrators have the permission.
- Roles do not have owners, so the rule for **COMMENT ON ROLE** is that you must be an administrator to comment on an administrator role, or have the CREATE ROLE permission to comment on non-administrator roles. A system administrator can comment on all objects.

Syntax

```
COMMENT ON
{ AGGREGATE func_name ([arg [, ...]])|
COLLATION object_name |
  COLUMN { table_name.column_name | view_name.column_name } |
  CONSTRAINT constraint_name ON table_name |
  DATABASE object_name |
  FUNCTION func_name ([arg [, ...]])|
  INDEX object_name |
  SEQUENCE seq_name |
  USER user_name |
  ROLE object_name |
  SCHEMA object_name |
  TABLE object_name |
  TABLESPACE object_name |
  VIEW object_name |
}
   IS 'text';
```

Parameters

- **func_name**

  Specifies the name of a function.

- **arg**

  Specifies the function variable.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **user_name**

  Value range: an existing username. For details about the username requirements, see **•user_name**.

- **object_name**

  Specifies the name of an object.

- **table_name.column_name**

  view_name.column_name

  Specifies a column name. You can add the table name or view name as the prefix.

- **constraint_name**

  Specifies the name of a table constraint.

- **table_name**

  Specifies the name of a table.

- **text**

  Specifies the comment content.

Examples

```
-- Create a table.
m_db=# CREATE TABLE emp(
    empno varchar(7),
    ename varchar(50),
    job varchar(50),
    mgr varchar(7),
    deptno int
);
-- Add comments to a table.
m_db=# COMMENT ON TABLE emp IS 'Department table';
-- Add comments to columns.
m_db=# COMMENT ON COLUMN emp.empno  IS  'Employee ID';
m_db=# COMMENT ON COLUMN emp.ename  IS  'Employee name';
m_db=# COMMENT ON COLUMN emp.job  IS  'Job';
m_db=# COMMENT ON COLUMN emp.mgr  IS  'Manager ID';
m_db=# COMMENT ON COLUMN emp.deptno  IS  'Department ID';

-- View table comments.
m_db=# \d+
 Schema |    Name     |   Type   | Owner |    Size    |             Storage              | Description
--------+-------------+----------+-------+------------+----------------------------------+-------------
 public | emp         | table    | omm   | 0 bytes    | {orientation=row,compression=no} | Department table

-- View column comments.
m_db=# \d+ emp
                            Table "public.emp"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 empno  | character varying(7)  |           | extended |              | Employee ID
 ename  | character varying(50) |           | extended |              | Employee name
 job    | character varying(50) |           | extended |              | Job
 mgr    | character varying(7)  |           | extended |              | Manager ID
 deptno | integer               |           | plain    |              | Department ID
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE

-- Delete the emp table.
m_db=# DROP TABLE emp;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.8.4 COMMIT

Description

Provides COMMIT to commit all operations of a transaction.

Precautions

Only the creator of the transaction or system administrators can run the COMMIT statement. The creation and commit operations do not have to be in different sessions.

Syntax

```
COMMIT [ WORK | TRANSACTION ];
```

Parameters

- **COMMIT**

  Commits the current transaction and makes all changes made by the transaction become visible to others.

- **WORK | TRANSACTION**

  Specifies an optional keyword, which has no effect except improving readability.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table.
m_db=# CREATE TABLE tpcds.customer_demographics_t2
(
   CD_DEMO_SK               INTEGER               NOT NULL,
   CD_GENDER                CHAR(1)                     ,
   CD_MARITAL_STATUS        CHAR(1)                     ,
   CD_EDUCATION_STATUS      CHAR(20)                    ,
   CD_PURCHASE_ESTIMATE     INTEGER                     ,
   CD_CREDIT_RATING         CHAR(10)                    ,
   CD_DEP_COUNT             INTEGER                     ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                     ,
   CD_DEP_COLLEGE_COUNT     INTEGER
);

-- Start a transaction.
m_db=# START TRANSACTION;

-- Insert data.
m_db=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
m_db=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 'BAD', 1,
0, 0);

-- Commit the transaction to make all changes permanent.
m_db=# COMMIT;

-- Query data.
m_db=# SELECT * FROM tpcds.customer_demographics_t2;

-- Delete the tpcds.customer_demographics_t2 table.
m_db=# DROP TABLE tpcds.customer_demographics_t2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Delete the schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

ROLLBACK

4.4.2.8.5 COMMIT PREPARED

Description

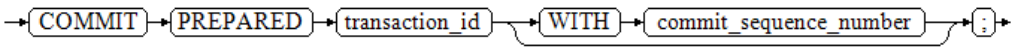
Commits a prepared two-phase transaction. This function is for internal use only. You are advised not to use it.

Precautions

- The function is only available in maintenance mode (when GUC parameter **xc_maintenance_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the transaction creators or system administrators can run the command. The creation and commit operations must be in different sessions.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
COMMIT PREPARED transaction_id [ WITH commit_sequence_number ];
```



Parameters

- **transaction_id**

  Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

- **commit_sequence_number**

  Specifies the sequence number of the transaction to be committed. It is a 64-bit, incremental, unsigned number.

Examples

```
-- Start.
gaussdb=# BEGIN;

-- Prepare a transaction whose identifier is trans_test.
gaussdb=# PREPARE TRANSACTION 'trans_test';

-- Create a table.
gaussdb=# CREATE TABLE item1(id int);

--Commit the transaction whose identifier is trans_test.
gaussdb=# COMMIT PREPARED 'trans_test';
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
gaussdb=# DROP TABLE item1;
```

Helpful Links

ROLLBACK PREPARED

4.4.2.8.6 COPY

Description

Copies data between tables and files.

COPY FROM copies data from a file to a table, and COPY TO copies data from a table to a file.

Precautions

- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** command. When the **enable_copy_server_files** parameter is enabled, users with the SYSADMIN permission or users who inherit the permissions of the built-in role gs_role_copy_files are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** command. By default, database configuration files and key files are not allowed, and you can run the **COPY FROM FILENAME** or **COPY TO FILENAME** command for certificate files and audit logs to prevent unauthorized users from viewing or modifying sensitive files. When **enable_copy_server_files** is enabled, an administrator can use the GUC parameter **safe_data_path** to set the path for common users to import and export to the subpath of the set path. If this GUC parameter is not set (by default), the path used by common users is not blocked. This parameter reports an error for **...** in the path of the COPY statement.
- COPY applies only to tables but not views.
- COPY TO requires the SELECT permission on the table to be read, and COPY FROM requires the INSERT permission on the table to be inserted.
- If a list of columns is specified, COPY copies only the data of the specified columns between the file and the table. If a table has any columns that are not in the column list, COPY FROM inserts default values into those columns.
- If a data source file is specified, the server must be able to access the file. If **STDIN** is specified, data flows between the client and the server. When entering data, use the **TAB** key to separate the columns of the table and use a backslash and a period (\.) in a new row to indicate the end of the input.
- COPY FROM throws an error if any row in the data file contains more or fewer columns than expected.
- The end of the data can be represented by a line that contains only backslashes and periods (\.). If data is read from a file, the end flag is unnecessary. If data is copied between client applications, an end tag must be provided.
- In COPY FROM, **\N** is an empty string. To enter the actual value **\N**, use **\\N**.
- COPY FROM does not support data preprocessing during data import, such as expression operation and default value filling. If you need to preprocess data

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

during the import, you need to import the data to a temporary table and then run SQL statements to insert the data into the table through operations. However, this method causes I/O expansion and reduces the import performance.

- When a data format error occurs during COPY FROM execution, the transaction is rolled back. However, the error information is insufficient, making it difficult to locate the error data from a large amount of raw data.
- COPY FROM and COPY TO apply to low concurrency and local import and export of a small amount of data.
- When COPY is used in binary format, transcoding in distributed mode is not supported.
- **COPY** is a server command and its operating environment is the same as that of the database server process. **\COPY** is a client meta-command and its operating environment is the same as that of gsql on the client. Note that when the database and gsql are used in the sandbox environment, the **COPY** and **\COPY** commands both use the paths in the sandbox. When the database is used in the sandbox environment and gsql is used outside the sandbox, the **COPY** command uses the path inside the sandbox, and the **\COPY** command uses the path outside the sandbox.
- When executing **COPY** to import data to a base table with a GSI, the **enable_stream_operator** parameter must be enabled to achieve optimal data import performance.
- When exporting data of the float type, you are advised to set **extra_float_digits** to **3** to avoid loss of significant digits and ensure consistency between the data before and after export and import.
- In M-compatible mode, you are advised not to perform COPY FROM/TO between databases with different **lower_case_table_names** parameter settings.

Syntax

- Copy data from a file to a table.

```
COPY table_name [ ( column_name [, ...] ) ] [WITH OIDS]
    FROM { 'filename' | STDIN } [LOAD] [ LOAD_DISCARD 'discard_path'] [LOAD_BAD 'bad_path']
    [ [ WITH ] ( option [, ...] ) | copy_option ];
```

  NOTE

  In the preceding syntax, copy_option is incompatible with option.

- Copy data from a table to a file.

```
COPY table_name [ ( column_name [, ...] ) ] [WITH OIDS]
    TO { 'filename' | STDOUT } [LOAD]
  [ [ WITH ] ( option [, ...] ) | copy_option ];

COPY query
    TO { 'filename' | STDOUT }
  [ [ WITH ] ( option [, ...] ) | copy_option ];
```

  NOTE

  copy_option is the native parameter, while **option** is the parameter imported by a compatible foreign table.

  The syntax of the optional parameter **option** is as follows:

```
FORMAT 'format_name'
| FORMAT binary
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
| DELIMITER 'delimiter_character'
| NULL  'null_string'
| QUOTE 'quote_character'
| HEADER [ boolean ]
| USEEOF [ boolean ]
| ESCAPE 'escape_character'
| FORCE_NOT_NULL ( column_name [, ...] )
| FORCE_QUOTE { column_name [, ...] | * }
| DATE_FORMAT 'date_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

The syntax of the optional parameter **copy_option** is as follows:

```
NULL [ AS ] 'null_string'
| FORCE_NOT_NULL column_name [, ...]
| BINARY
| CSV
| ESCAPE [ AS ] 'escape_character'
| ENCODING 'encoding_name'
| DELIMITER [ AS ] 'delimiter_character'
| ROWS 'string'
| DATE_FORMAT 'date_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
| SKIP int_number
| WHEN { ( start - end ) | column_name } { = | != } 'string'
| SEQUENCE ( { column_name ( integer [, incr] ) [, ...] } )
```

Parameters

- **query**

  Specifies that the results will be copied.

  Valid value: a **SELECT** or **VALUES** command in parentheses

- **table_name**

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- **column_name**

  Specifies an optional list of columns to be copied.

  Value range: any columns. All columns will be copied if no column list is specified.

- **STDIN**

  Specifies that input comes from the standard input. In the input, table columns are separated by tabs and each row ends with a backslash and a period (\.).

- **STDOUT**

  Specifies that output goes to the standard output.

- **LOAD**

  Specifies whether the COPY syntax is called by gs_loader or directly called by users. This parameter is not recommended because it is not used to actively call APIs.

- **LOAD_DISCARD 'discard_path'**

  Specifies the path where the **discard** file is generated on the gs_loader client. This parameter is not recommended because it is not used to actively call APIs.

- **LOAD_BAD 'bad_path'**

  Specifies the path where the **bad** file is generated on the gs_loader client. This parameter is not recommended because it is not used to actively call APIs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **OPTION { option_name ' value ' }**

  Specifies all types of parameters of a compatible foreign table.

  – FORMAT

  Specifies the format of the source data file in the foreign table.

  Value range: **CSV**, **TEXT**, and **BINARY**

  - The CSV file can process newline characters efficiently, but cannot process certain special characters well.
  - The TEXT file can process certain special characters efficiently, but cannot process newline characters well.
  - All data in the BINARY file is stored/read as binary format rather than as text. It is faster than the text and CSV formats, but a binary-format file is less portable.

  Default value: **TEXT**

  – DELIMITER

  Specifies the character that separates columns within each row (line) of the file.

  NOTE

  - The value of **delimiter** cannot be **\r** or **\n**.
  - A delimiter cannot be the same as the null value. The delimiter for the CSV format cannot be same as the **quote** value.
  - The delimiter for the TEXT format data cannot contain lowercase letters, digits, or special characters (.\).
  - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
  - You are advised to use multi-character delimiters or invisible delimiters. For example, you can use multi-characters (such as $^&) and invisible characters (such as 0x07, 0x08, and 0x1b).
  - To use a tab to isolate CSV data, set **delimiter** to **E'\t'**.

  Value range: a multi-character delimiter within 10 bytes

  Default value:

  - A tab character in text format
  - A comma (,) in CSV format

  – NULL

  Specifies the string that represents a null value.

  Value range:

  - A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
  - A null value cannot be the same as the **delimiter** or **quote** value.

  Default value:

  - The default value for the CSV format is an empty string without quotation marks.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The default value for the TEXT format is **\N**.

– HEADER

Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV files.

When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

Value range: **true**/**on** or **false**/**off**.

Default value: **false**

– ESCAPE

Specifies an escape character for a CSV file. The value must be a single-byte character.

Default value: '' If the value is the same as that of **quote**, it will be replaced by **'\0'**.

– FORCE_NOT_NULL (column_name [, ...])

Assigns a value to a specified column in **CSV COPY FROM** mode.

Value range: an existing column.

– DATE_FORMAT

Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

Value range: a valid DATE value For details, see **Datetime Functions**.

NOTE

You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

- **COPY_OPTION { option_name ' value ' }**

Specifies all types of native parameters of COPY.

– NULL null_string

Specifies the string that represents a null value.

NOTICE

When using COPY FROM, any data item that matches this string will be stored as a null value, so make sure that you use the same string as you used with COPY TO.

Value range:

- A null value cannot be **\r** or **\n**. The maximum length is 100 characters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- A null value cannot be the same as the **delimiter** or **quote** value.

Default value:

- The default value for the TEXT format is **\N**.
- The default value for the CSV format is an empty string without quotation marks.

– FORCE NOT NULL column_name [, ...]

Assigns a value to a specified column in **CSV COPY FROM** mode. If the column is null, its value is regarded as a string of 0 characters.

Value range: an existing column.

– BINARY

Specifies that data is stored and read in binary mode instead of text mode.

NOTE

- In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**.
- When **BINARY** is specified, **CSV** and **TEXT** cannot be specified through option or copy_option.
- If the GUC parameter **support_binary_copy_version** is set to **'header_encoding'**, the header of the binary file exported using COPY TO contains the server encoding information of the database. When the binary file is imported using COPY FROM, the consistency of the encoding information is checked to ensure that the encoding information in the file must be consistent with that on the server.
- If the GUC parameter **copy_special_character_version** is set to **'no_error'**, invalid characters will not be checked during the import and will be displayed as garbled characters in query results. Exercise caution when enabling this parameter. You can use the **LOG ERRORS** or **LOG ERRORS DATA** parameter in the COPY statement to code errors to an error table.
- In binary mode, **copy_special_character_version** is set to **'no_error'**, and it takes effect only for fields of the TEXT, CHAR, VARCHAR, and NVARCHAR2 types.

– CSV

Enables the CSV mode. When **CSV** is specified, **BINARY** and **TEXT** cannot be specified through option or copy_option.

– ESCAPE [AS] 'escape_character'

Specifies an escape character for a CSV file. The value must be a single-byte character.

Default value: **''**. If the value is the same as that of **quote**, it will be replaced by **'\0'**.

– ENCODING 'encoding_name'

Specifies the name of a file encoding format.

Value range: a valid encoding format

Default value: current encoding format

– DATE_FORMAT 'date_format_string'

Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

Value range: a valid DATE value For details, see **Datetime Functions**.

NOTE

You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

- SKIP int_number

  Specifies that the first *int_number* rows of the data file are skipped during data import.

- WHEN { (start - end) | column_name } { = | != } 'string'

  When data is imported, each row of data is checked. Only the rows that meet the WHEN condition are imported to the table.

- SEQUENCE ({ column_name (integer [, incr]) [, ...] })

  During data import, columns modified by SEQUENCE do not read data from the data file. The values are incremented by the value of **incr** based on the specified integer. If **incr** is not specified, the values are incremented from 1 by default.

NOTE

COPY FROM does not support expression conversion for distribution keys.

The following special backslash sequences are recognized by COPY FROM:

- **\b**: Backslash (ASCII 8)
- **\f**: Form feed (ASCII 12)
- **\n**: Newline character (ASCII 10)
- **\r**: Carriage return character (ASCII 13)
- **\t**: Tab (ASCII 9)
- **\v**: Vertical tab (ASCII 11)
- **\digits**: Backslash followed by one to three octal digits specifies that the ASCII value is the character with that numeric code.
- **\xdigits**: Backslash followed by an x and one or two hex digits specifies the character with that numeric code.

Permission Control Examples

```
m_db=# copy t1 from '/home/xy/t1.csv';
ERROR:  COPY to or from a file is prohibited for security concerns
HINT:  Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
m_db=# grant gs_role_copy_files to xxx;
```

This error occurs because a non-initial user does not have the COPY permission. To solve this problem, enable the **enable_copy_server_files** parameter. Then, the administrator can use the COPY function, and common users need to join the **gs_role_copy_files** group.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create the tpcds.ship_mode table.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# CREATE TABLE tpcds.ship_mode
(
   SM_SHIP_MODE_SK          INTEGER              NOT NULL,
   SM_SHIP_MODE_ID          CHAR(16)             NOT NULL,
   SM_TYPE                CHAR(30)                ,
   SM_CODE                 CHAR(10)                ,
   SM_CARRIER              CHAR(20)                ,
   SM_CONTRACT               CHAR(20)
) DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);

-- Insert a single data record into the tpcds.ship_mode table.
m_db=# INSERT INTO tpcds.ship_mode VALUES (1,'a','b','c','d','e');

-- Copy data from the tpcds.ship_mode file to the /home/omm/ds_ship_mode.dat file.
m_db=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

-- Output tpcds.ship_mode to STDOUT.
m_db=# COPY tpcds.ship_mode TO STDOUT;

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The delimiter is ',' (delimiter',') and the encoding format is UTF8 (encoding'utf8').
m_db=# COPY tpcds.ship_mode TO STDOUT WITH (delimiter ',', encoding 'utf8');

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The import format is CSV (format'CSV'), and the exported content of the SM_SHIP_MODE_SK column is enclosed in quotation marks (force_quote(SM_SHIP_MODE_SK)).
m_db=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

-- Create the tpcds.ship_mode_t1 table.
m_db=# CREATE TABLE tpcds.ship_mode_t1
(
   SM_SHIP_MODE_SK          INTEGER              NOT NULL,
   SM_SHIP_MODE_ID          CHAR(16)             NOT NULL,
   SM_TYPE                CHAR(30)                ,
   SM_CODE                 CHAR(10)                ,
   SM_CARRIER              CHAR(20)                ,
   SM_CONTRACT               CHAR(20)
) DISTRIBUTE BY HASH(SM_SHIP_MODE_SK );

-- Copy data from STDIN to the tpcds.ship_mode_t1 table.
m_db=# COPY tpcds.ship_mode_t1 FROM STDIN;

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table.
m_db=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the import format set to TEXT (format 'text'), the delimiter set to '\t' (delimiter E'\t'), excessive columns ignored (ignore_extra_data 'true'), and characters not escaped (noescaping 'true').
m_db=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

-- Delete tables and the schema.
m_db=# DROP TABLE tpcds.ship_mode;
m_db=# DROP TABLE tpcds.ship_mode_t1;
m_db=# DROP SCHEMA tpcds;
```

4.4.2.8.7 CREATE AUDIT POLICY

Description

Creates a unified audit policy. The current version does not support this syntax.

Precautions

- Only the user with the POLADMIN or SYSADMIN permission, or initial user has the permission to create and maintain audit policies.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Before creating an audit policy, the GUC parameter **enable_security_policy** must be set to **on** to make the masking policy takes effect.
- A SYSADMIN or POLADMIN can access the GS_AUDITING_POLICY, GS_AUDITING_POLICY_ACCESS, GS_AUDITING_POLICY_PRIVILEGES, and GS_AUDITING_POLICY_FILTERS system catalogs to query the created audit policies.
- The audit policy name must be unique to avoid conflicts with existing policies. You can use IF NOT EXISTS to check whether the specified audit policy exists to avoid repeated creation.

Syntax

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { { privilege_audit_clause | access_audit_clause }
[, ... ] [ filter_group_clause ] [ ENABLE | DISABLE ] };
```

- privilege_audit_clause
  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- access_audit_clause
  ```
  ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- filter_group_clause
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ... ] ) } [, ... ]
  ```
- DDL
  ```
  { ALTER | ANALYZE | COMMENT | CREATE | DROP | GRANT | REVOKE | SET | SHOW }
  ```
- DML
  ```
  { COPY | DEALLOCATE | DELETE | EXECUTE | REINDEX | INSERT | PREPARE | SELECT | TRUNCATE |
  UPDATE }
  ```
- FILTER_TYPE
  ```
  { APP | ROLES | IP }
  ```

Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string of no more than 63 characters. It must comply with the **naming conventions**. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the audit policy name. If an audit policy name contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create an audit policy name that contains uppercase letters, enclose the audit policy name with double quotation marks ("").

  NOTE

  The identifier must be lowercase letters, uppercase letters, underscores (_), digits (0–9), or dollar signs ($) and must start with a letter or underscore (_).

- **resource_label_name**

  Specifies the resource label name.

- **DDL**

  Specifies the operations that are audited in the database: CREATE, ALTER, DROP, ANALYZE, COMMENT, GRANT, REVOKE, SET, and SHOW.

  If this parameter is set to **ANALYZE**, both ANALYZE and VACUUM operations are audited.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **DML**

  Specifies the operations that are audited within the database: SELECT, COPY, DEALLOCATE, DELETE, EXECUTE, INSERT, PREPARE, REINDEX, TRUNCATE, and UPDATE.

- **ALL**

  Specifies all operations supported by the specified DDL or DML statements in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.

- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policy, including **APP**, **ROLES**, and **IP**.

- **filter_value**

  Specifies the detailed information to be filtered.

- **ENABLE|DISABLE**

  Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

- Creates audit policy for executing **CREATE** on the database.

```
-- Create the adt1 policy.
m_db=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- View the adt1 policy.
m_db=# SELECT * FROM GS_AUDITING_POLICY;
 polname | polcomments |         modifydate         | polenabled
---------+-------------+----------------------------+------------
 adt1    |             | 2023-11-06 16:41:40.947417 | t

-- Check the location where the audit policy is stored.
m_db=# SHOW audit_directory;

-- Delete the audit policy adt1.
m_db=# DROP AUDIT POLICY adt1;
```

- Create an audit policy to audit only the CREATE operation performed by the **dev_audit** user.

```
-- Create user dev_audit.
m_db=# CREATE USER dev_audit PASSWORD '********';

-- Create the tb_for_audit table.
m_db=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

-- Create the adt_lb0 resource label based on the tb_for_audit table.
m_db=# CREATE RESOURCE LABEL adt_lb0 add TABLE(public.tb_for_audit);

-- Create the adt2 audit policy for the CREATE operation on the adt_lb0 resource.
m_db=# CREATE AUDIT POLICY adt2 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON
ROLES(dev_audit);

-- Delete the audit policy adt2.
m_db=# DROP AUDIT POLICY adt2;

-- Delete the tb_for_audit table.
m_db=# DROP TABLE tb_for_audit;

-- Delete the dev_audit user.
m_db=# DROP USER dev_audit;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Create an audit policy to audit only the SELECT, INSERT, and DELETE operations performed on the **adt_lb0** resource by user **dev_audit** using client tool gsql on the servers whose IP addresses are **10.20.30.40** and **127.0.0.0/24**.

```
-- Create user dev_audit.
m_db=# CREATE USER dev_audit PASSWORD '********';

-- Create the audit policy adt3.
m_db=# CREATE AUDIT POLICY adt3 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON
LABEL(adt_lb0), DELETE FILTER ON ROLES(dev_audit), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

-- Delete the audit policy adt3.
m_db=# DROP AUDIT POLICY adt3;

-- Delete the dev_audit user.
m_db=# DROP USER dev_audit;
```

Helpful Links

ALTER AUDIT POLICY and **DROP AUDIT POLICY**

4.4.2.8.8 CREATE DATABASE

Description

DATABASE is the same as SCHEMA. This statement creates a schema.

Precautions

- A user who has the CREATEDB permission or a system administrator can create a database.
- During the database creation, an error message indicating that permission denied is displayed, possibly because the permission on the data directory in the file system is insufficient. If an error message, indicating no space left on device is displayed, the possible cause is that the disk space is used up.

Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] database_name [AUTHORIZATION role_id]
    [[create_option] [,...]]
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Parameters

- **database_name**

  Specifies the database name.

  Value range: a string complying with **Identifier Description**.

- **AUTHORIZATION role_id**

  (Optional) Specifies the owner of the created database. If omitted, the default owner is the current user.

  Value range: an existing username

- **COLLATE [=] collation_name**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

(Optional) Specifies the character set used by the new database. For example, set this parameter by using **collate = 'zh_CN.gbk'**.

The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the character set of the template database is used.

Value range: See **Database-level Character Sets and Collations**.

- **{CHAR SET | CHARSET | CHARACTER SET} [=] charset_name**

  (Optional) Specifies the character class used by the new database. For example, set this parameter by using **CHARSET = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

The restrictions on character encoding are as follows:

- If the locale is set to **C** (or **POSIX**), all encoding types are allowed. For other locale settings, the character encoding must be the same as that of the locale.
- If the character encoding mode is SQL_ASCII and the modifier is an administrator, the character encoding mode can be different from the locale setting.
- The encoding and region settings must match the template database, except that **template0** is used as a template. This is because other databases may contain data that does not match the specified encoding, or may contain indexes whose sorting order is affected by **LC_COLLATE** and **LC_CTYPE**. Copying this data will invalidate the indexes in the new database. **template0** does not contain any data or indexes that may be affected.

Examples

```
-- Create and switch to the test database.
m_db=# CREATE DATABASE test1;
m_db=# USE test1;
-- Create the a table.
m_db=# CREATE TABLE a(id int);
m_db=# DROP TABLE a;
-- Delete the database test1.
m_db=# USE public;
m_db=# DROP DATABASE test1;
```

Helpful Links

ALTER DATABASE and **DROP DATABASE**

Suggestions

- **create database**

  Database cannot be created in a transaction.

- **ENCODING LC_COLLATE LC_CTYPE**

  If the new database encoding does not match the template database (SQL_ASCII) (**'GBK'**, **'UTF8'**, **'LATIN1'**, **'GB18030_2022'**, **'ZHS16GBK'**, or **'GB18030'**), **template [=] template** must be specified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.8.9 CREATE EXTENSION

Description

Currently, this syntax is not supported. Installs an extension. This feature is for internal use only. You are advised not to use it.

Precautions

- Before running the **CREATE EXTENSION** command to load an extension into the database, you must install the support file for the extension first.
- The **CREATE EXTENSION** command installs a new extension to a database. Ensure that no extension with the same name has been installed.
- Installing an extension means executing an extended script file that creates an SQL entity, such as a function, data type, operator, and index-supported method.
- Installing an extension requires the same permissions as creating its component objects. For most extensions, this means that the superuser or database owner's permissions are required. For subsequent permission checks and entities created by the extension script, the role that runs the **CREATE EXTENSION** command becomes the owner of the extension.
- During the execution of CREATE EXTENSION, if the database contains database objects with the same name as that in the EXTENSION, such as synonyms, operators, directories, functions, stored procedures, views, and tables, the execution will fail.
- Do not directly create extensions for the database to avoid unexpected errors and incompatibility problems after the upgrade. To create an extension, set **enable_extension** to **true**.
- During the execution of **CREATE EXTENSION**, if the GUC parameter **enable_object_special_character** is set to **off** and "@extschema@" is used in the extension script file, the value of the **schema** parameter in the extension support file cannot contain any special characters in ["$'\].

Syntax

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ];
```

Parameters

- **IF NOT EXISTS**

  If an extension with the same name exists in the system, no error is reported. However, a message is displayed. Note that this parameter does not ensure that the existing extensions of the system are the same as those created by the script.

- **extension_name**

  Specifies the name of the extension to be installed. The database creates the extension by using the information in the **SHAREDIR/extension/extension_name.control** file.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **schema_name**

  The extension instance is installed in this schema, and the extended content can be reinstalled. The specified schema must exist. If it is not specified, the extended control file does not specify a schema either. In this case, the default schema is used.

  ⚠ CAUTION

  Extensions are not considered to be in any schema (no restriction is posed on the name of extensions within the scope of a database), but an extension instance belongs to a schema.

- **version**

  Specifies the version of the extension to be installed, which can be written as an identifier or a string. The default version is specified in the extended control file.

- **old_version**

  If you want to upgrade the content that is not contained in the **old style** module, you must specify **FROM old_version**. This option makes **CREATE EXTENSION** run an installation script to install new content into the extension instead of creating an entity. Note that **SCHEMA** specifies the schema that includes these existing entities.

Examples

Install an extension in the current database. For example, install **security_plugin**:

```
m_db=# CREATE EXTENSION IF NOT EXISTS security_plugin;
m_db=# DROP EXTENSION security_plugin;
```

Helpful Links

ALTER EXTENSION and **DROP EXTENSION**

4.4.2.8.10 CREATE FUNCTION

Description

Creates a function.

NOTE

This function is for internal use only and not available to users.

Precautions

- If the parameters or return values of a function have precision, the precision is not checked.
- When creating a function, you are advised to explicitly specify the schemas of tables in the function definition. Otherwise, the function may fail to be executed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If a function has output parameters, the GUC parameter **set behavior_compat_options** must be set to **'proc_outparam_override'** for the output parameters to take effect. When the function is called using SELECT or CALL, an actual parameter must be provided in the position of the output parameter. Otherwise, the function fails to be called.
- Only the functions compatible with PostgreSQL can be overloaded.
- You cannot create overloaded functions with different formal parameter names (the function name and parameter list type are the same).
- Formal parameters cannot be overloaded if only the custom ref cursor type is different from the sys_refcursor type.
- Function overloading is not supported if only the returned data types are different.
- Function overloading is not supported if only the default values are different.
- When an overloaded function is called, the variable type must be specified.
- If an undeclared variable is used in a function, an error is reported when the function is called.
- You can use the SELECT statement to specify different parameters using identical functions.
- When you create a function, you cannot insert other agg functions out of the avg function or other functions.
- By default, the permissions to execute new functions are granted to PUBLIC users. By default, a user inherits the permissions of the PUBLIC role. Therefore, the user has the permission to execute a function and view the definition of the function. In addition, to execute the function, the user must have the USAGE permission on the schema to which the function locates. You can revoke the default execution permissions from the PUBLIC role when creating a function and grant the function execute permission to users as needed. To avoid the time window during which new functions can be accessed by all users, create functions and set function execution permissions in a transaction. After the database object isolation attribute is enabled, common users can view only the definitions of functions that they have permission to execute.
- When functions without parameters are called inside another function, you can omit brackets and call functions using their names directly.
- When other functions with output parameters are called in a function and an assignment expression, set the GUC parameter **set behavior_compat_options** to **'proc_outparam_override'**, define variables of the same type as the output parameters in advance, and use the variables as output parameters to call other functions with output parameters for the output parameters to take effect. Otherwise, the output parameters of the called functions will be ignored.
- Users granted with the CREATE ANY FUNCTION permission can create or replace functions in the user schemas.
- The default permission on a function is AUTHID CURRENT_USER. To change the permission to AUTHID DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- For PL/SQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the behaviors of **out** and **inout** change. In the functions, **return**, **out**, and **inout** can be returned at the same time. Before

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the parameter is set to **'proc_outparam_override'**, only **return** is returned. For details, see **Examples**.

- For PL/SQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the restrictions are as follows:
  - If a function with the **out/inout** parameter already exists in the same schema or package, you cannot create another function with the same name with the **out/inout** parameter.
  - In some scenarios, functions cannot be used in expressions (compared with those before the parameter is enabled), for example, call function.
  - Functions without return cannot be called. The perform function can be used to call functions.
  - After the GUC parameter is set to **proc_outparam_override**, if the return type of the function is setof, the output parameter will not take effect.
- When creating a function, you cannot use the function itself as the default value of input parameter.
- The function with **OUT** parameter cannot be called by SQL statement.
- The function with **OUT** parameter cannot be called by SELECT INTO syntax.
- Functions with **OUT** parameters cannot be called in nested mode.

  Example:

  ```
  b := func(a,func(c,1));
  ```

  Should be changed to:

  ```
  tmp := func(c,1); b := func(a,tmp);
  ```
- When a function is created, the type of the return value of the function is not checked.
- If a function with the definer specified is created in a schema of another user, the function will be executed by another user, which may cause unauthorized operations. Therefore, exercise caution when performing this operation.
- In the schema of the O&M administrator, only the initial user, system administrator, and schema owner can create objects. The schema that does not allow other users to create and modify objects is the schema of the O&M administrator.
- If the **out** parameter is used as the output parameter in an expression, the expression does not take effect in the following scenarios: (a) The execute immediate sqlv using func syntax is used to execute a function. (b) The select func into syntax is used to execute a function. (c) DML statements such as INSERT and UPDATE are used to execute a function. (d) The select where a=func() statement is used. (e) When a function with the **out** output parameter is used as an input parameter, that is, **fun (func (out b), a)**, the **out b** parameter does not take effect.
- In the RETURN statement of a function, when the constructor of the composite type is called, if the actual return type is different from the defined return type, the result can be implicitly converted to the defined return type. Cross-schema call is supported, for example: **RETURN schema.record;**. Cross-database call is not supported, for example, **RETURN package.schema.record;**. In the RETURN statement of a function, when the function is called, the function with the OUT parameter in an expression of an arithmetic operation is not supported, for example, **RETURN func(c out) + 1**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When the complex function is called, for example, func(x).a, a composite type is returned. Cross-schema call is supported, but the database.schema.package.func(x).b call is not supported.
- Functions contain syntax and functions that use GUC parameters to control features. If GUC parameters are modified in a session, the result may be inconsistent with the expected result. If the parameters are modified, the function may retain the behavior before the modification. Therefore, exercise caution when modifying GUC parameters.

Syntax

```
CREATE OR REPLACE FUNCTION function_name
   ( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ]} [, ...] ] )
   [ RETURNS rettype | RETURNS TABLE ( { column_name column_type } [, ...] ) ]
   LANGUAGE lang_name
   [
     {IMMUTABLE | STABLE | VOLATILE}
      | {SHIPPABLE | NOT SHIPPABLE}
      | [ NOT ] LEAKPROOF
      | STRICT
      | {AUTHID DEFINER | AUTHID CURRENT_USER}
      | not fenced
      | COST execution_cost
      | ROWS result_rows
    ][...]
   {
      AS 'definition'
   };
```

Parameters

- **function_name**

  Specifies the name of the function to be created (optionally schema-qualified).

  Value range: a string complying with **Identifier Description**. The value contains a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function name.

  NOTE

  You are advised not to create a function with the same name as a system function. Otherwise, you need to specify the schema of the function when calling the function.

- **argname**

  Specifies the parameter name of the function.

  Value range: a string complying with **Identifier Description**. The value contains a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function parameter name.

- **argmode**

  Specifies the parameter mode of the function.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameter of the **OUT** mode can be followed by **VARIADIC**. The parameters of **OUT** and **INOUT** cannot be used in the function definition of RETURNS TABLE.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

VARIADIC specifies parameters of the array type.

- **argtype**

  Specifies the data type of a function parameter. **%TYPE** or **%ROWTYPE** can be used to indirectly reference a variable or table type.

- **expression**

  Specifies the default expression of a parameter.

  NOTE

  - If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the default expression is not supported when the parameter is in INOUT mode.
  - It is recommended that you define all default parameters after all non-default parameters.
  - If a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported.
  - If **proc_uncheck_default_param** is enabled and a function with default parameters is called, input parameters are added to the function from left to right. The number of defaulted inputs depends on the number of default parameters. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter.
  - When **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s1**, **proc_outparam_override** is disabled, and the function parameters include the output parameter **out** and **default**, the default value cannot be used.

- **rettype**

  Specifies the return data type. Same as **argtype**, **%TYPE** or **%ROWTYPE** can also be used to indirectly reference types.

  When there is **OUT** or **INOUT** parameter, the RETURNS clause can be omitted. If the clause is not omitted, the result type of the clause must be the same as that of the output parameter. If there are multiple output parameters, the result type of the clause is RECORD. Otherwise, the result type of the clause is the same as that of a single output parameter.

  The SETOF modifier indicates that the function will return a set of items, rather than a single item.

  NOTE

  In **FUNCTION argtype** and **rettype** outside **PACKAGE**, **%TYPE** cannot reference the type of the *PACKAGE* variable.

- **column_name**

  Specifies the column name.

- **column_type**

  Specifies the column type.

- **definition**

  Specifies a string constant defining a function. Its meaning depends on the language. It can be an internal function name, a path pointing to a target file, an SQL query, or text in a procedural language.

- **LANGUAGE lang_name**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the name of the language that is used to implement the function. It can be **SQL**, **internal**, or the name of a customized process language. To ensure downward compatibility, the name can use single quotation marks. Contents in single quotation marks must be capitalized.

NOTE

- When an internal function is defined, if AS specifies the function as an internal system function, the parameter type, number of parameters, and return value type of the new function must be the same as those of the internal system function, and the user who creates the internal function must have the permission to execute the internal system function.
- Only users with the SYSADMIN permission can create internal functions.

- **IMMUTABLE**

  Specifies that the function always returns the same result if the parameter values are the same.

- **STABLE**

  Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

  Specifies that the function value can change in a single table scan and no optimization is performed.

- **SHIPPABLE|NOT SHIPPABLE**

  Specifies whether the function can be pushed down for execution. This port is reserved and is not recommended.

- **NOT FENCED**

  Specifies whether the user-defined C function is executed in not-fenced mode. This port is reserved and is not recommended.

- **LEAKPROOF**

  Specifies that the function has no side effects. **LEAKPROOF** can be set only by the system administrator.

- **STRICT**

  Specifies that the function always returns null whenever any of its parameters is null. If this parameter is specified, the function is not executed when there are null parameters; instead a null result is returned automatically.

- **AUTHID CURRENT_USER**

  Specifies that the function will be executed with the permissions of the user who calls it. This parameter can be omitted.

- **AUTHID DEFINER**

  Specifies that the function will be executed with the permissions of the user who created it.

- **COST execution_cost**

  Estimates the execution cost of a function.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: ≥ 0

- **ROWS result_rows**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

Value range: ≥ 0. The default value is **1000**.

NOTICE

When you perform password-related operations, such as user creation, password change, and encryption/decryption, in a function body, the password will be recorded in the system catalogs and logs in plaintext. To prevent sensitive information leakage, you are advised not to perform operations on the function body related to sensitive information, such as passwords or keys.

Examples

```
-- Define a function as SQL query.
m_db=# CREATE OR REPLACE FUNCTION func_add_sql(integer, integer) RETURNS bigint
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    SHIPPABLE
    STRICT
    AUTHID DEFINER
    not fenced
    COST 10000;

-- Add an integer by parameter name using PL/pgSQL.
m_db=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
    BEGIN
        RETURN i + 1;
    END;
    $$ LANGUAGE plpgsql;

-- Create an internal function.
m_db=# CREATE OR REPLACE FUNCTION func_test(name) RETURNS text LANGUAGE INTERNAL AS
'series_internal';

-- Create a function with the same name.
m_db=# CREATE OR REPLACE FUNCTION func_add(int) RETURNS int AS $$
    BEGIN
        RETURN $1+10;
    END;
    $$ LANGUAGE PLPGSQL;

m_db=# CREATE OR REPLACE FUNCTION func_add(int,int) RETURNS int AS $$
    BEGIN
        RETURN $1+$2;
    END;
    $$ LANGUAGE PLPGSQL;

-- Call a function.
m_db=# SELECT func_add_sql(3, 7);
 func_add_sql
--------------
           10
(1 row)

m_db=# SELECT func_increment_plsql(5);
 func_increment_plsql
----------------------
                    6
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT func_add(4);
 func_add
----------
       14
(1 row)

m_db=# SELECT func_add(2,7);
 func_add
----------
        9
(1 row)

-- Delete the function.
m_db=# DROP FUNCTION func_add_sql;
m_db=# DROP FUNCTION IF EXISTS func_increment_plsql;
m_db=# DROP FUNCTION func_test() CASCADE;
m_db=# DROP FUNCTION func_add(int);
m_db=# DROP FUNCTION IF EXISTS func_add(int,int) RESTRICT;
```

Helpful Links

DROP FUNCTION

4.4.2.8.11 CREATE GLOBAL INDEX

Description

A global secondary index (GSI) is created on a specified table.

GSIs allow users to define indexes that are inconsistent with the distribution of base tables. In this way, single-node plans for querying non-distribution keys of base tables and unique/primary key constraints on non-distribution keys of base tables are achieved.

NOTE

You are advised to create GSIs on a base table as ordinary indexes to improve IUD execution efficiency. In addition, after the GSIs are created on the base table, the IUD performance is affected and deteriorates.

Precautions

- Similar to the base table constraints, the distribution key of GSIs does not support the UPDATE or MERGE INTO operation.
- The GSI can be created only in GTM-lite mode. If a GSI is created in other GTM modes, an error is reported.
- Astore does not support the creation of UB-tree other than GSI and does not support the creation of partitions for GSI.
- The GSI whose distribution is the same as that of the base table cannot be created. Otherwise, an error will be reported during the execution.
- CREATE GSI CONCURRENTLY can be executed only for Ustore tables. If CREATE GSI CONCURRENTLY is executed for Astore tables, a syntax error is reported. CREATE GSI CONCURRENTLY cannot be executed for expression indexes and partial indexes. Otherwise, a syntax error is reported.
- Hash-based GSIs can be created for row-store Astore tables, Ustore tables, partitioned tables, and segment-page tables whose base tables are hash-based. Base tables cannot be replica, level-2 partitioned tables, or list/range distribution tables. GSIs support hash-based distribution only.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- GSIs cannot be created for a base table when any column name, such as **ctid**, **xc_node_hash**, **xmin**, **xmax**, and **tableoid** (for a partitioned table) of the base table is duplicate with an existing column name after it is suffixed with "_new$$" or "_NEW$$."
- If the base table is a partitioned table, the GSI supports a maximum of 27 columns. If the base table is not a partitioned table, the GSI supports a maximum of 28 columns (including the index key and distribution key).
- For GSI creation, GSI rebuild, and operations related to GSI rebuild, for example, partitioning operations (such as DROP, TRUNCATE, MERGE, SPLIT and EXCHANGE PARTITION) on a partitioned table, as well as UPDATE DISTRIBUTED GLOBAL INDEX and ALTER TABLE involving data rebuild, you are advised to enable the STREAM mode to achieve optimal performance. To enable the STREAM mode, set **enable_stream_operator** to **ON** and **create_gsi_opt** to **build**.
- UPSERT is not supported. IUD returning is not supported in the base table where a GSI is created.
- If the VACUUM FULL, CLUSTER, or REINDEX operation on a table is interrupted, the GSI on the table may turn to the **UNUSABLE** state. In this case, an error will be reported when the GSI is queried. You are advised to run **REINDEX INDEX** to rebuild the GSI.
- When executing **COPY** or **GDS** to import data to a base table with a GSI, the **enable_stream_operator** parameter must be enabled to achieve optimal data import performance.
- The following operations will invalidate GSIs: database-level REINDEX, database-level/partition-level CLUSTER, ALTER TABLE PARTITION (DROP, TRUNCATE, MERGE, SPLIT, and EXCHANGE PARTITION). If **UPDATE DISTRIBUTED GLOBAL INDEX** is not specified, all GSIs on the partitioned table will be invalidated. If **UPDATE DISTRIBUTED GLOBAL INDEX** is not specified for EXCHANGE PARTITION, all GSIs on the ordinary table will be invalidated.
- TABLE ACCESS BY INDEX ROWID supports STREAM and non-STREAM distributed plans. The STREAM distributed plans inherit STREAM-related restrictions. Considering the STREAM communication latency, when the selectivity is too low or the number of predicate hit rows is small, the performance is not optimal. In this case, you are advised not to use TABLE ACCESS BY INDEX ROWID. Instead, you are advised to use ordinary indexes. For a non-STREAM distributed plan, GSI table hints are required. This plan is applicable to scenarios where the number of predicate hit rows is small. If the performance of a STREAM distributed plan is not optimal, you can use a non-STREAM distributed plan.
- In the INSERT INTO SELECT scenario, you are advised to enable the **enable_stream_operator** parameter to insert and execute the STREAM plan. (If the base table is a segment-page table or tamper-proof table, the CN plan is still used.) If **enable_stream_operator** is disabled, the execution plan goes back to the CN, and the performance is poor (similar to the performance of creating GSIs in version 503.1.0).
- For INSERT, UPDATE, and DELETE, the execution plan is executed in distributed mode, which causes performance loss. In batch UPDATE/DELETE scenarios, the execution plan goes back to the CN, which causes poor performance.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- GSIs can contain expressions but have the following constraints:
  - Similar to the base table constraints, distribution keys cannot contain expressions. In addition, GSIs whose index column contains only an expression cannot be created either because the distribution key must be an expression. If such GSI is created, a syntax error is reported.
  - Similar to ordinary index constraints, the **CONTAINING** column cannot contain expressions. Otherwise, a syntax error will be reported.
  - If any column name is prefixed with "expr", no GSI containing an expression can be created. Otherwise, a syntax error will be reported.
- When **ENABLE_PBE_OPTIMIZATION** is disabled, the GSI layer of the INSERT, UPDATE, DELETE, and MERGE INTO operations uses a gplan.

Syntax

```
CREATE GLOBAL [ UNIQUE ] INDEX [ [schema_name.]index_name ] [ USING method ] ON table_name

    ({column_name [(length)] | expression }[ COLLATE collation ] [ opclass ][ASC | DESC][ NULLS { FIRST |
LAST } ])
    [ DISTRIBUTE BY HASH ( column_name [, ...] )]
    [ WITH ( {storage_parameter = value} [, ... ] )]
    [ TABLESPACE tablespace_name ]
    [index_option]
    [algorithm_option | lock_option] ...
;

index_option: {
 index_type
  | COMMENT 'string'
}

index_type:
    USING {BTREE | UBTREE}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE};
```

Parameters

- **UNIQUE**

  Creates a unique index. In this way, the system checks whether new values are unique in the index column. If the inserted or updated value causes duplicate records, an error is reported.
- **schema_name**

  Specifies the schema name.

  Value range: an existing schema name.
- **index_name**

  Specifies the name of the index to be created. No schema name can be included here; the index is always created in the same schema as its parent table.

  Value range: a string. It must comply with the **naming convention**.
- **table_name**

  Specifies the name of the table to be indexed (optionally schema-qualified).

  Value range: an existing table name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **USING method**

  Specifies the method of creating an index.

  Value range: UB-tree. The multi-version B-tree index is provided. The index page contains transaction information.

- **column_name**

  Specifies the name of the column on which an index is to be created.

  If the index mode supports composite indexes, multiple columns can be declared. A maximum of 28 columns can be declared for a non-partitioned base table, and a maximum of 27 columns can be declared for a partitioned base table.

- **column_name (length)**

  Specifies the supported prefix key. For details, see **column_name (length)**.

  As an expression index, the prefix index has the same constraints as the expression key in GSI.

- **expression**

  Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax. However, the parentheses can be omitted if the expression has the form of a function call. The expression index can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on upper(col) would allow the WHERE upper(col) = 'JIM' clause to use an index. If an expression contains the IS NULL clause, the index for this expression is invalid. In this case, you are advised to create a partial index.

- **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. You can run the **select * from pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

- **opclass**

  Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column.

- **ASC**

  Specifies an ascending (default) sort order.

- **DESC**

  Specifies a descending sort order.

- **NULLS FIRST**

  Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

- **DISTRIBUTE BY**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the distribution key of the GSI, which is different from that of the base table, and only the hash distribution can be used.

- **DISTRIBUTE BY hash(column_name [,...])**

  Specifies the distribution key of the GSI.

  It needs to be included in the index key.

- **WITH ({storage_parameter = value} [, ...])**

  Specifies the storage parameters of the GSI.

  Value range:

  – FILLFACTOR

    The fill factor of an index is a percentage between 10 and 100.

    Value range: 10–100

  – STORAGE_TYPE

    Specifies the GSI storage type.

    Value range: **USTORE**

  – INDEXSPLIT

    Controls the method of selecting a proper split point for UB-tree, which is the same as that for B-tree.

    Value range: **DEFAULT** or **INSERTPT**

  – enable_tde

    Specifies the index as an encryption index. The database automatically encrypts the data in the encrypted index before storing it. Before using this parameter, ensure that the TDE function has been enabled using the GUC parameter **enable_tde**, the information for accessing the key service has been set using the GUC parameter **tde_key_info**, and the **enable_tde** attribute has been set for the base table. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter supports only B-tree and UB-tree indexes. Other indexes such as hash indexes are not supported.

    Value range: **on** and **off** If **enable_tde** is set to **on**, the values of **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated in the database.

    Default value: **off**

  – encrypt_algo

    Specifies the algorithm for encrypting indexes. This parameter must be used together with **enable_tde**.

    Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

    Default value: If **enable_tde** is set to **on** and the subparameter **table_algorithm** in the GUC parameter **tde_encrypt_config** is not empty, the default value is the same as that of **table_algorithm**. If **table_algorithm** is empty, the default value is **AES_128_CTR**. If **enable_tde** is set to **off**, the default value is null.

  – dek_cipher

    Specifies the DEK ciphertext. After you set the **enable_tde** parameter for an index, the index automatically copies the **dek_cipher** parameter of the base table. This parameter cannot be set or modified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: a string

Default value: empty.

- key_type

  Specifies the master key type. After you set the **enable_tde** parameter for a table, the index automatically copies the **key_type** parameter of the base table. This parameter cannot be set or modified.

  Default value: empty.

- cmk_id

  Specifies the master key ID. After you set the **enable_tde** parameter for a table, the index automatically copies the **cmk_id** parameter of the base table. This parameter cannot be set or modified.

  Value range: a string

  Default value: empty.

- **TABLESPACE tablespace_name**

  Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.

  Value range: an existing table name.

Examples

```
-- Create a base table test1 that contains three columns.
m_db=# CREATE TABLE test1(c1 int, c2 int, c3 int);

-- Create a GSI on the c2 column of the test table based on the hash distribution of the c2 column.
m_db=# CREATE GLOBAL INDEX idx_gsi_1 ON test1(c2) DISTRIBUTE BY HASH(c2);

-- Create a base table test2 that contains three columns.
m_db=# CREATE TABLE test2(c1 int, c2 int, c3 int);

-- Create a GSI on the c2 column of the test2 table based on the hash distribution of the c2 column.
m_db=# CREATE GLOBAL INDEX idx_gsi_2 ON test2(c2) ;

-- Create a base table test3 that contains three columns.
m_db=# CREATE TABLE test3(c1 int, c2 int, c3 int);

-- Create a GSI in UNIQUE form on the c2 column of the test3 table, which is based on the hash distribution of the c2 column by default.
m_db=# CREATE GLOBAL UNIQUE INDEX idx_gsi_3 ON test3(c2) DISTRIBUTE BY HASH(c2);

-- Delete the index.
m_db=# DROP INDEX idx_gsi_1;
m_db=# DROP INDEX idx_gsi_2;
m_db=# DROP INDEX idx_gsi_3;

-- Drop the table.
m_db=# DROP TABLE test1;
m_db=# DROP TABLE test2;
m_db=# DROP TABLE test3;
```

Helpful Links

ALTER INDEX, **CREATE INDEX**, and **DROP INDEX**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.8.12 CREATE GROUP

Description

Creates a user group.

Precautions

CREATE GROUP is an alias for CREATE ROLE, and it is not a standard SQL syntax and not recommended. Users can use CREATE ROLE directly.

Syntax

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] { PASSWORD | IDENTIFIED BY } { 'password'
[ EXPIRED ] | DISABLE };
```

The syntax of the option clause (optional) is as follows:

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameters

See **Parameters** in section "CREATE ROLE."

Examples

```
-- Create a user group.
m_db=# CREATE GROUP super_users WITH PASSWORD "********";

-- Create a user.
m_db=# CREATE ROLE lche WITH PASSWORD "********";

-- Create a user.
m_db=# CREATE ROLE jim WITH PASSWORD "********";

-- Add users to a user group.
m_db=# ALTER GROUP super_users ADD USER lche, jim;

-- Remove users from a user group.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# ALTER GROUP super_users DROP USER jim;

-- Change the name of a user group.
m_db=# ALTER GROUP super_users RENAME TO normal_users;

-- Delete the user.
m_db=# DROP ROLE lche, jim;

-- Delete the user group.
m_db=# DROP GROUP normal_users;
```

Helpful Links

ALTER GROUP, **DROP GROUP**, and **CREATE ROLE**

4.4.2.8.13 CREATE INDEX

Description

Creates an index in a specified table.

Indexes are primarily used to enhance database performance (though inappropriate use can deteriorate database performance). You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **SELECT * FROM t1 JOIN t2 ON t1.a=t2.a AND t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.
- Columns having filter criteria (especially scope criteria) of a WHERE clause
- Columns that appear after ORDER BY, GROUP BY, and DISTINCT

Different from the syntax used for creating an index in an ordinary table, the one used in a partitioned table follows this rule: If the GLOBAL or LOCAL keyword is specified during index creation, the system creates an index of the corresponding type. If no keyword is specified, a global index is created by default with one exception that a local index can be created for a unique index which contains all partition keys.

Precautions

- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Therefore, create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their parameters and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. To use a user-defined function in an index or WHERE clause, mark it as an immutable function.
- Partitioned indexes are classified into local indexes and global indexes. A local index binds to a specific partition, and a global index corresponds to the entire partitioned table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- A user granted the CREATE ANY INDEX permission can create indexes in both the public and user schemas.
- Other users are not allowed to create expression indexes that contain user-defined functions on tables of the initial user.
- If a user-defined function is called in the expression index, the expression index function is executed based on the permission of the function creator.
- Data of the XML type cannot be used as ordinary indexes, unique indexes, global indexes, or local indexes.
- Only B-tree and UB-tree indexes, instead of hash indexes, can be created online. Only ordinary indexes of non-partitioned tables, as well as global and local indexes of partitioned tables can be created. Online index column addition, deletion, and modification, PCR UB-tree indexes, level-2 partitions, and GSIs are not supported. Only ordinary indexes, global indexes, and local indexes of Astore and Ustore can be created online in parallel.

Syntax

- Create an index on a table.

```
CREATE  [ UNIQUE ] INDEX [ [schema_name.] index_name ] [index_type]
ON table_name  ( key_part,... )
    [ WITH ({ storage_parameter = value } [, ... ] ) ]
[ TABLESPACE tablespace_name ]
    [index_option]
    [algorithm_option | lock_option] ...

key_part:
    { { column_name [ ( length ) ] | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
[ NULLS { FIRST | LAST } ] }[, ...]

index_option: {
 index_type
  | COMMENT 'string'
}

index_type:
    USING {BTREE | UBTREE}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE};
```

- Create an index on a partitioned table.

```
CREATE  [ UNIQUE ] INDEX [ [schema_name.] index_name ][index_type]
 ON table_name
    [ LOCAL
        [ ( { PARTITION index_partition_name [ TABLESPACE index_partition_tablespace ]

            [, ...] } ) ]
       | GLOBAL ]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    [index_option]
    [algorithm_option | lock_option] ...

index_option: {
 index_type
  | COMMENT 'string'
}

index_type:
    USING {BTREE | UBTREE}
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE};
```

Parameters

- **UNIQUE**

  Creates a unique index. In this way, the system checks whether new values are unique in the index column. Attempts to insert or update data which would result in duplicate entries will generate an error.

  Currently, only B-tree indexes and UB-tree indexes support unique indexes.

- **schema_name**

  Specifies the schema name.

  Value range: an existing schema name

- **index_name**

  Specifies the name of the index to be created. The schema of the index is the same as that of the table.

  Value range: character strings complying with **Identifier Description**.

- **table_name**

  Specifies the name of the table to be indexed (optionally schema-qualified).

  Value range: an existing table name

- **USING method**

  Specifies the method of creating an index.

  Value range:

  – **BTREE**: The key values of data are stored in a B+ tree structure, enabling users to quickly search for indexes. B-tree indexes support comparison queries with ranges specified.

  – **UBTREE**: Multi-version B-tree index used only for Ustore tables. The index page contains transaction information and can be recycled. By default, the INSERTPT function is enabled for UB-tree indexes.

  Row-store tables (Astore) support the following index types: **BTREE** (default). Row-store tables (Ustore) support the following index type: **BTREE**. (The **UBTREE** index is created.)

  NOTICE

  B-tree is closely related to the table storage type Astore or Ustore. If the specified index type does not correspond to the main table, the index type is automatically converted during index creation. An index created by Astore is B-tree, and the one created by Ustore is UB-tree.

- **column_name**

  Specifies the name of the column on which an index is to be created.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Multiple columns can be specified if the index method supports composite indexes. A global index supports a maximum of 31 columns, and other indexes support a maximum of 32 columns.

- **column_name (length)**

  Creates a prefix key index based on a column in the table. **column_name** indicates the column name of the prefix key, and **length** indicates the prefix length.

  The prefix key uses the prefix of the specified column data as the index key value, which reduces the storage space occupied by the index. Indexes can be used for partial filter and join conditions that contain prefix key columns.

  NOTE

  - The prefix key supports the following index methods: B-tree and UB-tree.
  - The data type of the prefix key column must be binary or character (excluding special characters).
  - The prefix length must be a positive integer that does not exceed 2676 and cannot exceed the maximum length of the column. For the binary type, the prefix length is measured in bytes. For non-binary character types, the prefix length is measured in characters. The actual length of the key value is restricted by the internal page. If a column contains multi-byte characters or an index has multiple keys, the length of the index row may exceed the upper limit. As a result, an error is reported. Consider this situation when setting a long prefix length.
  - The prefix key is a special expression key. Some constraints and restrictions that are not described are the same as those of the expression key. For details, see the description of the expression index.

- **expression**

  Specifies an expression index based on one or more columns of the table. It is usually written in parentheses.

  The expression index can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on upper(col) would allow the clause WHERE upper(col) = 'JIM' to use an index.

  If an expression contains IS NULL, the index for this expression is invalid.

- **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. You can run the **SELECT * FROM pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

- **opclass**

  Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type int4 would use int4_ops; this operator class includes comparison functions for values of type int4. In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, users might want to sort a complex-number data type either by absolute value or by real part. They could do this by defining two operator classes for the data type and then selecting the proper class when making an index. In addition, if the value of **COLLATE** of an index of the string type (such as varchar, varchar2, and text) is not **C** or **POSIX**, but you want the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

index to support prefix matching, you need to specify the **varchar_pattern_ops** option.

- **LOCAL**

  Specifies that the partitioned index to be created is a local index.

- **GLOBAL**

  Specifies the partitioned index to be created as a global index. If no keyword is specified, a global index is created by default.

- **NULLS FIRST**

  Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

- **PARTITION clause**

  The syntax for specifying level-1 partition information is as follows:

  ```
  PARTITION index_partition_name  [ TABLESPACE index_partition_tablespace ]
  ```

  The syntax for specifying level-2 partition information is as follows:

  ```
  SUBPARTITION index_subpartition_name  [ TABLESPACE index_partition_tablespace ]
  ```

  – **PARTITION index_partition_name**

    Specifies the name of an index partition.

    Value range: character strings complying with **Identifier Description**.

  – **TABLESPACE index_partition_tablespace**

    Specifies the tablespace of an index partition.

    Value range: If this parameter is not specified, the value of **index_tablespace** is used.

    Note: You need to create or delete tablespaces in a non-M-compatible database.

    NOTE

    Level-2 partitions are not supported.

- **WITH ({storage_parameter = value} [, ...])**

  Specifies the storage parameter used for an index.

  Value range:

  Only UB-tree indexes support **INDEXSPLIT**. Only non-partitioned B-tree indexes support the **DEDUPLICATION** parameter. Only UB-tree indexes support the **INDEX_TXNTYPE** parameter.

  – STORAGE_TYPE

    Specifies the storage engine type of the table where the index is located. The storage engine type will be automatically corrected if it conflicts with the index type. Once set, this parameter cannot be modified.

    Value range:

    - **USTORE**: The table where the index is located uses an in-place update storage engine.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **ASTORE**: The table where the index is located uses an append-only storage engine.

Default value: **USTORE** for Ustore tables and **ASTORE** for Astore tables.

– FILLFACTOR

Specifies a fill factor for the index. It is a percentage between 10 and 100. When a large number of insertions occur concurrently with a dense key value range, there will be significant contention for the same index page. In such cases, a smaller fill factor is more appropriate.

Value range: 10–100

– **INDEXSPLIT**

Specifies the splitting policy of UB-tree indexes. The **DEFAULT** policy is the same as the splitting policy of B-tree indexes. The **INSERTPT** policy can significantly reduce the index space usage in some scenarios.

Value range: **INSERTPT** and **DEFAULT**

Default value: **INSERTPT**

– **INDEX_TXNTYPE**

Specifies the UB-tree index type (only UB-tree indexes support **INDEX_TXNTYPE**). When the value is **PCR**, flashback queries can be performed using UB-tree. PCR UB-tree indexes do not support online index creation, GSIs, ultimate RTO replay, and read on standby. If **INDEX_TXNTYPE** is not specified, the GUC parameter **index_txntype** specifies the type of the index to be created. **INDEX_TXNTYPE** cannot be modified by using ALTER INDEX INDEX_NAME SET (INDEX_TXNTYPE=PCR or RCR).

Type: character string (case-insensitive)

Value range: **RCR** and **PCR**

Default value: **RCR**

For example:

```
CREATE  INDEX t2_b_pkey ON t(b) WITH(index_txntype='pcr');
```

– **ACTIVE_PAGES**

Specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter.

– **DEDUPLICATION**

Specifies whether to deduplicate and compress tuples with duplicate key values. This is an index parameter. When there are a large number of indexes with duplicate key values, enabling this parameter can effectively reduce the space occupied by indexes. This parameter does not take effect for primary key indexes and unique indexes. If non-unique indexes are used and the index key value duplication rate is low or the index key values are unique, enabling this parameter will slightly deteriorate the index insertion performance. Currently, local and global indexes of partitioned tables are not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: Boolean value. The default value is the value of the **enable_default_index_deduplication** GUC parameter (the default value is **off**).

- **stat_state**

  Determines whether index statistics are locked. If locked, the index statistics cannot be updated.

  Value range: **locked** and **unlock**.

  Default value: **unlock**.

- **LPI_PARALLEL_METHOD**

  Specifies the parallel creation mode of local indexes for partitioned tables. **PAGE** creates indexes in parallel at the page level. Multiple sub-threads are started to scan and sort data. Each sub-thread processes a data page at a time. After scanning and sorting, the main thread combines the sorting results in serial mode and inserts tuples into the indexes. **PARTITION** concurrently creates indexes at the partition level and starts multiple sub-threads. Each sub-thread scans, sorts, and inserts indexes into a partition. **AUTO** estimates the cost of parallel index creation at the page level or partition level based on the partitioned table statistics, and selects the parallel index creation mode with a lower cost. (The possible difference between the statistics and the actual data may result in inaccurate calculation.) Set this parameter to **PARTITION** if data in the partitioned table is evenly distributed. This parameter supports only the B-tree local index of Astore partitioned tables. It does not support the global index of partitioned tables, non-partitioned table indexes, segment-page table indexes, or online index creation.

  Type: character string (case-insensitive)

  Value range: **PAGE**, **PARTITION**, and **AUTO**

  Default value: **PAGE**

  For example:

  ```
  CREATE INDEX idx ON tbl(col) WITH (lpi_parallel_method = 'partition');
  ```

  NOTE

  Currently, the distributed 505.2 environment does not support compression indexes, **COMPRESSTYPE**, **COMPRESS_LEVEL**, **COMPRESS_CHUNK_SIZE**, **COMPRESS_PREALLOC_CHUNKS**, **COMPRESS_BYTE_CONVERT**, and **COMPRESS_DIFF_CONVERT** options.

- **TABLESPACE tablespace_name**

  Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.

  Value range: an existing tablespace name

  Note: You need to create or delete tablespaces in a non-M-compatible database.

Examples

- **Ordinary index**

  ```
  -- Create the tbl_test1 table.
  m_db=# CREATE TABLE tbl_test1(
      id int,          -- User ID
      name varchar(50), -- Username
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    postcode char(6)  -- Postal code
);

-- Create the idx_test1 index for the tbl_test1 table and specify a tablespace.
m_db=# CREATE INDEX idx_test1 ON tbl_test1(name) TABLESPACE pg_default;

-- Query information about the idx_test1 index.
m_db=# SELECT indexname,tablename,tablespace FROM pg_indexes WHERE indexname = 'idx_test1';
 indexname | tablename | tablespace
-----------+-----------+------------
 idx_test1 | tbl_test1 |
(1 row)

-- Drop the index.
m_db=# DROP INDEX idx_test1;
```

- **Unique index**

```
-- Create the unique index idx_test2 for the tbl_test1 table.
m_db=# CREATE UNIQUE INDEX idx_test2 ON tbl_test1(id);

-- Query index information.
m_db=# \d tbl_test1
          Table "public.tbl_test1"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 name     | character varying(50) |
 postcode | character(6)          |
Indexes:
    "idx_test2" UNIQUE, btree (id) TABLESPACE pg_default

-- Drop the index.
m_db=# DROP INDEX idx_test2;
```

- **Partitioned index**

```
-- Create a table.
m_db=# CREATE TABLE student(id int, name varchar(20)) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Create a local partitioned index without specifying the index partition name.
m_db=# CREATE INDEX idx_student1 ON student(id) LOCAL;

-- Check the index partition information. It is found that the number of local index partitions is the
same as that of table partitions.
m_db=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student1'::regclass;
   relname
-------------
 p1_id_idx
 pmax_id_idx
(2 rows)

-- Drop the local partitioned index.
m_db=# DROP INDEX idx_student1;

-- Create a global index.
m_db=# CREATE INDEX idx_student2 ON student(name) GLOBAL;

-- Check the index partition information. It is found that the number of global index partitions is
different from the number of table partitions.
m_db=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student2'::regclass;
 relname
---------
(0 rows)

-- Drop the global partitioned index.
m_db=# DROP INDEX idx_student2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
m_db=# DROP TABLE student;
```

Helpful Links

ALTER INDEX and **DROP INDEX**

Suggestions

- create index

  You are advised to create indexes on:

  - Columns that are often queried
  - Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.
  - Columns having filter criteria (especially scope criteria) of a WHERE clause
  - Columns that appear after ORDER BY, GROUP BY, and DISTINCT

  Constraints:

  - An index of an ordinary table supports a maximum of 32 columns. A global index of a partitioned table supports a maximum of 31 columns.
  - The size of a single index cannot exceed the size of the index page (8 KB). The size of a B-tree or UB-tree index cannot exceed one-third of the page size.
  - When a global index is created on a partitioned table, the following constraints apply:
    - Expression indexes are not supported.
    - Only B-tree indexes (Astore) are supported. UB-tree indexes are supported by Ustore.
  - In the same attribute column, the local index and global index of a partition cannot be specified at the same time.
  - If the ALTER statement does not contain UPDATE GLOBAL INDEX, the original global index is invalid. In this case, other indexes are used for query. If the ALTER statement contains UPDATE GLOBAL INDEX, the original global index is still valid and the index function is correct.

4.4.2.8.14 CREATE NODE

Description

Creates a cluster node.

Precautions

CREATE NODE is an API of the cluster management tool. You are advised not to use this API, because doing so affects the cluster. Only the administrator has the permission to use this API.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
CREATE NODE nodename WITH
 (
   [ TYPE = nodetype,]
   [ HOST = hostname,]
   [ PORT = portnum,]
   [ HOST1 = 'hostname',]
   [ PORT1 = portnum,]
   [ HOSTPRIMARY [ = boolean ],]
   [ PRIMARY [ = boolean ],]
   [ PREFERRED [ = boolean ],]
   [ SCTP_PORT = portnum,]
   [ CONTROL_PORT = portnum,]
   [ SCTP_PORT1 = portnum,]
   [ CONTROL_PORT1 = portnum, ]
   [ NODEIS_CENTRAL [= boolean], ]
   [ NODEIS_ACTIVE [= boolean] ]
 );
```

NOTE

The port whose number is specified by **PORT** is used for internal communications between nodes. Unlike the port connecting to an external client, it can be queried in the pgxc_node table.

Parameters

- **nodename**

  Specifies the node name.

  Value range: character strings complying with **Identifier Description**.

- **TYPE = nodetype**

  Specifies the type of a node.

  Value range:

  – 'coordinator'

  – 'datanode'

- **HOST = hostname**

  Specifies the host name or IP address of a node.

- **PORT = portnum**

  Specifies the port number of the primary node to which a node is bound.

- **HOST1 = hostname**

  Specifies the name or IP address of the standby node corresponding to a node.

- **PORT1 = portnum**

  Specifies the port number of the standby node to which a node is bound.

- **HOSTPRIMARY**

  Specifies whether a host name or IP address is the primary one.

  Value range:

  – **true**

  – **false** (default value)

- **PRIMARY = boolean**

  Specifies whether the node is a primary node or not. A primary node allows read/write operations. A non-primary node allows only read operations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range:

- **true**
- **false** (default value)

- **PREFERRED = boolean**

  Specifies whether the node is a preferred node for read operations.

  Value range:

  - **true**
  - **false** (default value)

- **SCTP_PORT = portnum**

  Specifies the port used by the TCP proxy communications library of the primary node to listen on the data transmission channel. TCP is used to listen on connections.

- **CONTROL_PORT = portnum**

  Specifies the port used by the TCP proxy communications library of the primary node to listen on the control transmission channel. The TCP protocol is used to listen on connections.

- **SCTP_PORT1 = portnum**

  Specifies the port used by the TCP proxy communications library of the standby node to listen on the data transmission channel. TCP is used to listen on connections.

- **CONTROL_PORT 1= portnum**

  Specifies the port used by the TCP proxy communications library of the standby node to listen on the control transmission channel. The TCP protocol is used to listen on connections.

- **NODEIS_CENTRAL**

  Specifies whether the current node is a central control node. It is used only for CNs and is invalid for DNs.

  - **true**
  - **false** (default value)

- **NODEIS_ACTIVE**

  Specifies whether the current node is normal. It is used to mark whether the CN is removed and is invalid for DNs.

  - **true**
  - **false** (default value)

NOTE

In multi-CN environments, when you create a cluster node, it is only generated on the CN that is currently connected and is not synchronized with other CNs. This requires you to repeat the creation process on all CNs. However, to delete a cluster node, you only need to connect to one CN. If a cluster node is created on only one CN, you need to define the current CN name and delete the cluster node on the current CN. For details, see **DROP NODE**.

Examples

```
-- Create cluster nodes and execute them on all CNs.
gaussdb=# CREATE NODE datanode1 WITH(
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  TYPE = datanode,
  PREFERRED = false
);
gaussdb=# CREATE NODE datanode2 WITH(
  TYPE = datanode,
  PREFERRED = false
);

-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 datanode1 | f
 datanode2 | f
(2 rows)

-- Set datanode1 as the preferred DN.
gaussdb=# ALTER NODE datanode1 WITH(preferred = true);
-- Query the cluster DN status after the change.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 datanode1 | t
 datanode2 | f
(2 rows)

-- Connect to only one CN to delete cluster nodes.
gaussdb=# DROP NODE datanode1;
gaussdb=# DROP NODE datanode2;
```

Helpful Links

ALTER NODE and **DROP NODE**

4.4.2.8.15 CREATE NODE GROUP

Description

Creates a cluster node group.

Precautions

- CREATE NODE GROUP is an API of the cluster management tool.
- Only a system administrator has the permission.

Syntax

```
CREATE NODE GROUP groupname
   WITH ( nodename [, ... ] )
   [ BUCKETS [ ( bucketnumber [, ... ] ) ] ];
```

Parameters

- **groupname**

  Specifies the name of a node group.

  Value range: a string complying with **Identifier Description**. A value can contain a maximum of 63 characters.

  NOTE

  A node group name supports all ASCII characters, but you are advised to name a node group according to the naming convention.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **nodename**

  Specifies the node name.

  Value range: a string complying with **Identifier Description**. A value can contain a maximum of 63 characters.

- **BUCKETS [(bucketnumber [, ...])]**

  The current version does not support the BUCKETS option.

Examples

```
-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 dn_6001_6002_6003 | f
 dn_6004_6005_6006 | f
 dn_6007_6008_6009 | f
(3 rows)

-- Create a node group and replace dn_6001_6002_6003 with the actual node name obtained in the previous step.
gaussdb=# CREATE NODE GROUP test_group WITH ( dn_6001_6002_6003 );

-- Query the node group.
gaussdb=# SELECT group_name, group_members FROM pgxc_group;
   group_name   |   group_members
----------------+-------------------
 group_version1 | 16384 16388 16394
 test_group     | 16384
(2 rows)

-- Delete the node group.
gaussdb=# DROP NODE GROUP test_group;
```

Helpful Links

ALTER NODE GROUP and **DROP NODE GROUP**

4.4.2.8.16 CREATE RESOURCE LABEL

Description

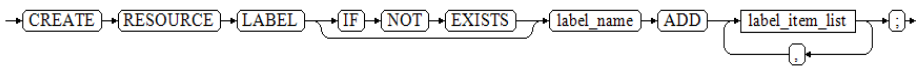
CREATE RESOURCE LABEL is used to create a resource label. The current version does not support this syntax.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

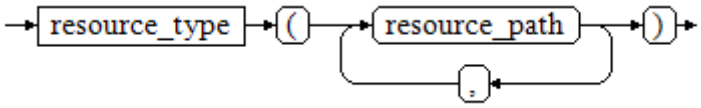
Syntax

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...];
```

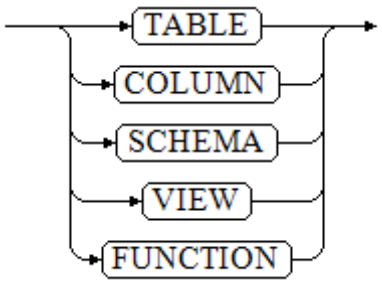


- label_item_list

  ```
  resource_type(resource_path[, ...])
  ```



- resource_type
  { TABLE | COLUMN | SCHEMA | VIEW | FUNCTION }



Parameters

- **IF NOT EXISTS**

  If a resource label with the same name already exists, no error is generated. Instead, a message is displayed, indicating that the resource label already exists.

- **label_name**

  Specifies the resource label name, which must be unique.

  Value range: a string of no more than 63 characters. It must comply with the **naming conventions**. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the resource label name. If a resource label name contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a resource label name that contains uppercase letters, enclose the resource label name with double quotation marks ("").

  NOTE

  The identifier must be lowercase letters, uppercase letters, underscores (_), digits (0–9), or dollar signs ($) and must start with a letter or underscore (_).

- **resource_type**

  Specifies the type of database resources to be labeled.

  Value range: **TABLE**, **COLUMN**, **SCHEMA**, **VIEW**, and **FUNCTION**.

- **resource_path**

  Specifies the path of database resources.

Examples

```
-- Create table tb_for_label.
m_db=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create a resource label based on a table.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create an existing table resource label again and compare the differences between adding IF NOT EXISTS and not adding IF NOT EXISTS.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);
NOTICE:  table_label label already defined, skipping
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CREATE RESOURCE LABEL
m_db=# CREATE RESOURCE LABEL table_label add TABLE(public.tb_for_label);
ERROR:  table_label label already defined

-- Create a resource label based on columns.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

-- Create schema schema_for_label.
m_db=# CREATE SCHEMA schema_for_label;

-- Create a resource label based on a schema.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create view view_for_label.
m_db=# CREATE VIEW view_for_label AS SELECT 1;

-- Create a resource label based on a view.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create function func_for_label.
m_db=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

-- Create a resource label based on a function.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Drop the table resource label table_label.
m_db=# DROP RESOURCE LABEL IF EXISTS table_label;

-- Drop the column resource label column_label.
m_db=# DROP RESOURCE LABEL IF EXISTS column_label;

-- Drop the function resource label func_for_label.
m_db=# DROP FUNCTION func_for_label;

-- Drop the view resource label view_for_label.
m_db=# DROP VIEW view_for_label;

-- Drop the schema resource label schema_for_label.
m_db=# DROP SCHEMA schema_for_label;

-- Drop the tb_for_label table.
m_db=# DROP TABLE tb_for_label;
```

Helpful Links

ALTER RESOURCE LABEL and DROP RESOURCE LABEL

4.4.2.8.17 CREATE ROLE

Description

Creates a role.

A role is an entity that owns database objects and permissions. In different environments, a role can be considered a user, a group, or both.

Precautions

- If a role is added to the database, the role does not have the login permission.
- Only the user who has the CREATE ROLE permission or a system administrator is allowed to create roles.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] { PASSWORD | IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

The syntax of role information configuration clause **option** is as follows:

```
{SYSADMIN | NOSYSADMIN}
  | {MONADMIN | NOMONADMIN}
  | {OPRADMIN | NOOPRADMIN}
  | {POLADMIN | NOPOLADMIN}
  | {AUDITADMIN | NOAUDITADMIN}
  | {CREATEDB | NOCREATEDB}
  | {USEFT | NOUSEFT}
  | {CREATEROLE | NOCREATEROLE}
  | {INHERIT | NOINHERIT}
  | {LOGIN | NOLOGIN}
  | {REPLICATION | NOREPLICATION}
  | {PERSISTENCE | NOPERSISTENCE}
  | CONNECTION LIMIT connlimit
  | VALID BEGIN 'timestamp'
  | VALID UNTIL 'timestamp'
  | USER GROUP 'groupuser'
  | NODE GROUP logic_cluster_name
  | IN ROLE role_name [, ...]
  | IN GROUP role_name [, ...]
  | ROLE role_name [, ...]
  | ADMIN role_name [, ...]
  | USER role_name [, ...]
  | DEFAULT TABLESPACE tablespace_name
  | PROFILE DEFAULT
  | PROFILE profile_name
  | PGUSER
```

Parameters

- **role_name**

  Specifies the name of a role.

  Value range: a string of up to 63 characters in compliance with **Identifier Description**. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the role name. When a role is created, the database will display a message. The role name can be enclosed in back quotes (`). When **sql_mode** is set to **ANSI_QUOTES**, the role name can also be enclosed in double quotation marks ("). Role names that are not enclosed in back quotes (`) or double quotation marks (") will be converted to lowercase letters.

  NOTE

  The identifier must be letters, underscores (_), digits (0-9), or dollar signs ($) and must start with a letter (a-z) or underscore (_).

- **password**

  Specifies the login password.

  A new password must:

  - Contain at least eight characters. This is the default length.
  - Differ from the username or the username spelled backward.
  - Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$ %^&*()-_=+\|[{}];:,<.>/?).
  - The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

are advised not to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and ensure that the password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.

– Be enclosed by single quotation marks when a role is created.

Value range: a character string that cannot be empty.

- **EXPIRED**

  When creating a user, you can specify the **EXPIRED** parameter to create a user whose password is invalid. The user cannot perform simple or extended queries. The statement can be executed only after the password is changed.

- **DISABLE**

  By default, you can change your password unless it is disabled. To disable the password of a user, use this parameter. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, Kerberos authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run **ALTER USER** and specify the password.

- **SYSADMIN | NOSYSADMIN**

  Specifies whether a new role is a system administrator. Roles with the SYSADMIN attribute have the highest permission.

  Value range: If not specified, **NOSYSADMIN** is the default.

  When separation of duties is disabled, users with the SYSADMIN permission can create users with the SYSADMIN, CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

  When separation of duties is enabled, users with the SYSADMIN permission do not have the permission to create users.

- **MONADMIN | NOMONADMIN**

  Specifies whether a role is a monitor administrator.

  Value range: If not specified, **NOMONADMIN** is the default.

- **OPRADMIN | NOOPRADMIN**

  Specifies whether a role is an O&M administrator.

  Value range: If not specified, **NOOPRADMIN** is the default.

- **POLADMIN | NOPOLADMIN**

  Specifies whether a role is a security policy administrator.

  Value range: If not specified, **NOPOLADMIN** is the default.

- **AUDITADMIN | NOAUDITADMIN**

  Specifies whether a role has the audit and management attributes.

  If not specified, **NOAUDITADMIN** is the default.

- **CREATEDB | NOCREATEDB**

  Specifies a role's permission to create databases.

  A new role does not have the permission to create databases.

  Value range: If not specified, **NOCREATEDB** is the default.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **USEFT | NOUSEFT**

  This parameter is reserved and not used in this version.

- **CREATEROLE | NOCREATEROLE**

  Specifies whether a role will be permitted to create new roles (that is, execute **CREATE ROLE** and **CREATE USER**). A role with the CREATEROLE permission can also modify and delete other roles.

  Value range: If not specified, **NOCREATEROLE** is the default.

  When separation of duties is disabled, users with the CREATEROLE permission can create users with the CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

  When separation of duties is enabled, users with the CREATEROLE permission can create users with the CREATEROLE, MONADMIN, POLADMIN, or CREATEDB permission and common users.

- **INHERIT | NOINHERIT**

  Specifies whether a role "inherits" the permissions of roles in the same group. It is not recommended.

- **LOGIN | NOLOGIN**

  Specifies whether a role is allowed to log in to a database. A role having the **LOGIN** attribute can be considered as a user.

  Value range: If not specified, **NOLOGIN** is the default.

- **REPLICATION | NOREPLICATION**

  Specifies whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the **REPLICATION** attribute is specific to replication.

  If not specified, **NOREPLICATION** is the default.

- **PERSISTENCE | NOPERSISTENCE**

  Defines a permanent user. Only the initial user is allowed to create, modify, and delete permanent users with the PERSISTENCE attribute.

- **CONNECTION LIMIT connlimit**

  Specifies how many concurrent connections the role can make.

  NOTICE

  - The system administrator is not restricted by this parameter.
  - connlimit is calculated separately for each primary database node. Number of connections of an M-compatible database = connlimit x Number of normal primary database nodes.

  Value range: an integer in the range [–1, 2^{31} – 1]. The default value **–1** means no limit.

- **VALID BEGIN 'timestamp'**

  Sets the timestamp when a role takes effect. If this clause is omitted, the role has no valid start time. **timestamp** indicates the start time. The format is 'YYYY-MM-DD HH:mm:ss'.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **VALID UNTIL 'timestamp'**

  Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the role has no valid end time. **timestamp** indicates the end time. The format is 'YYYY-MM-DD HH:mm:ss'.

- **USER GROUP 'groupuser'**

  Creates a sub-user. This function is not supported in the current version.

- **IN ROLE role_name**

  Lists one or more existing roles whose permissions will be inherited by a new role. It is not recommended.

- **IN GROUP role_name**

  Specifies an obsolete spelling of IN ROLE. It is not recommended.

- **ROLE role_name**

  Lists one or more existing roles which are automatically added as members of the new role.

- **ADMIN role_name**

  Similar to ROLE. However, ADMIN grants permissions of new roles to other roles.

- **USER role_name**

  Specifies an obsolete spelling of the ROLE clause.

- **DEFAULT TABLESPACE tablespace_name**

  The DEFAULT TABLESPACE clause is ignored.

- **PROFILE profile_name**

  The PROFILE clause is ignored.

- **PGUSER**

  In the current version, this attribute is reserved only for forward compatibility.

Examples

```
-- Create a role manager whose password is ********.
m_db=# CREATE ROLE manager IDENTIFIED BY '********';

-- Create a role with its validity from January 1, 2015 to January 1, 2026.
m_db=# CREATE ROLE miriam WITH LOGIN PASSWORD '********' VALID BEGIN '2015-01-01' VALID UNTIL
'2026-01-01';

-- Change the password of role manager to ********.
m_db=# ALTER ROLE manager IDENTIFIED BY '********' REPLACE '**********';

-- Change role manager to the system administrator.
m_db=# ALTER ROLE manager SYSADMIN;

-- Delete role manager.
m_db=# DROP ROLE manager;

-- Delete role miriam.
m_db=# DROP GROUP miriam;
```

Helpful Links

SET ROLE, ALTER ROLE, DROP ROLE, and GRANT

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.8.18 CREATE SCHEMA

Description

Creates a schema. DATABASE is synonymous with SCHEMA.

Named objects are accessed either by "qualifying" their names with the schema name as a prefix, or by setting a search path that includes the desired schema. When creating named objects, you can also use the schema name as a prefix.

Optionally, CREATE SCHEMA can include sub-commands to create objects within the new schema. The sub-commands are treated essentially the same as separate commands issued after creating the schema. If the AUTHORIZATION clause is used, all the created objects are owned by this user.

Precautions

- Only a user with the **CREATE** permission on the current database can perform this operation.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.
- **templatem** cannot be used as the schema name when creating a schema.

Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] database_name [AUTHORIZATIO role_id]
    [[create_option] [,...]]
create_option: [DEFAULT] {
    CHARACTER SET [=] default_charset
  | CHAR SET [=] default_charset
  | CHARSET [=] default_charset
  | COLLATE [=] default_collation
}
```

Parameters

- **schema_name**

  Specifies the schema name.

  NOTICE

  - The schema name must be unique in the current database.
  - The schema name cannot be the same as the initial username of the current database.
  - The schema name cannot start with **pg_**.
  - The schema name cannot start with **gs_role_**.

  Value range: a string complying with **Identifier Description**.

  NOTE

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **AUTHORIZATION role_id**

  (Optional) Speicfies the owner of the created database. By default, the owner of the database is the current user.

  Value range: an existing username.

- **COLLATE [=] collation_name**

  (Optional) Specifies the character set used by the new database. For example, set this parameter by using **collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the character set of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

- **{CHAR SET | CHARSET | CHARACTER SET} [=] charset_name**

  (Optional) Specifies the character classification used by the new database. For example, set this parameter by using **CHARSET = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used.

  Value range: See **Database-level Character Sets and Collations**.

Examples

```
-- Create and switch to the test database.
m_db=# CREATE SCHEMA test1;
m_db=# USE test1;
-- Create the a table.
m_db=# CREATE TABLE a(id int);
m_db=# DROP TABLE a;
-- Drop the test1 schema.
m_db=# USE public;
m_db=# DROP SCHEMA test1;
```

Helpful Links

ALTER SCHEMA and **DROP SCHEMA**

4.4.2.8.19 CREATE SEQUENCE

Description

Adds a sequence to the current database. The owner of a sequence is the user who creates the sequence.

Precautions

- A sequence is a special table that stores arithmetic progressions. It has no actual meaning and is usually used to generate unique identifiers for rows or tables.
- If a schema name is given, the sequence is created in the specified schema; otherwise, it is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views, or foreign tables in the same schema.
- After the sequence is created, functions nextval() and generate_series(1,N) insert data into the table. Make sure that the number of times for calling

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

nextval is greater than or equal to N+1. Otherwise, errors will be reported because the number of times for calling function generate_series() is N+1.

- A user granted with the CREATE ANY SEQUENCE permission can create sequences in the public and user schemas.

Syntax

```
CREATE SEQUENCE name [ INCREMENT [ BY ] increment ]
   [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
   [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
   [ RESTART [ WITH ] value | RESTART ]
   [ OWNED BY { table_name.column_name | NONE } ];
```

Parameters

- **name**

  Specifies the name of a sequence to be created.

  Value range: a string containing only lowercase letters, uppercase letters, special characters #_$, and digits.

- **increment**

  Optional. Specifies the step for a sequence. A positive number generates an ascending sequence, and a negative number generates a decreasing sequence.

  The default value is **1**.

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  Optional. Specifies the minimum value of the sequence. If **MINVALUE** is not declared, or **NO MINVALUE** is declared, the default value of the ascending sequence is **1**, and that of the descending sequence is **$-2^{63}-1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  Optional. Specifies the maximum value of the sequence. If **MAXVALUE** is not declared, or **NO MAXVALUE** is declared, the default value of the ascending sequence is **$2^{63}-1$**, and that of the descending sequence is **-1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start**

  Optional. Specifies the start value of the sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.

- **cache**

  Optional. Specifies the number of sequences stored in the memory for quick access purposes.

  Default value **1** indicates that one sequence can be generated each time.

  > NOTE
  >
  > It is not recommended that you define **cache** and **maxvalue** or **minvalue** at the same time. The continuity of sequences cannot be ensured after **cache** is defined because unacknowledged sequences may be generated, causing waste of sequences.

- **[NO] CYCLE | NOCYCLE**

  Optional. Recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If **NO CYCLE** is specified, any calling of nextval would return an error after the number of sequences reaches **maxvalue** or **minvalue**.

- **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
- If **CYCLE** is specified, the sequence uniqueness cannot be ensured.

- **OWNED BY**

  Optional. Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to. The associated table and sequence must be owned by the same user and in the same schema. OWNED BY only establishes the association between a table column and the sequence. Sequences on the column do not increase automatically when data is inserted.

  The default value **OWNED BY NONE** indicates that such association does not exist.

NOTICE

You are advised not to use the sequence created using OWNED BY in other tables. If multiple tables need to share a sequence, the sequence must not belong to a specific table.

Examples

Create an ascending sequence named **seq1**. The sequence starts from 101 and the step is 10.

```
m_db=# CREATE SEQUENCE seq1
   START 101
   INCREMENT 10;

-- Select the next number from the sequence.
m_db=# SELECT nextval('seq1');
 nextval
---------
     101
(1 row)
m_db=# SELECT nextval('seq1');
 nextval
---------
     111

-- Delete the sequence.
m_db=# DROP SEQUENCE seq1;
```

Helpful Links

DROP SEQUENCE and **ALTER SEQUENCE**

4.4.2.8.20 CREATE TABLE

Description

Creates an initially empty table in the current database. The table will be owned by the creator.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- If an error occurs during table creation, after it is fixed, the system may fail to delete the empty disk files (whose size is 0) created before the last automatic clearance. This problem seldom occurs and does not affect system running of the database.
- When JDBC is used, the **DEFAULT** value can be set through **PrepareStatement**.
- A user granted the CREATE ANY TABLE permission can create tables in the public and user schemas.
- Prefix keys are not supported in primary key indexes.
- Foreign key constraints are not supported for distributed instances.
- If a distribution key of the character type is used, you are advised not to use case-insensitive collations (such as utf8mb4_general_ci, utf8mb4_unicode_ci, and utf8mb4_0900_ai_ci). Otherwise, data skew may occur and data cannot be evenly distributed. The utf8mb4_bin collation is recommended.

Syntax

Create a table.

```
CREATE [ [ [LOCAL | GLOBAL] TEMPORARY | UNLOGGED]  ] TABLE [ IF NOT EXISTS ] table_name
    ({ column_name data_type [ {CHARACTER SET | CHAR SET | CHARSET} charset_name ]
[ compress_mode ] [ COLLATE collation_name][COMMENT [=] 'string'] [ column_constraint [ ... ] ]
        | table_constraint
        | {INDEX | KEY} [ [schema_name.] index_name ] [ USING method ] ({column_name [(length)] | (expr)
[ASC | DESC]}[,...]) [[COMMENT 'string' | USING method][...]]
        | LIKE source_table [ like_option [...] ] }
        [, ... ])
    [ table_option ][[,...]|[...]]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
    [ TABLESPACE tablespace_name ]
    [ DISTRIBUTE BY
        { REPLICATION
        | HASH ( column_name [, ...] )
        | RANGE ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_less_than_item [, ...] )
        | LIST ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_values_item [, ...] ) }
        }
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
    [ INTERNAL DATA data_content];
```

- Create a table based on an existing source table.

  ```
  CREATE [ [LOCAL | GLOBAL] TEMPORARY ] TABLE [IF NOT EXISTS] table_name LIKE source_table;
  ```

  By default, attributes of the source table are migrated to the target table, including the column definition, index, distribution and partition method, storage definition, and comment. This syntax is also applicable to the adaptation of GaussDB to advanced enhanced capabilities in distributed mode, such as GSI.

- **table_option** is as follows:

  ```
  {
  [DEFAULT] {CHARACTER SET | CHAR SET | CHARSET} [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
    | COMMENT [=] 'string'
    | ENGINE [=] engine_name
    | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  }
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If **COMMENT** is set for multiple times in the same place, only the last value takes effect.
- If **COLLATE** is set for multiple times in the same place, only the last value takes effect.
- When **ENGINE** and **ROW_FORMAT** are used, no error message is displayed. However, the settings do not take effect.
- **WITH (ORIENTATION = column)** is not supported.
- **WITH (HASHBUCKET = ON/OFF)** is not supported.

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  ON UPDATE now_expr |
  UNIQUE [KEY] index_parameters |
  [ PRIMARY ] KEY index_parameters |
 }
```

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

```
{  DICTIONARY  }
```

- **table_constraint** is as follows:

```
[ CONSTRAINT [ constraint_name ] ]
{ CHECK ( expression ) |
UNIQUE [ INDEX | KEY ] [ index_name ][ USING method ] ( { { column_name [ ( length ) ] |
( expression) } [ ASC | DESC ] } [, ... ] ) index_parameters [USING method| COMMENT 'string']|
  PRIMARY KEY [index_name][ USING method ] ( { column_name  }[ ASC | DESC ] } [, ... ] )
index_parameters [USING method| COMMENT 'string']|
  }
```

- **index_parameters** is as follows:

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

- **like_option** is as follows:

```
{ INCLUDING | EXCLUDING }  { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS
   | PARTITION | RELOPTIONS | DISTRIBUTION | UPDATE | ALL }
```

- Range distribution rules

```
For slice_less_than_item:
SLICE slice_name VALUES LESS THAN ({ literal | MAXVALUE } [, ...]) [ DATANODE dn_name |
( dn_name [, ... ] ) ]
For slice_start_end_item:
SLICE slice_name_prefix {
{ START ( literal ) END ( { literal | MAXVALUE } ) } |
{ START ( literal ) } |
{ END ( { literal | MAXVALUE } ) }
}
```

- List distribution rules are as follows:

```
For slice_values_item:
[SLICE slice_name VALUES (list_values_item) [DATANODE dn_name | ( dn_name [, ... ] ) ]
For list_values_item:
{ DEFAULT | { expression[, ...] } }
```

- **now_expr** is as follows:

```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP  | LOCALTIME | NOW() }
```

Parameters

- **UNLOGGED**

  If this keyword is specified, the created table is an unlogged table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - Data written to unlogged tables is not written to WALs. When the OS is restarted, the database is restarted, a switchover occurs, the power supply is cut off, or the system is shut down abnormally, the write operation on the unlogged table is automatically truncated, which may cause data loss. Contents of an unlogged table are also not replicated to standby nodes. Any indexes created on an unlogged table are not automatically logged as well.
  - Unlogged tables have no data on the standby node, and an error is reported when an unlogged table is queried on the standby node.

  Usage scenario: Unlogged tables do not ensure data security. 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 an abnormal shutdown or other unexpected operations, users should rebuild indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify **GLOBAL** or **LOCAL** before **TEMPORARY**. If the GLOBAL keyword is specified, "WARNING: GLOBAL is deprecated in temporary table creation." is reported. In M-compatible mode, a local temporary table is created.

- **TEMPORARY**

  If TEMPORARY is specified, the created table is a temporary table.

  A temporary table is visible only in the current session and is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected CN in the session is normal. 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 advised not to perform operations on temporary tables in DDL statements.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp**. Users should not delete schemas starting with **pg_temp** or **pg_toast_temp**.
- If TEMPORARY is not specified when creating the table and the schema of the table is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables or indexes are being used by other sessions, do not perform ALTER or DROP on the tables or indexes.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, TRUNCATE, REINDEX, and ANALYZE are valid only for the current session.
- You can set the GUC parameter **max_active_global_temporary_table** to determine whether to enable the global temporary table function. If **max_active_global_temporary_table** is set to **0**, the global temporary table function is disabled.
- A temporary table is visible only to the current session. Therefore, it cannot be used together with **\parallel on**.
- Temporary tables do not support primary/standby switchover.
- Global temporary tables do not respond to autovacuum. In persistent connection scenarios, it is advisable to use global temporary tables with the ON COMMIT DELETE ROWS clause or conduct regular vacuum operations manually. Failure to do so may result in Clogs not being recycled.
- Global temporary tables do not support the following scenarios:
  - Global temporary sequences cannot be created. Global temporary tables of each session use shared sequences, which can only ensure uniqueness but not continuity.
  - Global temporary views cannot be created.
  - Partitioned tables cannot be created.
  - Hash bucket tables cannot be created.
  - Extended statistics are not supported.

- **IF NOT EXISTS**

  Sends a notice, but does not throw an error, if a table with the same name exists.

- **table_name**

  Specifies the name of the table to be created.

- **column_name**

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

- **constraint_name**

  Specifies the name of the constraint specified during table creation.

- **index_name**

  Index name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

For a unique key constraint, if both **constraint_name** and **index_name** are specified, the value of **index_name** is used as the index name.

- **USING method**

  Specifies the method of creating an index.

  For details about the value range, see "USING method" in **Parameters**.

NOTICE

Currently, only the B-tree index method is supported. However, B-tree indexes cannot be created for Ustore tables.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

- **expression**

  Specifies an expression index constraint based on one or more columns of the table. It must be written in parentheses.

- **data_type**

  Specifies the data type of a column.

- **compress_mode**

  Specifies whether to compress a table column. The option specifies the compression algorithm preferentially used by table columns. Row-store tables do not support compression. This is unavailable in M-compatible databases.

  Value range: DICTIONARY

- **{CHARACTER SET | CHAR SET | CHARSET} charset_name**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

- **COLLATE collation_name**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **select * from pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result. The utf8mb4_bin, utf8mb4_general_ci, utf8mb4_unicode_ci, binary, gbk_chinese_ci, gbk_bin, gb18030_chinese_ci, and gb18030_bin collations are supported. For details, see **Table-level Character Sets and Collations**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Only the character type supports the specified character set. If the BINARY character set or collation is specified, the character type is converted to the corresponding binary type. If the type mapping does not exist, an error is reported. Currently, only the mapping from the TEXT type to the BLOB type is available.
- Except the BINARY character set and collation, only the character set that is the same as the database encoding can be specified.
- If the character set or collation of a column is not explicitly specified and the default character set or collation of the table is specified, the character set or collation of the column is inherited from the table.

Table 4-24 Supported character sets and collations

Collation	Character Set	Description
utf8mb4_general_ci	utf8mb4 (utf8)	The general collation is used, which is case-insensitive.
utf8mb4_unicode_ci	utf8mb4 (utf8)	The general collation is used, which is case-insensitive.
utf8mb4_bin	utf8mb4 (utf8)	The binary collation is used, which is case-sensitive.
binary	binary	The binary collation is used.
gbk_chinese_ci	gbk	The Chinese collation is used.
gbk_bin	gbk	The binary collation is used, which is case-sensitive.
gb18030_chinese_ci	gb18030	The Chinese collation is used.
gb18030_bin	gb18030	The binary collation is used, which is case-sensitive.
utf8mb4_0900_ai_ci	utf8mb4	The unicode collation algorithm (UCA) rule is used, which is case-insensitive.
utf8_general_ci	utf8	The general collation is used, which is case-insensitive.
utf8_bin	utf8	The binary collation is used, which is case-sensitive.

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

  Specifies a table, known as the source table, for the target table to automatically inherit all column names, data types, constraints, and indexes.

  The target table and the source table are decoupled after creation is complete. This means that any subsequent modifications to the source table will not be propagated to the target table, and data from the target table will not be included in scans of the source table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The copied columns and constraints are not merged based on the same name. If the same name is explicitly specified or is included in another LIKE clause, an error will be reported.
  - The NOT NULL constraints of the source table will be copied to the target table.
  - If **INCLUDING** is specified, the target table will copy the relevant content from the source table. Conversely, specifying **EXCLUDING** will result in the opposite behavior compared to **INCLUDING**.
  - If **INCLUDING DEFAULTS** is specified, the default values in the source table are copied to the target table.
  - If **INCLUDING UPDATE** is specified, the **ON UPDATE now_expr** attribute of the source table column is copied to the target table column.
  - If **INCLUDING CONSTRAINTS** is specified, the CHECK constraints in the source table are copied to the target table.
  - If **INCLUDING INDEXES** is specified, indexes on the source table are also created on the target table.
  - If **INCLUDING STORAGE** is specified, the **STORAGE** setting of the source table column is copied to the target table.
  - If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied.
  - If **INCLUDING PARTITION** is specified, the partition definition in the source table is copied to the target table.
  - If **INCLUDING RELOPTIONS** is specified, the storage parameters (WITH clauses) of the source table are copied to the target table.
  - If **INCLUDING DISTRIBUTION** is specified, the distribution information (including the distribution type and distribution key) in the source table is copied to the target table.
  - If neither **INCLUDING** nor **EXCLUDING** is specified, it is equivalent to specifying **INCLUDING ALL**. **INCLUDING ALL** includes the content of **INCLUDING DEFAULTS**, **INCLUDING UPDATE**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, **INCLUDING RELOPTIONS**, and **INCLUDING DISTRIBUTION**.

    NOTICE

    Normal tables cannot be created from temporary tables.

    If the LIKE, WITH, PARTITION BY, and DISTRIBUTE BY clauses are included during table creation, the created table will use the WITH clause information, partition information, and distribution information specified in the SQL statement rather than those from the source table. If both CHARSET and COMMENT are specified, the relevant information from the source table will be used.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

NOTICE

- A table-level comment can contain a maximum of 2048 characters, and a column-level or index-level comment can contain a maximum of 1024 characters.
- Comments in table_constraint support only primary keys and unique keys.

- **WITH ({ storage_parameter = value } [, ...])**

  Specifies an optional storage parameter for a table or an index. The WITH clause used for tables can also contain **OIDS=FALSE**, indicating that OIDs are not allocated.

  The description of parameters is as follows:

  - **FILLFACTOR**

    The fill factor of a table is a percentage from 10 to 100. If the Ustore is used, the default value is **92**. If the Astore is used, the default value is **100** (completely filled). When a smaller fill factor is specified, **INSERT** operations fill 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 entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

    Value range: 10–100

  - **ORIENTATION**

    Specifies the storage mode of table data. This parameter cannot be modified once it is set. Only row-store mode is supported in M-compatible mode.

    Value range:

    - **ROW** indicates that table data is stored in rows.

      Row store applies to OLTP service and scenarios with a large number of point queries or addition/deletion operations.

    Default value:

    If an ordinary tablespace is specified, the default is **ROW**.

  - **STORAGE_TYPE**

    Specifies the storage engine type. This parameter cannot be modified once it is set.

    Value range:

    - **USTORE** indicates that tables support the in-place update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloating may occur.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **ASTORE** indicates that tables support the append-only storage engine.

Default value:

If **ORIENTATION** and **STORAGE_TYPE** are not specified, Ustore is used by default, indicating that the table supports the in-place update-based storage engine.

– **INIT_TD**

Specifies the number of TDs to be initialized when a Ustore table is created. This parameter can be modified by running the **ALTER TABLE** command. Note that this parameter affects the maximum size of a single tuple stored on the data page. The conversion method is **MAX_TUPLE_SIZE** = **BLCKSZ** – **INIT_TD** x **TD_SIZE**. For example, if you change the number of **INIT_TD** from 4 to 8, the maximum size of a single tuple decreases by 4 x **INIT_TD**.

Value ranges: 2–128. The default value is **4**.

– **COMPRESSION**

Specifies the compression level of 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 mode of table data loaded.

Value range: The row-store table does not support compression. The default value is **NO**.

– **COMPRESSLEVEL**

Specifies the table data compression ratio and duration at the same compression level. This divides a compression level into sublevels, providing more choices for compression ratio and duration. As the value becomes greater, the compression ratio becomes higher and duration longer at the same compression level.

Value range: 0 to 3. The default value is **0**.

– **segment**

The data is stored in segment-page mode. This parameter supports only row-store tables. Protection against unauthorized deletion and damage of physical files 1 to 5 is not supported.

Value range: **on** and **off**

Default value: **off**

– **enable_tde**

Specifies that the table is an encryption table. The database automatically encrypts the data in the encryption table before storing it. Before using this parameter, ensure that TDE function has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter supports only row-store tables, segment-page tables, temporary tables, and unlogged tables.

Value range: **on** and **off** When **enable_tde** is set to **on**, **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated by the database and cannot be manually specified or modified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Default value: **off**

- **parallel_workers**

  Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

  Value range: [0,32], int type. The value **0** indicates that concurrent index creation is disabled.

  Default value: If this parameter is not set, the concurrent index creation function is disabled.

- encrypt_algo

  Specifies the encryption algorithm of the encryption table. This parameter must be used together with **enable_tde**.

  Value range: **AES_128_CTR** and **SM4_CTR** (character string).

  Default value: the value assigned by **table_algorithm** in the **tde_encrypt_config** parameter.

- parallel_workers

  Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

  Value range: [0,32], int type. The value **0** indicates that concurrent index creation is disabled.

  Default value: If this parameter is not set, the concurrent index creation function is disabled.

- dek_cipher

  Specifies the DEK ciphertext. After the **enable_tde** parameter is set for a table, the database automatically generates a data key.

  Value range: character strings.

  Default value: empty.

- key_type

  Specifies the master key type. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key type from the GUC parameter **tde_key_info**.

  Value range: character strings.

  Default value: empty.

- cmk_id

  Specifies the master key ID. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key ID from the GUC parameter **tde_key_info**.

  Value range: character strings.

  Default value: empty.

- **hasuids**

  If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

  Value range: **on** and **off**

  Default value: **off**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **collate**

    Records the default collation of tables. Generally, this parameter is used only for internal storage, import, and export. You are advised not to set or modify this parameter.

    Value range: OIDs in the supported collation

    Default value: **0**

  - **stat_state**

    Determines whether table statistics are locked. If locked, the table statistics cannot be updated.

    Value range: **locked** and **unlock**.

    Default value: **unlock**.

    NOTE

    Currently, the distributed 505.2 environment does not support compression algorithms, **COMPRESSTYPE**, **COMPRESS_LEVEL**, **COMPRESS_CHUNK_SIZE**, **COMPRESS_PREALLOC_CHUNKS**, **COMPRESS_BYTE_CONVERT**, and **COMPRESS_DIFF_CONVERT** options.

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

  ON COMMIT determines what to do when you commit a temporary table creation operation. The options are **PRESERVE ROWS** and **DELETE ROWS**.

  - **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.

- **TABLESPACE tablespace_name**

  Specifies the tablespace where a table is created. If not specified, the default tablespace is used.

  Note: You need to create or delete tablespaces in a non-M-compatible database.

- **CONSTRAINT constraint_name**

  Specifies the name of 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 a particular column and can be applied to more than one column.

- **NOT NULL**

  Specifies that the column value cannot be **NULL**.

- **NULL**

  Specifies that the column value can be **NULL**, which is the default value.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (expression)**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is true or unknown; otherwise, an error is thrown and the database is not altered.

NOTE

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

- **DEFAULT default_expr**
  - The DEFAULT clause is used to specify a default expression for a 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**.
  - The following content can be specified: constants, numeric constants with positive and negative signs, and **update_expr**.
  - The value of **update_expr** can be used as the default value only for columns of the TIMESTAMP or DATETIME type, and the precision of the columns must be the same as that of **update_expr**.
- **ON UPDATE now_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - In terms of syntax, **update_expr** supports four keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, LOCALTIME, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The four types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified on columns of the TIMESTAMP and DATETIME types.
  - The precision specified by this attribute must be the same as that specified by the type in the corresponding column. Otherwise, an error is reported. For example, **CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(6));**. If the precision is inconsistent, **ERROR: Invalid ON UPDATE clause for "col1"** is reported.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.
- **[DEFAULT] {CHARACTER SET | CHAR SET | CHARSET} [=] default_charset**

  Specifies the default character set of a table. If this parameter is specified separately, the default collation of the table is set to the default collation of the specified character set.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **[DEFAULT] COLLATE [=] default_collation**

  Specifies the default collation of a table. If this parameter is specified separately, the default character set of the table is set to the character set corresponding to the specified collation.

  For details about the collation, see **Character Sets and Collations**.

  NOTE

  If the character set or collation of a table is not explicitly specified and the default character set or collation of the schema is specified, the character set or collation of the table is inherited from the schema.

- **Column-level unique constraint: UNIQUE [KEY] index_parameters**

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

  UNIQUE KEY has the same semantics as UNIQUE.

- **Table-level unique constraint: UNIQUE [INDEX | KEY][index_name] [USING method]({{ column_name [(length)] | (expression) } [ASC | DESC] }[, ...]) index_parameters**

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

  For the purpose of a unique constraint, NULL is not considered equal.

  column_name (length) indicates the prefix key. For details, see **•column_name (length)**.

  index_name indicates the index name.

  NOTICE

  For a unique key constraint, if both **constraint_name** and **index_name** are specified, the value of **index_name** is used as the index name.

- **Column-level primary key constraint: [PRIMARY] KEY index_parameters**

  Table-level primary key constraint: PRIMARY KEY [index_name] [USING method] ({ column_name [ASC | DESC] } [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique and not-null values.

  Only one primary key can be specified for a table.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default_tablespace**. If **default_tablespace** is empty, the default tablespace of the database is used.

  NOTE

  This attribute is not supported when the version compatibility control parameter **m_format_dev_version** of an M-compatible database is set to **'s1'** or later.

- **DISTRIBUTE BY**

  Specifies how the table is distributed or replicated between nodes.

  Value range:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **REPLICATION**: Each row in the table exists on all DNs, that is, each DN has complete table data.
- **HASH (column_name)**: Each row of the table will be placed into specified DNs based on the hash value of the specified column.
- **RANGE(column_name)**: maps a specified column based on the range and distributes data to the corresponding DNs.
- **LIST(column_name)**: maps a specified column based on a specific value and distributes data to the corresponding DNs.

  NOTE

  - For HASH distribution, the maximum number of distribution keys is the same as that of columns. A maximum of 1600 distribution keys are supported. For RANGE (VALUE LESS THAN) and LIST distributions, the distribution key supports a maximum of four columns. For RANGE (START END) distribution, the distribution key supports only one column.
  - For a RANGE distribution policy using the VALUE LESS THAN clause, the distribution rules are as follows:
    - The comparison starts from the first column of values to be inserted.
    - If the value of the inserted first column is smaller than the boundary value of the current column in the local slice, the values are directly inserted.
    - If the value of the inserted first column is equal to the boundary value of the current column in the local slice, compare the value of the inserted second column with the boundary value of the next column in the local slice. If the value of the inserted second column is smaller than the boundary value of the next column in the local slice, the values are directly inserted. If they are equal, the comparison of the next columns between the source and target continues.
    - If the value of the inserted first column is greater than the boundary value of the current column in the local slice, compare the value with that in the next slice.
  - If the shard of the RANGE distributed table corresponds to multiple DNs, the hash value of the distribution key is calculated to perform modulo operation on the number of DN and map a new DN. For details, see the example.
  - If the shard of the list distributed table corresponds to multiple DNs, the hash value of the distribution key is calculated for the default shards. Then, use the calculated hash value to perform modulo operation on the number of DN and map a new DN. For non-default shards, the round-robin method is used to map values in the values list to DNs. For details, see the example.
  - The RANGE/LIST distributed tables support only scale-out but not scale-in. For details about the slice scale-out rule, contact the administrator.

For the hash distribution, **column_name** supports the following data types:

- **INTEGER TYPES** includes **TINYINT**, **SMALLINT**, **INTEGER**, **MEDIUMINT**, **BIGINT**, **TINYINT UNSIGNED**, **SMALLINT UNSIGNED**, **INTEGER UNSIGNED**, **MEDIUMINT UNSIGNED**, **BIGINT UNSIGNED**, **NUMERIC[(p[,s])]**, and **DECIMAL[(p[,s])]**.
- **CHARACTER TYPES** include **CHAR[(n)]**, **VARCHAR[(n)]**, **TINYTEXT**, **TEXT**, **MEDIUMTEXT**, and **LONGTEXT**.
- **DATE/TIME TYPES** include **DATE**, **TIME[(p)]**, **TIMESTAMP[(p)]**, **DATETIME[(p)]**, and **YEAR**.

For the RANGE (VALUES LESS THAN) or LIST distribution, **column_name** supports the following data types:

- **INTEGER TYPES** include **TINYINT**, **SMALLINT**, **INT**, **MEDIUMINT**, **BIGINT**, **TINYINT UNSIGNED**, **SMALLINT UNSIGNED**, **INTEGER UNSIGNED**, **MEDIUMINT UNSIGNED**, **BIGINT UNSIGNED**, **NUMERIC[(p[,s])]**, and **DECIMAL[(p[,s])]**.
- **CHARACTER TYPES** include **CHAR[(n)]**, **VARCHAR[(n)]**, **BINARY[(n)]**, **VARBINARY(n)**, **TINYTEXT**, **TEXT**, **MEDIUMTEXT**, and **LONGTEXT**.
- **DATE/TIME TYPES** include **DATE**, **TIMESTAMP[(p)]**, **DATETIME[(p)]**, and **YEAR**.

For the range (start end) distribution, the data type of column_name must be one of the following:

- **INTEGER TYPES** include **TINYINT**, **SMALLINT**, **INT**, **MEDIUMINT**, **BIGINT**, **TINYINT UNSIGNED**, **SMALLINT UNSIGNED**, **INTEGER UNSIGNED**, **MEDIUMINT UNSIGNED**, **BIGINT UNSIGNED**, **NUMERIC[(p[,s])]**, and **DECIMAL[(p[,s])]**.
- **DATE/TIME TYPES** include **DATE**, **TIMESTAMP[(p)]**, **DATETIME[(p)]**, and **YEAR**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

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

- Select appropriate distribution keys.

  A hash table's distribution key should evenly distribute data on each DN to prevent skewing the data or distributing it unevenly across DNs. Determine appropriate distribution keys based on the following principles:

  1. Determine whether data is skewed.

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

     ```
     gaussdb=# 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) on each DN, a data skew occurs. Change the data distribution key based on the following principles:

  2. Rebuild a table to change its distribution keys. ALTER TABLE cannot change distribution keys. Therefore, you need to rebuild a table when changing its distribution keys.

     Principles for selecting distribution keys are as follows:

     The value of the distribution key should be discrete so that data can be evenly distributed on each DN. You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID card number column as the distribution key.

     With the above principles are 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.

- Select appropriate partition keys.

  In range partitioning, a table is partitioned based on ranges defined by one or more 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). Obtain 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 data volume of a single query.

- RANGE/LIST distribution

  If no DN is specified for the shards of a RANGE/LIST distributed table, the database uses the Round Robin algorithm to allocate DNs to the shards. In addition, if RANGE/LIST distribution is used, you are advised to define as many shards as possible when creating a table for future capacity expansion. If the defined number of shards is less than the number of DNs before scale-out, data redistribution cannot be performed on new DNs. Note that the sharding rules are designed by users. In some extreme cases, scale-out may not solve the problem of insufficient storage space.

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

  TO GROUP specifies the node group to which the table to be created belongs. **TO NODE** is used for internal scale-out tools.

- **INTERNAL DATA data_content**

  INTERNAL DATA data_content is only for internal scale-out.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

- Temporary table

```
-- Create a temporary table.
m_db=# CREATE TEMPORARY TABLE test_t1(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);

-- Insert data to the current session.
m_db=# INSERT INTO test_t1 VALUES ('0000009','Jack','Guangzhou','China');

-- Data in the temporary table is valid only in the current transaction. Therefore, the table does not
contain data in another session.
m_db=# SELECT * FROM test_t1;
 id | name | age
----+------+-----
(0 rows)
-- Create a temporary table in a transaction and specify that this table is deleted when the
transaction is committed.
m_db=# START TRANSACTION;
m_db=# CREATE TEMPORARY TABLE test_t2(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
) ON COMMIT DELETE ROWS;
m_db=# INSERT test_t2 VALUES('aid','aname','aprovince','China');
m_db=# COMMIT;
m_db=# SELECT * FROM test_t2;
 id | name | province | country
----+------+----------+---------
(0 rows)
-- Drop the table.
m_db=# DROP TABLE test_t1;
m_db=# DROP TABLE test_t2;
```

- IF NOT EXIST keyword

  If this keyword is used, NOTICE is reported when the table does not exist. If this keyword is not used, ERROR is reported. In either case, the table fails to be created.

```
m_db=# CREATE TABLE test_t3(id INT);
-- Create a table named test_t3.
m_db=# CREATE TABLE test_t3(id INT);
ERROR:  relation "test_t3" already exists in schema "public"
DETAIL:  creating new table with existing name in the same schema

-- Use the IF NOT EXISTS keyword.
m_db=# CREATE TABLE IF NOT EXISTS test_t3(id INT);
NOTICE:  relation "test_t3" already exists, skipping
CREATE TABLE

-- Drop the table.
m_db=# DROP TABLE test_t3;
```

- Create a table using CREATE TABLE ... LIKE.

```
-- Create the source table t1.
m_db=# CREATE TABLE t1(col INT);
CREATE TABLE

m_db=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create the target table t2.
m_db=# CREATE TABLE t2(LIKE t1);
CREATE TABLE

m_db=# \d t2
      Table "public.t2"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |

-- Create the target table t3.
m_db=# CREATE TABLE t3 LIKE t1;
CREATE TABLE

m_db=# \d t3
      Table "public.t3"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |
```

Examples of Creating a Table and Adding Constraints to the Table

- NOT NULL constraints

  If no value is specified for a column with a NOT NULL constraint when data is added, an error is reported. You can add NOT NULL constraints to multiple columns in a table.

  ```
  -- Create a table and add a NOT NULL constraint to the id column.
  m_db=# CREATE TABLE test_t4(
     id       CHAR(7) NOT NULL,
     name     VARCHAR(20),
     province VARCHAR(60),                    -- Province
     country   VARCHAR(30) DEFAULT 'China'        -- Country
  );

  -- If the value of id is not specified or is NULL during data insertion, the NOT NULL constraint is
  triggered. As a result, the insertion fails.
  m_db=# INSERT INTO test_t4 (name,province) VALUES ('scott','Shanghai');
  ERROR:  null value in column "id" violates not-null constraint
  DETAIL:  Failing row contains (null, scott, Shanghai, China)

  -- Drop the table.
  m_db=# DROP TABLE test_t4;
  ```

- Unique constraint

  The keyword UNIQUE is used to add a unique constraint to a column. When data is inserted, the constraint is triggered if the column is duplicate. Multiple **NULL** values are not duplicate values. When a unique constraint is added, a unique index is automatically added. You can add unique constraints to multiple columns in a table.

  ```
  -- Create a table and add a unique constraint.
  m_db=# CREATE TABLE test_t5(
     id       CHAR(7) UNIQUE,
     name     VARCHAR(20),
     province VARCHAR(60),                    -- Province
     country   VARCHAR(30) DEFAULT 'China'        -- Country
  );

  -- You can also use the following method to manually name unique constraints and add constraints
  to multiple columns:
  m_db=# CREATE TABLE test_t6(
     id       CHAR(7),
     name     VARCHAR(20),
     province VARCHAR(60),                     -- Province
     country   VARCHAR(30) DEFAULT 'China',       -- Country
     CONSTRAINT unq_test_id UNIQUE (id,name)
  );
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- When data with duplicate IDs is inserted, constraints are triggered. As a result, the insertion fails.
m_db=# INSERT INTO test_t5(id) VALUES('0000010');
INSERT 0 1
m_db=# INSERT INTO test_t5(id) VALUES('0000010');
ERROR:  duplicate key value violates unique constraint "test_t5_id_key"
DETAIL:  Key (id)=(0000010) already exists.
-- The constraint is not triggered when data whose id is NULL is inserted for multiple times.
m_db=# INSERT INTO test_t5(id) VALUES (NULL);
INSERT 0 1
m_db=# INSERT INTO test_t5(id) VALUES (NULL);
INSERT 0 1

m_db=# SELECT * FROM test_t5;
  id    | name | province | country
---------+------+----------+--------
 0000010 |      |          | China
       |      |          | China
       |      |          | China

-- Drop the table.
m_db=# DROP TABLE test_t5;
m_db=# DROP TABLE test_t6;
```

- Primary key constraints

  The keyword PRIMARY KEY is used to add a primary key constraint to a column. The column must be unique and cannot be null. When a primary key constraint is added, a unique index is automatically created for the table, and a NOT NULL constraint is automatically added for the column.

  Only one primary key constraint can be defined in each table.

```
-- Create a table and add a primary key constraint to the table.
m_db=# CREATE TABLE test_t6(
    id       CHAR(7) PRIMARY KEY,
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);
m_db=# INSERT INTO test_t6 (id,name,province) VALUES ('0000001','july','Beijing');

-- You can also use the following method to manually name unique constraints and add constraints
to multiple columns:
m_db=# CREATE TABLE test_t7(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China',        -- Country
    CONSTRAINT pk_test_t6_id PRIMARY KEY (id,name)
);
-- Insert data whose id is NULL to trigger the constraint.
m_db=# INSERT INTO test_t6 (id,name,province) VALUES (NULL,'july','Beijing');
ERROR:  null value in column "id" violates not-null constraint
DETAIL:  Failing row contains (null, july, Beijing, China).

-- Insert data with duplicate id values to trigger the constraint.
m_db=# INSERT INTO test_t6 (id,name,province) VALUES ('0000001','ben','Shanghai');
ERROR:  duplicate key value violates unique constraint "test_t6_pkey"
DETAIL:  Key (id)=(0000001) already exists.
-- Drop the table.
m_db=# DROP TABLE test_t6;
m_db=# DROP TABLE test_t7;
```

- Check constraints

  The keyword CHECK adds a check constraint to a column. The check constraint must reference one or more columns in the table, and the result returned by the expression must be a Boolean value. Currently, expressions cannot contain subqueries. Both CHECK and NOT NULL constraints can be defined for the same column.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a table and add check constraints.
m_db=# CREATE TABLE test_t8 (
    id       CHAR(7),
    name     VARCHAR(20),
    age      INT CHECK(age > 0 AND age < 150)
);

-- You can also use the following SQL statements to name a check constraint and add the check
constraint for one or more columns:
m_db=# CREATE TABLE test_t9 (
    id       CHAR(7),
    name     VARCHAR(20),
    age      INT,
    CONSTRAINT chek_test_t8_age CHECK(age > 0 AND age < 150)
);
-- If a value that does not comply with the expression is inserted, the check constraint is triggered. As
a result, the insertion fails.
m_db=# INSERT INTO test_t8 (id,name,age) VALUES ('0000007','scott',200);
ERROR:  new row for relation "test_t8" violates check constraint "test_t8_age_check"
DETAIL:  N/A
-- Drop the table.
m_db=# DROP TABLE test_t8;
m_db=# DROP TABLE test_t9;
```

Data Distribution Example

- **REPLICATION**

```
gaussdb=# CREATE TABLE test_replication(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
)DISTRIBUTE BY REPLICATION;

-- Query table information.
gaussdb=# \d+ test_replication
                                  Table "public.test_replication"
  Column  |         Type          |             Modifiers              | Storage  | Stats target | Description
----------+-----------------------+------------------------------------+----------+--------------+-------------
 id       | character(7)          |                                    | extended |              |
 name     | character varying(20) |                                    | extended |              |
 province | character varying(60) |                                    | extended |              |
 country  | character varying(30) | default 'China'::character varying | extended |              |
Has OIDs: no
Distribute By: REPLICATION
Location Nodes: ALL DATANODES
Options: orientation=row, logical_repl_node=-1, compression=no

-- Drop.
gaussdb=# DROP TABLE test_replication;
```

- **HASH**

```
-- Define a hash table.
gaussdb=# CREATE TABLE test_hash(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
)DISTRIBUTE BY HASH(id);

-- Insert data.
gaussdb=# INSERT INTO test_hash VALUES ('0000001', 'Bob',   'Shanghai', 'China'),
                        ('0000002', 'Jack',  'Beijing',  'China'),
                        ('0000003', 'Scott', 'Beijing',  'China');
-- View data distribution.
gaussdb=# SELECT a.count,b.node_name
FROM (SELECT COUNT(*) AS count, xc_node_id FROM test_hash GROUP BY xc_node_id) a,
     pgxc_node b
WHERE a.xc_node_id=b.node_id ORDER BY a.count DESC;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
count |     node_name
-------+-------------------
     2 | dn_6001_6002_6003
     1 | dn_6004_6005_6006
-- Drop the table.
gaussdb=# DROP TABLE test_hash;
```

- **RANGE**

```
-- Define a range-distributed table. (Change the DN name based on the actual situation. You can run the SELECT node_name FROM PGXC_NODE WHERE node_type = 'D' command to query the DN name.)
gaussdb=# CREATE TABLE test_range(
    id       INT,
    name     VARCHAR(20),
    province VARCHAR(60),                       -- Province
    country   VARCHAR(30) DEFAULT 'China'       -- Country
)DISTRIBUTE BY RANGE(id)(
    SLICE s1 VALUES LESS THAN (100) DATANODE dn_6001_6002_6003,
    SLICE s2 VALUES LESS THAN (200) DATANODE dn_6004_6005_6006,
    SLICE s3 VALUES LESS THAN (MAXVALUE) DATANODE dn_6007_6008_6009
);

-- Insert data.
gaussdb=# INSERT INTO test_range VALUES (52,  'Bob',    'Beijing',   'China');
gaussdb=# INSERT INTO test_range VALUES (100, 'Ben',    'Shanghai',  'China');
gaussdb=# INSERT INTO test_range VALUES (150, 'Scott',  'Guangzhou', 'China');
gaussdb=# INSERT INTO test_range VALUES (300, 'Jordan', 'Beijing',   'China');
-- View data distribution.
gaussdb=# SELECT a.count,b.node_name
FROM (SELECT COUNT(*) AS count, xc_node_id FROM test_range GROUP BY xc_node_id) a,
     pgxc_node b
WHERE a.xc_node_id=b.node_id ORDER BY a.count DESC;
 count |     node_name
-------+-------------------
     2 | dn_6004_6005_6006
     1 | dn_6001_6002_6003
     1 | dn_6007_6008_6009
(3 rows)
-- Query data stored on each DN.
gaussdb=# SELECT b.node_name, a.*
FROM (SELECT *, xc_node_id FROM test_range) a,
     pgxc_node b
WHERE a.xc_node_id=b.node_id order by node_name;
     node_name     | id |  name  | province  | country | xc_node_id
-------------------+-----+--------+-----------+---------+-------------
 dn_6001_6002_6003 |  52 | Bob    | Beijing   | China   | -1072999043
 dn_6004_6005_6006 | 100 | Ben    | Shanghai  | China   |  -564789568
 dn_6004_6005_6006 | 150 | Scott  | Guangzhou | China   |  -564789568
 dn_6007_6008_6009 | 300 | Jordan | Beijing   | China   |  1532339558
(4 rows)
-- Drop the table.
gaussdb=# DROP TABLE test_range;
```

- **LIST**

```
-- Define a list-distributed table. (Change the DN name based on the actual situation. You can run the SELECT node_name FROM PGXC_NODE WHERE node_type = 'D' command to query the DN name.)
gaussdb=# CREATE TABLE test_list(
    id       INT,
    name     VARCHAR(20),
    country   VARCHAR(30) DEFAULT 'China'       -- Country
)DISTRIBUTE BY LIST(country)(
    SLICE s1 VALUES ('China') DATANODE dn_6001_6002_6003,
    SLICE s2 VALUES ('USA')   DATANODE dn_6004_6005_6006,
    SLICE s3 VALUES (DEFAULT) DATANODE dn_6007_6008_6009
);

-- Insert data.
gaussdb=# INSERT INTO test_list VALUES (1,'Scott','China');
gaussdb=# INSERT INTO test_list VALUES (2,'Henry','USA');
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
gaussdb=# INSERT INTO test_list VALUES (3,'Michael','France');
gaussdb=# INSERT INTO test_list VALUES (4,'Jack','UK');
-- Query data stored on each DN.
gaussdb=# SELECT b.node_name, a.*
FROM (SELECT *, xc_node_id FROM test_list) a,
    pgxc_node b
WHERE a.xc_node_id=b.node_id order by node_name;
    node_name     | id |  name   | country | xc_node_id
------------------+----+---------+---------+-------------
 dn_6001_6002_6003 |  1 | Scott   | China   | -1072999043
 dn_6004_6005_6006 |  2 | Henry   | USA     |  -564789568
 dn_6007_6008_6009 |  3 | Michael | France  |  1532339558
 dn_6007_6008_6009 |  4 | Jack    | UK      |  1532339558
(4 rows)
-- Drop the table.
gaussdb=# DROP TABLE test_list;
```

Helpful Links

ALTER TABLE and **DROP TABLE**

Suggestions

- UNLOGGED
  - The unlogged table and its indexes do not use the WAL mechanism during data writing. Their write speed is much higher than that of ordinary tables. Therefore, they can be used for storing intermediate result sets of complex queries to improve query performance.
  - The unlogged table has no primary/standby mechanism. In case of system faults or abnormal breakpoints, data loss may occur. Therefore, the unlogged table cannot be used to store basic data.
- TEMPORARY
  - A temporary table is visible only in the current session and is automatically dropped at the end of the current session.
- LIKE
  - The target table automatically inherits all column names, data types, and NOT NULL constraints from this table. The target table is irrelevant to the source table after the creation.
- ORIENTATION ROW
  - Creates a row-store table. Row store applies to the OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using row store can improve the efficiency.
- DISTRIBUTE BY
  - It is recommended that a fact table or dimension table containing a large amount of data be created as a distributed table. Each row of the table will be placed into specified DNs based on the hash value of the specified column. The syntax is DISTRIBUTE BY HASH(column_name).
  - It is recommended that a dimension table containing a small amount of data be created as a replication table. Each row in the table exists on all DNs. That is, each DN has complete table data. The syntax is DISTRIBUTE BY REPLICATION.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.8.21 CREATE TABLE PARTITION

Description

Creates a partitioned table. Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and each physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table.

The common forms of partitioning include range partitioning, hash/key partitioning, list partitioning, and range/list columns partitioning. Currently, row-store tables support range partitioning, hash partitioning, and list partitioning.

Range Partitioning:

- In range partitioning, a table is partitioned based on ranges defined by one or more columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.
- The partitioning policy for range partitioning refers to how data is inserted into partitions. Currently, only the range partitioning policy can be used.
- In range partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

Hash/Key Partitioning:

- In hash partitioning, a modulus and a remainder are specified for each partition based on a column in the table, and records to be inserted into the table are allocated to the corresponding partition, the rows in each partition must meet the following condition: The value of the partition key divided by the specified modulus generates the remainder specified for the partition key.
- In hash partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

List Partitioning:

- List partitioning is to allocate the records to be inserted into a table to the corresponding partition based on the key values in each partition. The key values do not overlap in different partitions. Each group of key values has a dedicated partition for data storage.
- In list partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

Partitioning has the following advantages:

- Certain types of queries experience a significant performance enhancement, particularly when frequently accessed rows are concentrated in a single partition or a few partitions. Partitioning narrows the range of data search and improves data access efficiency.
- When you query or update most of the records in a partition, performance can be improved by taking advantage of continuous scan of that partition instead of partitions scattered across the whole table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Frequent loading or deletion operations on records in a separate partition can be accomplished by reading or deleting that partition. This not only improves performance but also avoids the VACUUM overload caused by bulk DELETE operations (hash partitions cannot be deleted).

Precautions

- When a unique constraint or primary key constraint is added to a distributed table, a local index is created for the constraint if the constraint key contains all partition keys. Otherwise, a global index is created.
- Currently, hash partitioning supports only single-column partition keys, and does not support multi-column partition keys.
- In the PARTITION FOR (values) syntax for partitioned tables, values can only be constants.
- In the PARTITION FOR (values) syntax for partitioned tables, if data type conversion is required for values, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- The maximum number of partitions is 1048575. Generally, it is impossible to create so many partitions, because too many partitions may cause insufficient memory. Create partitions based on the value of **local_syscache_threshold**. The memory used by the partitioned tables is about (number of partitions x 3/1024) MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local_syscache_threshold**. In addition, some space must be reserved for other functions.
- Considering the impact on performance, it is recommended that the maximum number of partitions in a single table be less than or equal to 2,000 and the number of subpartitions multiplied by (Number of local indexes + 1) be less than or equal to 10,000.
- If the memory is insufficient due to too many partitions, the performance deteriorates sharply.
- Currently, the statement specifying a partition cannot perform global index scan.
- The following data types cannot be used as partition keys or level-2 partition keys: BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB, TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT, and BIT.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
 {( [
   { column_name data_type [ {CHARACTER SET | CHAR SET | CHARSET} charset ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ TableLikeOptionList ]
] )
   [ table_option [ [ , ] ... ] ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
   [ DISTRIBUTE BY
      { REPLICATION
      | HASH ( column_name [, ...] )
      | RANGE ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_less_than_item [, ...] )
      | LIST ( column_name [, ...] ) { SLICE REFERENCES tablename | ( slice_values_item [, ...] ) }
      }
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
    PARTITION BY {
      {RANGE [COLUMNS] (partition_key) [ PARTITIONS integer ] ( partition_less_than_item [, ... ] )
[partition_options] } |
      {RANGE [COLUMNS] (partition_key) [ PARTITIONS integer ] ( partition_start_end_item [, ... ] )
[partition_options] } |
      {LIST [COLUMNS] (partition_key)  [ PARTITIONS integer ] ( PARTITION partition_name VALUES [IN]
(list_values)  [TABLESPACE [=] tablespace_name][, ... ] [partition_options] )} |
      {{ HASH | KEY } (partition_key) [ PARTITIONS integer ] ( PARTITION partition_name ] [TABLESPACE
[=] tablespace_name][partition_options] [, ... ])}
    }
    INTERNAL DATA data_content];
;
```

- **table_option** is as follows:

  ```
  [DEFAULT] CHARACTER SET [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
    | COMMENT [=] 'string'
    | ENGINE [=] engine_name
    | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  }
  ```

  NOTE

  - If **COMMENT** is set multiple times in the same place, only the final value will be enforced.
  - If **COLLATE** is set multiple times in the same place, only the final value will be enforced.
  - When **ENGINE** and **ROW_FORMAT** are used, no error message is displayed. However, the settings do not take effect.

- **column_constraint** is as follows:

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr |
    ON UPDATE update_expr |
    COMMENT 'string' |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
   }
  ```

- **table_constraint** is as follows:

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ index_name ][ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] )
  index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters

        }
  ```

- **TableLikeOptionList** is as follows:

  TableLikeOptionList contains **TableLikeIncludingOption** and **TableLikeExcludingOption**.

  ```
  Both of them can be:
     DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS
    | PARTITION | RELOPTIONS | DISTRIBUTION | OIDS | GENERATED   | UPDATE | IDENTITY | ALL
  ```

- **index_parameters**:

  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING method ]
  [ COMMENT 'string']
  ```

- **partition_less_than_item**:

  ```
  PARTITION partition_name VALUES LESS THAN {( { partition_value | MAXVALUE } [,...] ) | MAXVALUE }
  [TABLESPACE [=] tablespace_name]
  ```

- **partition_start_end_item**:

  ```
  PARTITION partition_name {
        {START(partition_value) END ({partition_value | MAXVALUE})} |
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    {START(partition_value)} |
    {END({partition_value | MAXVALUE})}
} [TABLESPACE [=] tablespace_name]
```

- **partition_options**:

```
ENGINE [=] 'string' |
STORAGE ENGINE [=] 'string'
```

  Currently, no error is reported for the syntax, but the syntax does not take effect.

- **update_expr**:

```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
```

Parameters

- **IF NOT EXISTS**

  Sends a notice instead of an error if tables have identical names. The notice prompts that the table relationship already exists.

- **partition_table_name**

  Specifies the name of a partitioned table.

  Value range: character strings complying with **Identifier Description**.

- **column_name**

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

  Value range: character strings complying with **Identifier Description**.

- **data_type**

  Specifies the data type of a column.

- **CONSTRAINT constraint_name**

  Specifies the name of 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 a particular column and can be applied to more than one column. **constraint_name** is optional.

- **index_name**

  Index name.

  NOTICE

  If both **constraint_name** and **index_name** are specified for the UNIQUE KEY constraint, the value of **index_name** will be used as the index name.

- **USING method**

  Specifies the method of creating an index.

  For details about the value range, see "USING method" in **Parameters**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

Currently, only the B-tree index method is supported.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

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

  Specifies a table from which the target table automatically copies all column names, their data types, and their NOT NULL constraints.

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

  - Default expressions for the copied column definitions will be copied only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the target table having default values **NULL**.
  - The target table will always inherit the NOT NULL constraint. The CHECK constraint will be inherited only when **INCLUDING CONSTRAINTS** is specified. No other constraints will be inherited. This rule also applies to table constraints and column constraints.
  - The inherited columns and constraints are not merged based on the same name. If the same name is explicitly specified or is included in another LIKE clause, an error will be reported.
  - If **INCLUDING UPDATE** is specified, columns in the target table will inherit the **ON UPDATE CURRENT_TIMESTAMP** attribute from the source table. By default, this attribute is not inherited.
  - If **INCLUDING INDEXES** is specified, indexes on the source table will also be created on the target table. By default, no indexes are created.
  - If **INCLUDING STORAGE** is specified, columns in the target table will inherit the **STORAGE** attribute from the source table. By default, the **STORAGE** attribute is not inherited.
  - If **INCLUDING COMMENTS** is specified, the target table will inherit the comments on columns, constraints, and indexes from the source table. By default, no comments are inherited.
  - If **INCLUDING RELOPTIONS** is specified, the target table will inherit the storage parameters (WITH clause) from the source table. By default, no storage parameters are inherited.
  - If **INCLUDING DISTRIBUTION** is specified, the target table will inherit the distribution information (including distribution types and keys) from the source table and will no longer use the DISTRIBUTE BY clause. By default, no distribution information is inherited.
  - If **INCLUDING ILM** is specified, the target table will inherit the ILM policy information from the source table.
  - **INCLUDING ALL** includes the content of **INCLUDING DEFAULTS**, **INCLUDING UPDATE**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING RELOPTIONS**, and **INCLUDING DISTRIBUTION**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.
  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

  For details, see **•COMMENT[=]'string'**.
- **{CHARACTER SET | CHAR SET | CHARSET} charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.
- **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. You can run the **select * from pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result. The utf8mb4_bin, utf8mb4_general_ci, utf8mb4_unicode_ci, binary, gbk_chinese_ci, gbk_bin, gb18030_chinese_ci, and gb18030_bin collations are supported. For details, see **Table-level Character Sets and Collations**.

  NOTE

  - Only the character type supports the specified character set. If the BINARY character set or collation is specified, the character type is converted to the corresponding binary type. If the type mapping does not exist, an error is reported. Currently, only the mapping from the TEXT type to the BLOB type is available.
  - Except the BINARY character set and collation, only the character set that is the same as the database encoding can be specified.
  - If the character set or collation of a column is not explicitly specified and the default character set or collation of the table is specified, the character set or collation of the column is inherited from the table.
- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. The description of parameters is as follows:
  - FILLFACTOR

    Specifies a fill factor for the table. It is a percentage from 10 to 100. If the Ustore is used, the default value is **92**. If the Astore is used, the default value is **100** (completely filled). When a smaller fill factor is specified, INSERT operations fill 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 entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

    Value range: 10–100
  - ORIENTATION

    Determines the data storage mode of the table. Currently, an M-compatible database supports only the row-store mode.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range:

ROW (default value): The data will be stored in rows.

NOTICE

orientation cannot be modified.

- STORAGE_TYPE

  Specifies the storage engine type. Once set, this parameter cannot be modified.

  Value range:

  - **USTORE**: The table uses an in-place update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloating may occur.
  - **ASTORE**: The table uses an append-only storage engine.

  Default value:

  If no table is specified, data is stored in in-place update mode by default.

- COMPRESSION

  Row-store tables do not support compression.

- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Temporary tables and unlogged tables are not supported.

  Value range: **on** and **off**

  Default value: **off**

- statistic_granularity

  Records the default PARTITION_MODE when the table analyzes statistics. This parameter is invalid for non-partitioned tables.

  Value range: See the options of PARTITION_MODE.

  Default value: **AUTO**.

Table 4-25 PARTITION_MODE options

PARTITION_MODE Option	Description
ALL	Specifies whether to collect statistics about the entire table and level-1 partition.
GLOBAL	Specifies whether to collect statistics about the entire table.
PARTITION	Specifies whether to collect statistics about the level-1 partition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

PARTITION_MODE Option	Description
GLOBAL AND PARTITION	Specifies whether to collect statistics about the entire table and level-1 partition.
ALL COMPLETE	Specifies whether to collect statistics about the entire table and level-1 partition.
AUTO	Default value.

- **TABLESPACE tablespace_name**

  Specifies that the target table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

  Note: You need to create or delete tablespaces in a non-M-compatible database.

- **DISTRIBUTE BY**

  Specifies how the table is distributed or replicated between nodes.

  For the value range and details, see **•DISTRIBUTE BY**.

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

  TO GROUP specifies the node group to which the table to be created belongs. **TO NODE** is used for internal scale-out tools.

- **PARTITION BY RANGE [COLUMNS] (partition_key)**

  Creates a range partition. **partition_key** is the name of the partition key.

  The semantics of PARTITION BY RANGE COLUMNS is the same as that of PARTITION BY RANGE.

  (1) Assume that the VALUES LESS THAN syntax is used.

  NOTICE

  If the VALUES LESS THAN clause is used, a range partitioning policy supports a partition key with up to 16 columns.

  Data types supported by the partition key are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, INTEGER UNSIGNED, BIGINT UNSIGNED, DATE, YEAR, CHAR[(n)], VARCHAR[(n)], MEDIUMINT, MEDIUMINT UNSIGNED, BINARY[(n)], VARBINARY(n), DATETIME[(p)], and TIME[(p)].

  (2) Assume that the START END syntax is used.

  NOTICE

  In this case, only one partition key is supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data types supported by the partition key are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, INTEGER UNSIGNED, BIGINT UNSIGNED, and DATE.

- **PARTITION partition_name VALUES LESS THAN {({ partition_value | MAXVALUE } [,...])** | MAXVALUE } [TABLESPACE tablespace_name]

  Specifies the information of partitions. **partition_name** is the name of a range partition. **partition_value** is the upper limit of a range partition, and the value depends on the type of **partition_key**. *MAXVALUE* usually specifies the upper limit of the last range partition.

  NOTICE

  - Each partition requires an upper limit.
  - The data type of the upper limit must be the same as that of the partition key.
  - In a partition list, partitions are arranged in ascending order of upper limits. A partition with a smaller upper limit value is placed before another partition with a larger one.
  - MAXVALUE that is not in parentheses can have only one partition key.
  - **partition_value** does not support expressions.

- **PARTITION partition_name {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)}** [TABLESPACE tablespace_name]

  Specifies the information of partitions.

  - **partition_name**: name or name prefix of a range partition. It is the name prefix only in the following cases (assuming that **partition_name** is **p1**):

    If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition, and its name will be **p1_0**. The other partitions are then named **p1_1**, **p1_2**, and the like. For example, if **PARTITION p1 START(1), PARTITION p2 START(2)** is defined, generated partitions are (*MINVALUE*, 1), [1, 2), and [2, *MAXVALUE*), and their names will be **p1_0**, **p1_1**, and **p2**. In this case, **p1** is a name prefix and **p2** is a partition name. **MINVALUE** means the minimum value.
  - **partition_value**: start value or end value of a range partition. The value depends on **partition_key** and cannot be *MAXVALUE*.
  - *MAXVALUE* usually specifies the upper limit of the last range partition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

1. If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition.
2. The START END syntax must comply with the following rules:
   - The value of **START** (if any, same for the following situations) in each **partition_start_end_item** must be smaller than that of **END**.
   - In two adjacent **partition_start_end_item** statements, the value of the first **END** must be equal to that of the second **START**.
   - Each partition includes the start value (unless it is *MINVALUE*) and excludes the end value. The format is as follows: [**START**, **END**).
   - Partitions created by the same **partition_start_end_item** belong to the same tablespace.
   - If **partition_name** is a name prefix of a partition, the length must not exceed 57 bytes. If there are more than 57 bytes, the prefix will be automatically truncated.
   - When creating or modifying a partitioned table, ensure that the total number of partitions in the table does not exceed the maximum value **1048575**.
3. In statements for creating partitioned tables, START END and LESS THAN cannot be used together.
4. The START END syntax in a partitioned table creation SQL statement will be replaced by the VALUES LESS THAN syntax when gs_dump is executed.

- **PARTITION BY LIST [COLUMNS] (partition_key)**

  Create a list partition. **partition_key** is the name of the partition key.

  The semantics of PARTITION BY LIST COLUMNS is the same as that of PARTITION BY LIST.

  – **partition_key** does not support expressions. If no level-2 partition is specified, a list partitioning policy supports a partition key with up to 16 columns. If a level-2 partition is specified, the list partitioning policy supports a partition key with only one column and does not support expressions.

  – For the clause syntax VALUES [IN] (list_values), if **list_values** contains the key values of the corresponding partition, it is recommended that the number of key values of each partition be less than or equal to 64.

  – The semantics of VALUES IN is the same as that of VALUES.

  Data types supported by the partition key are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, INTEGER UNSIGNED, BIGINT UNSIGNED, DATE, YEAR, CHAR[(n)], VARCHAR[(n)], MEDIUMINT, MEDIUMINT UNSIGNED, BINARY[(n)], VARBINARY(n), DATETIME[(p)], and TIME[(p)]. The number of partitions cannot exceed 1048575.

- **PARTITION BY HASH(partition_key)**

  Create a hash partition. **partition_key** is the name of the partition key.

  For **partition_key**, the partition key of the hash partitioning policy supports only one column and does not support expressions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data types supported by the partition key are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, INTEGER UNSIGNED, BIGINT UNSIGNED, DATE, YEAR, CHAR[(n)], VARCHAR[(n)], MEDIUMINT, MEDIUMINT UNSIGNED, BINARY[(n)], VARBINARY(n), DATETIME[(p)], TIME[(p)], TIMESTAMP[(p)], NUMERIC[(p[,s])], FLOAT4[(p, s)], and FLOAT8[(p,s)]. The number of partitions cannot exceed 1048575.

- **PARTITION BY KEY(partition_key)**

  The semantics is the same as that of PARTITION BY HASH(partition_key).

- **PARTITIONS integer**

  Specifies the number of partitions.

  integer indicates the number of partitions. The value must be an integer greater than 0 and cannot be greater than 1048575.

  - When this clause is specified after the range and list partitions, each partition must be explicitly defined, and the number of defined partitions must be equal to the integer value. This clause can be specified after RANGE and LIST partitions.
  - When this clause is specified after the hash and key partitions, if the definition of each partition is not listed, an *integer* number of partitions are automatically generated. The automatically generated partition name is "p+number", and the number ranges from 0 to *integer* minus 1. The tablespace of the partition is the tablespace of the table by default. If each partition definition is explicitly defined, the number of defined partitions must be the same as the value of *integer*. If neither the partition definition nor the number of partitions is specified, a unique partition is created.

- **NOT NULL**

  Specifies that the column value cannot be **NULL**. **ENABLE** can be omitted.

- **NULL**

  Specifies that the column value can be **NULL**, which is the default value.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition)**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is true or unknown; otherwise, an error is thrown and the database is not altered.

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

- **DEFAULT default_expr**

  - The DEFAULT clause is used to specify a default expression for a 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**.
  - The following content can be specified: constants, numeric constants with positive and negative signs, and **update_expr**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The value of **update_expr** can be used as the default value only for columns of the TIMESTAMP or DATETIME type, and the precision of the columns must be the same as that of **update_expr**.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - In terms of syntax, **update_expr** supports four keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, LOCALTIME, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The four types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified on columns of the TIMESTAMP and DATETIME types.
  - The precision specified by this attribute must be the same as that specified by the type in the corresponding column. Otherwise, an error is reported. For example, **CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(6));**. If the precision is inconsistent, **ERROR: Invalid ON UPDATE clause for "col1"** is reported.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.

- **Column-level unique constraint: UNIQUE [KEY] index_parameters**

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

  UNIQUE KEY has the same semantics as UNIQUE.

- **Table-level unique constraint: UNIQUE [INDEX | KEY][index_name] [USING method]({{ column_name [(length)] | (expression) } [ASC | DESC] }[, ...]) index_parameters**

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

  For the purpose of a unique constraint, NULL is not considered equal.

  column_name (length) indicates the prefix key. For details, see **•column_name (length)**.

  index_name indicates the index name.

  NOTICE

  If both **constraint_name** and **index_name** are specified for the UNIQUE KEY constraint, the value of **index_name** will be used as the index name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Column-level primary key constraint: [PRIMARY] KEY index_parameters**

  Table-level primary key constraint: PRIMARY KEY [index_name] [USING method] ({ column_name [ASC | DESC] } [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique and not-null values.

  Only one primary key can be specified for a table.
- INTERNAL DATA data_content

  INTERNAL DATA data_content is only for internal scale-out.

Range Partitioning

- VALUES LESS THAN

```
-- Create a partitioned table test_range1.
m_db=# CREATE TABLE test_range1(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION p2 VALUES LESS THAN (400),
    PARTITION p3 VALUES LESS THAN (600),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Insert 1000 data records.
m_db=# INSERT INTO test_range1 VALUES(GENERATE_SERIES(1,1000),'abcd');

-- View the number of rows (199) in the p1 partition. The value range is [1, 200).
m_db=# SELECT COUNT(*) FROM test_range1 PARTITION (p1);
 count
-------
   199
(1 row)

-- View the number of rows (200) in the p2 partition. The value range is [200, 400).
m_db=# SELECT COUNT(*) FROM test_range1 PARTITION (p2);
 count
-------
   200
(1 row)

-- View the partition information.
m_db=# SELECT a.relname, a.boundaries
FROM pg_partition a
WHERE a.parentid = 'test_range1'::regclass and a.parttype = 'p';
 relname | boundaries
---------+------------
 pmax    | {NULL}
 p3      | {600}
 p2      | {400}
 p1      | {200}
(4 rows)

-- Delete.
m_db=# DROP TABLE test_range1;
```

- START END

```
-- Create a partitioned table.
m_db=# CREATE TABLE test_range2(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 START(1) END(600),
    PARTITION p2 START(600) END(800),
    PARTITION pmax START(800) END(MAXVALUE)
);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- View the partition information.
m_db=# SELECT relname, boundaries FROM pg_partition WHERE parentid = 'test_range2'::regclass
AND parttype = 'p' ORDER BY 1;
relname | boundaries
---------+------------
 p1_0    | {1}
 p1_1    | {600}
 p2      | {800}
 pmax    | {NULL}
(4 rows)

-- Drop.
m_db=# DROP TABLE test_range2;
```

List Partitioning

```
-- Create a list partitioned table.
m_db=# CREATE TABLE test_list ( NAME VARCHAR ( 50 ), area VARCHAR ( 50 ) )
PARTITION BY LIST (area) (
    PARTITION p1 VALUES ('Beijing'),
    PARTITION p2 VALUES ('Shanghai'),
    PARTITION p3 VALUES ('Guangzhou'),
    PARTITION p4 VALUES ('Shenzhen'),
    PARTITION pdefault VALUES (DEFAULT)
);

-- Insert data.
m_db=# INSERT INTO test_list VALUES ('bob', 'Shanghai'),('scott', 'Sichuan');

-- Query partition data.
m_db=# SELECT * FROM test_list PARTITION (p2);
 name |   area
------+----------
 bob  | Shanghai
(1 row)
m_db=# SELECT * FROM test_list PARTITION (pdefault);
 name  |  area
-------+---------
 scott | Sichuan
(1 row)

-- Drop.
m_db=# DROP TABLE test_list;
```

Hash Partitioning

```
-- Create a hash partitioned table and specify the number of partitions.
m_db=# CREATE TABLE test_hash1(c1 int) PARTITION BY HASH(c1) PARTITIONS 3;

-- Create a hash partitioned table and specify the partition name.
m_db=# CREATE TABLE test_hash2(c1 int) PARTITION BY HASH(c1)(
    PARTITION pa,
    PARTITION pb,
    PARTITION pc
);

-- View the partition information.
m_db=# SELECT b.relname AS table_name,
       a.relname AS partition_name
FROM pg_partition a,
     pg_class b
WHERE b.relname LIKE 'test_hash%'
  AND a.parttype = 'p'
  AND a.parentid = b.oid;
 table_name | partition_name
------------+----------------
 test_hash1 | p2
 test_hash1 | p1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 test_hash1 | p0
 test_hash2 | pc
 test_hash2 | pb
 test_hash2 | pa
(6 rows)

-- Drop.
m_db=# DROP TABLE test_hash1,test_hash2;
```

Helpful Links

ALTER TABLE PARTITION and **DROP TABLE**

4.4.2.8.22 CREATE USER

Description

Creates a user.

Precautions

- A user created using the CREATE USER statement has the LOGIN permission by default.
- When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

```
CREATE USER user_name [ [ WITH ] option [ ... ] ]{ PASSWORD | IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```

The option clause is used to configure information, including permissions and properties.

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | USER GROUP 'groupuser'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  | PROFILE profile_name
  | PGUSER
```

Parameters

- **user_name**

  Specifies the username.

  Value range: a string of up to 63 characters in compliance with **Identifier Description**. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the username. When a user is created, the database will display a message. The username can be enclosed in backquote (`). When **sql_mode** is set to **ANSI_QUOTES**, the role name can also be enclosed in double quotation marks ("). Usernames that are not enclosed in backquote or double quotation marks will be converted to lowercase letters.

  NOTE

  The identifier must be letters, underscores (_), digits (0-9), or dollar signs ($) and must start with a letter (a-z) or underscore (_).

- **password**

  Specifies the login password.

  The new password must:

  - Contain at least eight characters. This is the default length.
  - Differ from the username or the username spelled backward.
  - Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$ %^&*()-_=+\|[{}];:,<.>/?).
  - The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are advised not to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and ensure that the password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.
  - Be enclosed by single quotation marks when a user is created.

  Value range: a string

For other parameters, see **CREATE ROLE**.

Examples

```
-- Create user jim whose login password is ********.
m_db=# CREATE USER jim PASSWORD '********';

-- Alternatively, you can run the following statement:
m_db=# CREATE USER kim IDENTIFIED BY '********';

-- To create a user with the CREATEDB permission, add the CREATEDB keyword.
m_db=# CREATE USER dim CREATEDB PASSWORD '********';

-- Change the login password of user jim from ******** to **********.
m_db=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

-- Add the CREATEROLE permission to jim.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# ALTER USER jim CREATEROLE;

-- Lock jim.
m_db=# ALTER USER jim ACCOUNT LOCK;

-- Delete the user.
m_db=# DROP USER kim CASCADE;
m_db=# DROP USER jim CASCADE;
m_db=# DROP USER dim CASCADE;
```

Helpful Links

ALTER USER, **CREATE ROLE**, and **DROP USER**

4.4.2.8.23 CREATE VIEW

Description

Creates a view. A view is a virtual table, not a base table. The database stores only the view definition, not the view data itself. Instead, the view data is stored in the base table. If data in the base table changes, the data queried from the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database.

- You can define frequently used data as views to encapsulate complex query SQL statements. This simplifies operations.
- You can query only data defined in a view, which is secure. Columns in a base table are hidden to protect the data structure of the database.
- User permission management is simplified. Users are granted only the permission to use views.

Precautions

Users with the CREATE ANY TABLE permission can create views in the public and user schemas.

Syntax

```
CREATE [ OR REPLACE ] [ TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

You can use WITH(security_barrier) to create a relatively secure view. This prevents attackers from printing hidden base table data by using the RAISE statement of low-cost functions.

After a view is created, you are not allowed to use REPLACE to modify column names in the view or delete the columns.

Calculation operations (such as the function calls and operator calculation) are not allowed in a query for fields of the following data types:

- BINARY[(n)]
- BOOLEAN/BOOL
- VARBINARY(n)
- CHAR[(n)]
- VARCHAR(n)
- TIME[(p)]
- DATETIME[(p)]
- TIMESTAMP[(p)]
- BIT[(n)]
- NUMERIC[(p[,s])]
- DECIMAL[(p[,s])]
- DEC[(p[,s])]
- FIXED[(p[,s])]
- FLOAT4[(p, s)]
- FLOAT8[(p,s)]
- FLOAT[(p)]
- REAL[(p, s)]
- FLOAT[(p, s)]
- DOUBLE[(p,s)]
- DOUBLE PRECISION[(p,s)]
- TEXT
- TINYTEXT
- MEDIUMTEXT
- LONGTEXT
- BLOB
- TINYBLOB
- MEDIUMBLOB
- LONGBLOB

The concepts involved in the constraints on updating views and subqueries, and inserting and deleting views are described as follows:

- Join view: view created with multiple tables using JOIN.
- Key-preserved table: Insertion, update, and deletion of multi-table join views are restricted by key-preserved tables. In a multi-table view, if each row in the source table corresponds to each row in the view, and a row in the source table does not correspond to multiple rows in the view after the JOIN, the source table is a key-preserved table.
- Relationship between the top layer and the bottom layer: A view may have multiple nested layers. For example, a view consists of one or more views or subqueries. The view that is directly operated by the DML is called the top layer. The tables and views that form the view and the tables and views in the WITH clause are called the corresponding bottom layer.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Updatable columns: user columns that are directly referenced from the base table. System columns and whole-row references are not considered updatable.
- Updatable views: views that support insert, update, and delete operations. To be updatable, views must not contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause, set operations (UNION and EXCEPT), aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series).

Parameters

- **OR REPLACE**

  (Optional) Redefines the view if it already exists.

- **TEMPORARY**

  (Optional) Creates a temporary view. The view is automatically deleted when the current session ends. If any table referenced by a view is a temporary table, the view is created as a temporary view (regardless of whether **TEMPORARY** is specified in the SQL statement).

- **view_name**

  Specifies the name of the view to be created. It is optionally schema-qualified.

  Value range: character strings complying with **Identifier Description**.

- **column_name**

  Specifies a list of names to be used as the column names of the view. If not specified, the column names will be automatically derived from the query.

  Value range: character strings complying with **Identifier Description**.

- **view_option_name [= view_option_value]**

  Specifies an optional parameter for a view.

  Currently, **view_option_name** supports only the **security_barrier** parameter. This parameter is used when the view attempts to provide row-level security.

  Value range: **true** and **false** (Boolean).

- **query**

  Specifies a SELECT or VALUES statement that will provide the columns and rows of the view.

NOTICE

If **query** contains a clause specifying the partition of a partitioned table, the OID of the specified partition is fixed to the system catalog when the view is created. If a DDL operation that will change the OID of the specified partition is used, for example, DROP, SPLIT, or MERGE, the view is unavailable. In this case, you need to create a view.

Example

- Common view:

```
-- Create the test_tb1 table and insert 100 data records into the table.
m_db=# CREATE TABLE test_tb1(col1 int, col2 int);
m_db=# INSERT INTO test_tb1 VALUES (generate_series(1,100),generate_series(1,100));
-- Create a view whose value of col1 is less than 5.
m_db=# CREATE VIEW test_v1 AS SELECT * FROM test_tb1 WHERE col1 < 3;
-- Query a view.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT * FROM test_v1;
 col1 | col2
------+------
    1 |    1
    2 |    2
(2 rows)

-- Delete the table and the view.
m_db=# DROP VIEW test_v1;
m_db=# DROP TABLE test_tb1;
```

- Temporary view:

```
-- Create a table and a temporary view.
m_db=# CREATE TABLE test_tb2(c1 int, c2 int);
m_db=# CREATE TEMPORARY VIEW test_v2 AS SELECT * FROM test_tb2;
-- Query the table and view information. (The temporary table belongs to a schema starting with
pg_temp instead of public.)
m_db=# \d
                                        List of relations
         Schema          |   Name   | Type  | Owner |                      Storage
-------------------------+----------+-------+-------+-----------------------------------------------------
 pg_temp_dn_6001_1_1_9473 | test_v2  | view  | omm   |
 public                   | test_tb2 | table | omm   |
{orientation=row,compression=no,storage_type=USTORE}
(2 rows)

-- Exit the current session and log in again. Check whether the temporary view is deleted.
m_db=# \d
                                  List of relations
 Schema |   Name   | Type  | Owner |                      Storage
--------+----------+-------+-------+-----------------------------------------------------
 public | test_tb2 | table | omm   | {orientation=row,compression=no,storage_type=USTORE}
(1 row)

-- Delete the view and the table.
m_db=# DROP VIEW test_v2 ;
m_db=# DROP TABLE test_tb2 CASCADE;
```

Helpful Links

ALTER VIEW and **DROP VIEW**

4.4.2.9 D

4.4.2.9.1 DEALLOCATE

Description

Deallocates a previously prepared statement.

Precautions

If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

Syntax

```
{DEALLOCATE | DROP}  PREPARE name;
```

Parameters

name

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the name of the prepared statement to be deallocated.

Examples

```
-- Create three prepared statements q1, q2, and q3, delete q1, and then delete all prepared statements.
m_db=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name | statement | parameter_types
------+-----------+-----------------
(0 rows)

m_db=# PREPARE q1 from  'SELECT 1 AS a';
PREPARE
m_db=# PREPARE q2 from 'SELECT 1 AS a';
PREPARE
m_db=# PREPARE q3 from 'SELECT 1 AS a';
PREPARE
m_db=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name |          statement           | parameter_types
------+------------------------------+-----------------
 q1   | PREPARE q1 AS SELECT 1 AS a; | {}
 q3   | PREPARE q3 AS SELECT 1 AS a; | {}
 q2   | PREPARE q2 AS SELECT 1 AS a; | {}
(3 rows)

m_db=# DEALLOCATE PREPARE q1;
DEALLOCATE
m_db=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name |          statement           | parameter_types
------+------------------------------+-----------------
 q3   | PREPARE q3 AS SELECT 1 AS a; | {}
 q2   | PREPARE q2 AS SELECT 1 AS a; | {}
(2 rows)
```

4.4.2.9.2 DELETE

Description

Deletes rows that satisfy the WHERE clause from the specified table. If the WHERE clause is absent, the effect is to delete all rows in the table. The result is a valid, but an empty table.

Precautions

- The table owner, users with the DELETE permission on the current table, and users with the DELETE ANY TABLE permission can delete data from the table. System administrators automatically have this permission when separation of duties is disabled. Additionally, you must have the SELECT permission on the tables referenced by the USING clause and the tables read by **condition**.
- For a DELETE statement whose subquery is a STREAM plan, UPDATE cannot be performed on the deleted row data.

Syntax

```
DELETE [/*+ plan_hint */] [IGNORE] FROM {table_name [partition_clause] | view_name}
   [ WHERE condition ]
   [ ORDER BY { expression  [ ASC | DESC ] } [ NULLS { FIRST | LAST } ] ]
   [ LIMIT { count } ]
   [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **plan_hint**

  Follows the DELETE keyword in the **/*+ */** format. It is used to optimize the plan of a DELETE statement block. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **IGNORE**

  When the DELETE statement uses the IGNORE keyword, some ERROR-level errors can be degraded to WARNING-level errors, and invalid values can be adjusted to the closest values based on error scenarios. GaussDB supports the following error downgrade scenarios:

  sql_mode set to loose

- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: existing table names.

- **partition_clause**

  Deletes a specified partition.

  PARTITION { (partition_name | subpartition_name [, ...])

  For keywords, see **SELECT**.

- **view_name**

  Specifies name of the target view.

  NOTE

  The restrictions on deleting a view are as follows:

  - The DELETE operation can be performed only on columns that directly reference user columns in the base table.
  - A view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
  - Views that contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause at the top layer are not supported.
  - Views that contain set operations (UNION and EXCEPT) at the top layer are not supported.
  - Views whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
  - Table types supported in views include ordinary tables, temporary tables, global temporary tables, partitioned tables, Ustore tables, and Astore tables.
  - A join view can delete only the key-preserved tables in the view. If there is only one key-preserved table, the data in the table is deleted. If there are multiple key-preserved tables, only the data in the first key-preserved table following **FROM** is deleted. For details about the key-preserved table, see **CREATE VIEW**.
  - The DELETE operation cannot be performed on the system view.

- **NULLS FIRST**

  Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **condition**

  Specifies an expression that returns a Boolean value. Only rows for which this expression returns **true** will be deleted. To prevent unexpected outcomes, avoid using numeric types such as int for **condition**, because these types can be implicitly converted into Boolean values: **0** becomes **false**, while other values become **true**.

- **ORDER BY**

  For keywords, see **SELECT**.

- **LIMIT**

  For keywords, see **SELECT**.

- **RETURNING**

  Returns the inserted rows. The syntax of the RETURNING list is identical to that of the output list of SELECT.

Example

- **Delete some data.**

```
-- Create a table.
m_db=# CREATE TABLE test_t1(col1 INT,col2 INT);
m_db=# INSERT INTO test_t1 VALUES (1, 1), (2, 2), (3, 3), (4, 4), (4, 6);

-- Delete all records from a table.
m_db=# DELETE FROM test_t1 WHERE col1 = 4;

-- Query.
m_db=# SELECT * FROM test_t1;
 col1 | col2
------+------
    1 |    1
    2 |    2
    3 |    3
(3 rows)
```

- **Delete all data.**

```
-- Delete all data.
m_db=# DELETE FROM test_t1;

-- Query.
m_db=# SELECT * FROM test_t1;
 col1 | col2
------+------
(0 rows)

-- Drop the table.
m_db=# DROP TABLE test_t1;
```

- **Delete a view.**

```
-- Create a schema.
m_db=# CREATE SCHEMA del_view;
CREATE SCHEMA
m_db=# SET CURRENT_SCHEMA = 'del_view';
SET

-- Create a table and insert data into the table.
m_db=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
m_db=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
m_db=# CREATE TABLE tdata (x INT PRIMARY KEY, y INT);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tdata_pkey" for table "tdata"
CREATE TABLE
m_db=# CREATE TABLE tinfo (z INT PRIMARY KEY, comm VARCHAR(20));
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tinfo_pkey" for table "tinfo"
CREATE TABLE
m_db=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
m_db=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
m_db=# INSERT INTO tdata VALUES (1, 1), (2, 2), (3, 3);
INSERT 0 3
m_db=# INSERT INTO tinfo VALUES (1,'one'), (2, 'two'), (3, 'three'), (5, 'wrong three');
INSERT 0 4

-- Create a single table view.
m_db=# CREATE VIEW v_del1 AS SELECT * FROM t1;
CREATE VIEW

-- Delete data from t1 using a view.
m_db=# DELETE FROM v_del1 where y1 = 3;
DELETE 1

-- Create a multi-table join view.
m_db=# CREATE VIEW vvt1t2 AS SELECT * FROM t1, t2 WHERE x1 = x2;
CREATE VIEW
m_db=# CREATE VIEW vv_dup AS SELECT td1.x x1, td1.y y1, td2.x x2, td2.y y2 FROM tdata td1, tdata
td2, tinfo WHERE td2.y=tinfo.z AND td1.x=td2.y;
CREATE VIEW

-- Delete the multi-table join view.
m_db=# SELECT * FROM vvt1t2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
  5 |  5 |  5 |  5
(3 rows)

m_db=# DELETE FROM vvt1t2 WHERE y2 = 5;
DELETE 1
m_db=# SELECT * FROM vvt1t2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
(2 rows)

m_db=# SELECT * FROM vv_dup;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
  3 |  3 |  3 |  3
(3 rows)

m_db=# DELETE FROM vv_dup WHERE y1 = 2;
DELETE 1
m_db=# SELECT * FROM vv_dup;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  3 |  3 |  3 |  3
(2 rows)

-- Drop the schema.
m_db=# DROP SCHEMA del_view;
NOTICE:  drop cascades to 7 other objects
DETAIL:  drop cascades to table t1
drop cascades to table t2
drop cascades to table tdata
drop cascades to table tinfo
drop cascades to view v_del1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
drop cascades to view vvt1t2
drop cascades to view vv_dup
DROP SCHEMA
```

Suggestions

- delete

  To delete all records in a table, use the TRUNCATE syntax.

4.4.2.9.3 DO

Description

This syntax is for internal tools only.

Executes an anonymous code block.

A code block is regarded as a function body without parameters. The return value type is void. It is parsed and executed a single time.

Precautions

- Currently, only PL/pgSQL can be installed.
- To use an untrusted language, you must have the USAGE permission on the programming language or the system administrator permission.

Syntax

```
DO code;
```

Parameters

- **code**

  Specifies the programming language code that can be executed. The value must be a character string.

Examples

```
-- Create the webuser user.
m_db=# CREATE USER webuser PASSWORD '********';

-- Grant all permissions on all views in the tpcds schema to the webuser user.
m_db=# DO $$DECLARE r record;
BEGIN
   FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class c,pg_namespace n
         WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
   LOOP
      EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
   END LOOP;
END$$;

-- Delete the webuser user.
m_db=# DROP USER webuser CASCADE;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.9.4 DROP AUDIT POLICY

Description

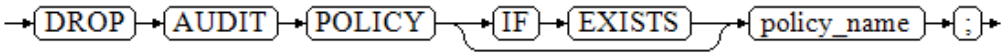
Deletes an audit policy. Currently, the distributed mode is not supported.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```



Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the **naming conventions**.

- **IF EXISTS**

  Checks whether the audit policy exists. If it exists, the deletion is successful. Otherwise, a NOTICE message is sent.

Examples

```
-- Create the adt1 policy.
m_db=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;
CREATE AUDIT POLICY

-- Delete the audit policy adt1.
m_db=# DROP AUDIT POLICY adt1;
DROP AUDIT POLICY

-- When you delete the audit policy adt0 that does not exist, the system displays a message indicating that the deletion fails because the audit policy does not exist.
m_db=# DROP AUDIT POLICY adt0;
ERROR:  adt0 policy does not exist, drop failed
```

Helpful Links

CREATE AUDIT POLICY and **ALTER AUDIT POLICY**

4.4.2.9.5 DROP DATABASE

Description

In M-compatible databases, DATABASE and SCHEMA are synonyms. Deleting a database is deleting a schema.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- Only the database owner or a user granted with the DROP permission can run the **DROP DATABASE** command. The system administrator has this permission by default.
- Users except the initial user and O&M administrator cannot drop the database of the O&M administrator.

NOTICE

- DROP DATABASE cannot be undone.
- Schemas beginning with **pg_temp** or **pg_toast_temp** are for internal use. Do not delete them. Otherwise, unexpected consequences may be incurred.

Syntax

```
DROP DATABASE [ IF EXISTS ] database_name;
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified database or schema does not exist.
- **database_name**

  Specifies the name of the database or schema to be deleted.

  Value range: an existing database name

Examples

See **Examples** in section "CREATE DATABASE."

Helpful Links

ALTER DATABASE and **CREATE DATABASE**

4.4.2.9.6 DROP EXTENSION

Description

Currently, this syntax is not supported. Deletes an extension. This feature is for internal use only. You are advised not to use it.

Precautions

- The **DROP EXTENSION** command deletes an extension from the database. When you delete an extension, the components that make up the extension are also deleted.
- The **DROP EXTENSION** command can be used only by the owner of the extension.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  If the **IF EXISTS** parameter is used and the extension does not exist, no error is reported. Instead, a notice is generated.

- **name**

  Name of an installed extension.

- **CASCADE**

  Automatically deletes objects that depend on the extension.

- **RESTRICT**

  If any object depends on an extension, the extension cannot be deleted (unless all its member objects and other extension objects are deleted at a time using the **DROP** command). This is a default processing.

Examples

Delete the PL/pgSQL extension from the current database. The PL/pgSQL extension is created by the database by default.

```
m_db=# DROP EXTENSION IF EXISTS plpgsql;
```

In the current database, if an object that uses PL/pgSQL exists, this command will fail. For example, a column in any table is of the PL/pgSQL type. Adding the **CASCADE** option can forcibly delete the extension and objects that depend on it.

Helpful Links

ALTER EXTENSION and **CREATE EXTENSION**

4.4.2.9.7 DROP FUNCTION

Description

Deletes a function.

NOTE

This function is for internal use only and not available to users.

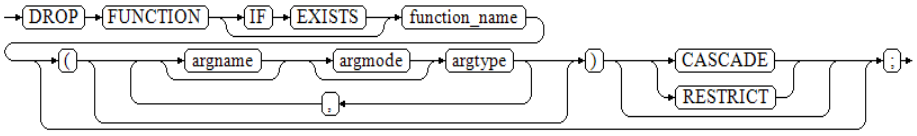
Precautions

- If a function involves operations on temporary tables, DROP FUNCTION cannot be used.
- Only the function owner or a user granted with the DROP permission can run the **DROP FUNCTION** command. The system administrator has this permission by default.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```



Parameters

- **IF EXISTS**

  Specifies that a notice instead of an error is reported if the specified function does not exist.

- **function_name**

  Specifies the name of the function to be deleted.

  Value range: an existing function name

- **argmode**

  Specifies the parameter mode of the function.

- **argname**

  Specifies the parameter name of the function.

- **argtype**

  Specifies the parameter type of the function.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the function.
  - **RESTRICT**: refuses to delete the function if any objects depend on it. This is the default action.

Examples

- Delete the list of parameters that can be omitted when a function.

```
-- Create a function.
m_db=# CREATE OR REPLACE FUNCTION func_add_sql(integer, integer) RETURNS bigint
AS 'select $1 + $2;'
LANGUAGE SQL
IMMUTABLE
SHIPPABLE
STRICT
AUTHID DEFINER
not fenced
COST 10000;
/

-- Delete the function.
m_db=# DROP FUNCTION func_add_sql;
```

- Delete a function with the same name.

  If a function with the same name exists, add a parameter list when deleting the function. Otherwise, an error is reported.

```
-- Create a function.
m_db=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
BEGIN
    RETURN i + 1;
END;
$$ LANGUAGE plpgsql;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- func_add is an overloaded function.
m_db=# CREATE OR REPLACE FUNCTION func_add(int) RETURNS int AS $$
BEGIN
    RETURN $1+10;
END;
$$ LANGUAGE PLPGSQL;

-- Delete the function.
m_db=# DROP FUNCTION func_add(int);
m_db=# DROP FUNCTION func_add(int,int);
```

Helpful Links

CREATE FUNCTION

4.4.2.9.8 DROP GROUP

Description

Deletes a user group.

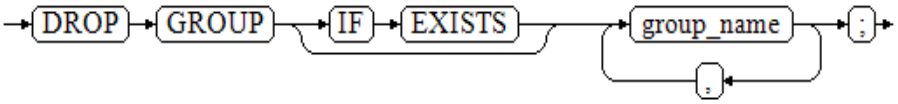
DROP GROUP is an alias for DROP ROLE.

Precautions

DROP GROUP is an API of the M-compatible database management tool and is used to manage M-compatible databases. You are advised not to use this API, because doing so affects the M-compatible databases.

Syntax

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```



Parameters

- **IF EXISTS**

  If a role does not exist, no error is reported. Instead, a notification is sent, indicating that the role does not exist.

- **group_name**

  Specifies the name of the role to be deleted.

  Value range: an existing role name

Examples

See **Examples** in section "CREATE GROUP."

Helpful Links

ALTER GROUP, **CREATE GROUP**, and **DROP ROLE**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.9.9 DROP INDEX

Description

Drops an index.

Precautions

- Only the index owner, owner of the schema where the index resides, a user who has the INDEX permission on the table where the index resides, or a user who has the DROP ANY INDEX permission can run the **DROP INDEX** command. When separation of duties is disabled, a system administrator has this permission by default.
- For a global temporary table, if a session has initialized a global temporary table object (including creating a global temporary table and inserting data into the global temporary table for the first time), other sessions cannot delete indexes from the table.

Syntax

```
DROP INDEX  [ IF EXISTS ]
   index_name [, ...] [ CASCADE | RESTRICT];

DROP INDEX index_name [, ...] [CASCADE | RESTRICT] ON table_name [algorithm_option | lock_option];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified index does not exist.
- **index_name**

  Specifies the index name to be deleted.

  Value range: an existing index
- **CASCADE | RESTRICT**
  - **CASCADE**: Dropping the index automatically removes dependent objects.
  - **RESTRICT**: The database refuses to drop the index if any objects depend on it. This is the default value.
- **table_name**

  Name of the table.
- **algorithm_option**

  For details, see **CREATE INDEX**.
- **lock_option**

  For details, see **CREATE INDEX**.

Examples

```
-- Create a table.
m_db=# CREATE TABLE test1_index (id INT, name VARCHAR(20));
m_db=# CREATE INDEX idx_test1 on test1_index (id);

-- Delete an index.
m_db=# DROP INDEX IF EXISTS idx_test1 CASCADE;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Delete a table.
m_db=# DROP TABLE test1_index;
```

Helpful Links

ALTER INDEX and **CREATE INDEX**

4.4.2.9.10 DROP NODE

Description

Deletes nodes from a cluster.

Precautions

DROP NODE is an API of the cluster management tool. You are advised not to use this API, because doing so affects the cluster. Only an administrator has the permission to use this API.

Syntax

```
DROP NODE [ IF EXISTS ] nodename [WITH ( cnnodename [, ... ] )];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified node does not exist.

- **nodename**

  Specifies the name of the node to be deleted.

  Value range: an existing node name

- **cnnodename**

  Specifies the CN name. If this parameter is defined, the cluster nodes under the currently connected CN and the defined CN are deleted.

  Value range: an existing CN name

NOTE

If there are multiple CNs, cluster nodes are created only on the currently connected CNs and will not be synchronized to other CNs. You need to create cluster nodes on all CNs. To delete a cluster node, you only need to connect to one CN. If a cluster node is created on only one CN, you need to define the current CN name and delete the cluster node on the current CN.

Examples

```
-- Create cluster nodes on all CNs.
gaussdb=# CREATE NODE datanode1 WITH(
  TYPE = datanode,
  PREFERRED = false
);

-- Connect to only one CN to delete cluster nodes.
gaussdb=# DROP NODE datanode1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Helpful Links

CREATE NODE and **ALTER NODE**

4.4.2.9.11 DROP NODE GROUP

Description

Deletes a node group.

Precautions

- DROP NODE GROUP is an API of the cluster management tool.
- Only the system administrator or a user who has the DROP permission can perform this operation.

Syntax

```
DROP NODE GROUP groupname;
```

Parameters

groupname

Specifies the name of the node group to be deleted.

Value range: an existing node group name

Examples

```
-- Query the initial cluster DN status.
gaussdb=# SELECT node_name, nodeis_preferred FROM pgxc_node WHERE node_type = 'D' ORDER BY 1;
 node_name | nodeis_preferred
-----------+------------------
 dn_6001_6002_6003 | f
 dn_6004_6005_6006 | f
 dn_6007_6008_6009 | f
(3 rows)

-- Create a node group and replace dn_6001_6002_6003 with the actual node name obtained in the previous step.
gaussdb=# CREATE NODE GROUP test_group WITH ( dn_6001_6002_6003 );

-- Delete the node group.
gaussdb=# DROP NODE GROUP test_group;
```

Helpful Links

ALTER NODE GROUP and **CREATE NODE GROUP**

4.4.2.9.12 DROP OWNED

Description

Deletes the database objects owned by a database role.

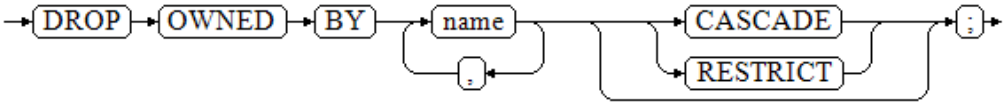
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- The role's permissions on all the database objects in the current database and shared objects (databases and tablespaces) will be revoked.
- DROP OWNED is often used to prepare for removing one or more roles. Because DROP OWNED affects only the objects in the current database, you need to run this statement in each database that contains the objects owned by the role to be removed.
- Using the **CASCADE** option may cause this statement to recursively remove objects owned by other users.
- The databases and tablespaces owned by the role will not be removed.

Syntax

```
DROP OWNED BY name [, ...] [ CASCADE [CONSTRAINTS] | RESTRICT ];
```



Parameters

- **name**

  Specifies the role name.

- **CASCADE [CONSTRAINTS] | RESTRICT**
  - **CASCADE [CONSTRAINTS]**: cascadingly deletes the objects that depend on the objects to be deleted. **CONSTRAINTS** can be omitted.
  - **RESTRICT**: refuses to delete the object if any objects depend on it. This is the default action.

Examples

```
-- Create user jim.
m_db=# CREATE USER jim PASSWORD '********';

-- Revoke user jim's permissions on all objects in the current database and on shared objects (databases and tablespaces).
m_db=# DROP OWNED BY jim;

-- Delete user jim.
m_db=# DROP USER jim;
```

Helpful Links

DROP ROLE

4.4.2.9.13 DROP PREPARE

Description

Deallocates a previously prepared statement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

Syntax

```
{DEALLOCATE | DROP}  PREPARE name;
```

Examples

See **Examples** in section "DEALLOCATE."

Helpful Links

PREPARE

4.4.2.9.14 DROP RESOURCE LABEL

Description

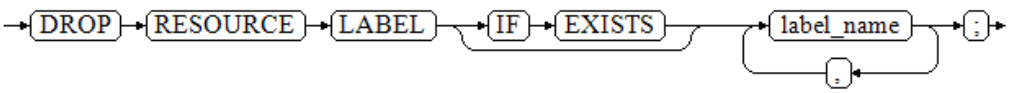
DROP RESOURCE LABEL is used to delete resource labels. The current version does not support this syntax.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP RESOURCE LABEL [IF EXISTS] label_name[, ...];
```



Parameters

- **label_name**

  Specifies the resource label name.

  Value range: a string. It must comply with the **naming conventions**.

Example

```
-- Create table tb_for_label.
m_db=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create a resource label based on a table.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create an existing table resource label again and compare the differences between adding IF NOT EXISTS and not adding IF NOT EXISTS.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);
NOTICE:  table_label label already defined, skipping
CREATE RESOURCE LABEL
m_db=# CREATE RESOURCE LABEL table_label add TABLE(public.tb_for_label);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
ERROR:  table_label label already defined

-- Create a resource label based on columns.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

-- Create schema schema_for_label.
m_db=# CREATE SCHEMA schema_for_label;

-- Create a resource label based on a schema.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create view view_for_label.
m_db=# CREATE VIEW view_for_label AS SELECT 1;

-- Create a resource label based on a view.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create function func_for_label.
m_db=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
LANGUAGE SQL;

-- Create a resource label based on a function.
m_db=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Drop the table resource label table_label.
m_db=# DROP RESOURCE LABEL IF EXISTS table_label;

-- Drop the column resource label column_label.
m_db=# DROP RESOURCE LABEL IF EXISTS column_label;

-- Drop the function resource label func_for_label.
m_db=# DROP FUNCTION func_for_label;

-- Drop the view resource label view_for_label.
m_db=# DROP VIEW view_for_label;

-- Drop the schema resource label schema_for_label.
m_db=# DROP SCHEMA schema_for_label;

-- Drop the tb_for_label table.
m_db=# DROP TABLE tb_for_label;
```

Helpful Links

ALTER RESOURCE LABEL and **CREATE RESOURCE LABEL**

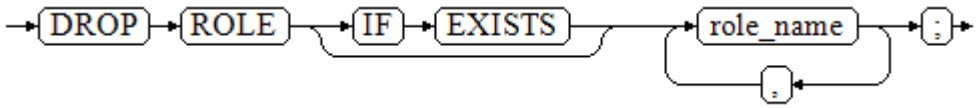
4.4.2.9.15 DROP ROLE

Description

Deletes a role.

Syntax

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```



 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified role does not exist.
- **role_name**

  Specifies the name of the role to be deleted.

  Value range: an existing role name. For details, see **•role_name**.

Examples

See **Examples** in section "CREATE ROLE."

Helpful Links

CREATE ROLE, **ALTER ROLE**, and **SET ROLE**

4.4.2.9.16 DROP SCHEMA

Description

Deletes a schema from the current database.

Precautions

- Only the owner of a schema or a user granted the DROP permission for a schema has the permission to run the **DROP SCHEMA** command. If separation of duties is disabled, the system administrator has this permission by default.
- Users except the initial user and O&M administrator cannot drop the schema of the O&M administrator.

Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified schema does not exist.
- **schema_name**

  Specifies the schema name.

  Value range: an existing schema name

NOTICE

Schemas beginning with **pg_temp** or **pg_toast_temp** are for internal use. Do not delete them. Otherwise, unexpected consequences may be incurred.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

See **Examples** in section "CREATE SCHEMA."

Helpful Links

ALTER SCHEMA and **CREATE SCHEMA**

4.4.2.9.17 DROP SEQUENCE

Description

Deletes a sequence from the current database.

Precautions

Only the owner of a sequence, the owner of the schema to which the sequence belongs, or a user granted the DROP permission on a sequence or a user granted the DROP ANY SEQUENCE permission can delete a sequence. If separation of duties is disabled, a system administrator has this permission by default.

Syntax

```
DROP SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified sequence does not exist.
- **sequence_name**

  Specifies the name of the sequence to be deleted.
- **CASCADE**

  Automatically deletes the objects that depend on the sequence.
- **RESTRICT**

  Refuses to delete the sequence if any objects depend on it. This is the default action.

Examples

```
-- Create an ascending sequence named serial, starting from 101.
m_db=# CREATE SEQUENCE serial START 101;

-- Delete a sequence.
m_db=# DROP SEQUENCE serial;
```

Helpful Links

ALTER SEQUENCE and **CREATE SEQUENCE**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.9.18 DROP TABLE

Description

Deletes a table.

Precautions

- **DROP TABLE** forcibly deletes a specified table. After a table is deleted, the indexes that depend on the table will be deleted. Deleting a partitioned table also deletes all partitions in the table.
- The owner of a table, the owner of the schema of the table, users granted with the DROP permission on the table, or users granted with the DROP ANY TABLE permission can delete the specified table. When separation of duties is disabled, the system administrator has the permission to delete the specified table by default.

Syntax

```
DROP [TEMPOARARY] TABLE [ IF EXISTS ]
   { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] [ PURGE ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified table does not exist.
- **schema**

  Schema name.
- **table_name**

  Table name.
- **CASCADE | RESTRICT**
  - **CASCADE**: allows cascade deletion of the objects (such as views) that depend on the table.
  - **RESTRICT**: refuses to delete the table if any objects depend on it. This is the default.

    NOTE

    This attribute is supported only as syntax but does not take effect when the version of an M-compatible database is s1 or later (for example, the version compatibility control parameter **m_format_dev_version** is set to **'s1'**).
- **PURGE**

  Specifies that even if the recycle bin function is enabled, the table is physically dropped instead of being moved to the recycle bin.

Examples

```
-- Create the test table.
m_db=# CREATE TABLE test(c1 int, c2 int);
-- Drop the test table.
m_db=# DROP TABLE IF EXISTS test;

-- Failed to delete the table when there are views dependent on the table.
m_db=# create schema s1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# create table s1.t1(a int);
m_db=# create view s1.v1 as select * from s1.t1;
m_db=# create view s1.v2 as select * from s1.v1;
m_db=# drop table s1.t1;
ERROR:  cannot drop table s1.t1 because other objects depend on it
DETAIL:  view s1.v1 depends on table s1.t1
view s1.v2 depends on view s1.v1
HINT:  Please drop the dependent first.
m_db=# drop table s1.t1 cascade;
ERROR:  cannot drop table s1.t1 because other objects depend on it
DETAIL:  view s1.v1 depends on table s1.t1
view s1.v2 depends on view s1.v1
HINT:  Please drop the dependent first.
```

Helpful Links

ALTER TABLE and **CREATE TABLE**

4.4.2.9.19 DROP USER

Description

Deletes a user and the schema with the same name as the user.

Precautions

- **CASCADE** is used to delete the objects (excluding databases) that depend on the user. It cannot delete locked objects unless the objects are unlocked or the threads locking the objects are terminated.
- In M-compatible databases, the **gaussdb.conf** configuration file contains the **enable_kill_query** parameter. This parameter affects **CASCADE**.
  - If **enable_kill_query** is **on** and **CASCADE** is used, the statement automatically kills the threads locking dependent objects and then deletes the specified user.
  - If **enable_kill_query** is **off** and **CASCADE** is used, the statement waits until the threads locking dependent objects end and then deletes the specified user.
- If the dependent objects are other databases or reside in other databases, manually delete them before deleting the user from the current database. **DROP USER** cannot delete objects across databases.
- Before deleting a user, you need to delete all the objects owned by the user and revoke the user's permissions on other objects. Alternatively, you can specify **CASCADE** to delete the objects owned by the user and the granted permissions.

Syntax

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE [CONSTRAINTS] | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified user does not exist.
- **user_name**

  Specifies the name of the user to be deleted.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: an existing username. For details about the username requirements, see **•user_name**.

- **CASCADE [CONSTRAINTS]| RESTRICT**
  - **CASCADE [CONSTRAINTS]**: cascadingly deletes objects that depend on the user and revokes the permissions granted to the user. **CONSTRAINTS** can be omitted.
  - **RESTRICT**: refuses to delete the user if any objects depend on it. This is the default action.

    NOTE

    In M-compatible databases, the **gaussdb.conf** configuration file contains the **enable_kill_query** parameter. This parameter affects **CASCADE**.

    - If **enable_kill_query** is **on** and **CASCADE** is used, the statement automatically kills the threads locking dependent objects and then deletes the specified user.
    - If **enable_kill_query** is **off** and **CASCADE** is used, the statement waits until the threads locking dependent objects end and then deletes the specified user.

Examples

See **Examples** in section "CREATE USER."

Helpful Links

ALTER USER and **CREATE USER**

4.4.2.9.20 DROP VIEW

Description

Forcibly deletes a view from the database.

Precautions

The owner of a view, owner of the schema of the view, users granted with the DROP permission on the view, or users granted with the DROP ANY TABLE permission can run the **DROP VIEW** command. By default, the system administrator has the permission to run the command when separation of duties is disabled.

Syntax

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified view does not exist.

- **view_name**

  Specifies the name of the view to be deleted.

  Value range: an existing view name

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **CASCADE | RESTRICT**
  - **CASCADE**: allows cascade deletion of the objects (such as views) that depend on the table.
  - **RESTRICT**: refuses to delete the view if any objects depend on it. This is the default action.

    NOTE

    This attribute is supported only as syntax but does not take effect when the version of an M-compatible database is s1 or later (for example, the version compatibility control parameter **m_format_dev_version** is set to **'s1'**).

Examples

See **Example** in section "CREATE VIEW."

Helpful Links

ALTER VIEW and **CREATE VIEW**

4.4.2.10 E

4.4.2.10.1 EXECUTE

Description

Executes a prepared statement. Because a prepared statement exists only in the lifetime of the session, the prepared statement must be created earlier in the current session by using the PREPARE statement.

Syntax

```
EXECUTE name;
```

Parameters

name

Specifies the name of the prepared statement to be executed.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create the reason table.
m_db=# CREATE TABLE tpcds.reason (
   CD_DEMO_SK          INTEGER          NOT NULL,
   CD_GENDER           character(16)           ,
   CD_MARITAL_STATUS   character(100)
);

-- Insert data.
m_db=# INSERT INTO tpcds.reason VALUES(51, 'AAAAAAAADDAAAAAA', 'reason 51');

-- Create the reason_t1 table.
m_db=# CREATE TABLE tpcds.reason_t1 LIKE tpcds.reason;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create a prepared statement for an INSERT statement and execute the prepared statement.
m_db=# PREPARE insert_reason FROM 'INSERT INTO tpcds.reason_t1
VALUES(52,''AAAAAAAADDAAAAAA'',''reason 52'')';

m_db=# EXECUTE insert_reason;

-- Delete the reason and reason_t1 tables.
m_db=# DROP TABLE tpcds.reason;
m_db=# DROP TABLE tpcds.reason_t1;

-- Delete the schema.
m_db=# DROP SCHEMA tpcds;
```

4.4.2.10.2 EXECUTE DIRECT

Description

Executes an SQL statement on a specified node. Generally, the cluster automatically allocates an SQL statement to proper nodes. EXECUTE DIRECT is mainly used for database maintenance and testing.

Precautions

- When **enable_nonsysadmin_execute_direct** is **off**, only a system administrator has the **EXECUTE DIRECT** permission.
- To ensure data consistency across nodes, only the SELECT statement can be used. Transaction statements, DDL, and DML cannot be used.
- When the stddev aggregation calculation is performed on the specified DN using such statements, the result set is returned in triplet. For example, {3, 8, 30} indicates that the count result is 3, the sum result is 8, and the sum of squares is 30. When the AVG aggregation calculation is performed on the specified DN using such statements, the result set is returned in a binary tuple, for example, {4,2}. The result of count is 4, and that of sum is 2.
- When multiple nodes are specified, aggregate functions are not supported. If the query contains an aggregate function, the message "EXECUTE DIRECT on multinode not support agg functions." is returned.
- CN nodes do not store user table data. Therefore, do not execute **SELECT** for querying user tables on a CN node.
- If the SQL statement to be executed is also **EXECUTE DIRECT**, do not nest it into **EXECUTE DIRECT**; instead, directly execute the inner **EXECUTE DIRECT**.
- The query result of the **agg** function is inconsistent with that on the CN. Multiple pieces of information are returned. The **array_avg** function is not supported.
- If the **nodeoid** or **nodeoid list** parameter is specified, this parameter takes effect only when the **enable_direct_standby_datanodes** parameter is successfully set in distributed mode. To query data on a specified DN, invalid or duplicate **nodeoid** values are not supported, and **nodeoid** and **nodename** cannot be used together.

Syntax

```
EXECUTE DIRECT ON ( nodename [, ... ] ) query ;
EXECUTE DIRECT ON ( nodeoid [, ... ] ) query;
EXECUTE DIRECT ON { COORDINATORS | DATANODES | ALL } query;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameters

- **nodename**

  Specifies the node name.

  Value range: an existing node name

- **nodeoid**

  Specifies the node OID.

  Value range: an existing DN OID, which can be obtained from the **PGXC_NODE** system catalog.

- **query**

  Specifies the query statement to be executed.

- **COORDINATORS**

  Runs the query statement on all CNs.

- **DATANODES**

  Runs the query statement on all DNs.

- **ALL**

  Runs the query statement on all CNs and DNs.

Examples

```
-- Query the name of the current node in the cluster.
gaussdb=# SELECT node_name,oid FROM pgxc_node WHERE node_type IN ('C','D') ORDER BY 1;
     node_name     |  oid
-------------------+-------
 cn_5001           | 15517
 cn_5002           | 16387
 cn_5003           | 16391
 dn_6001_6002_6003 | 16384
 dn_6004_6005_6006 | 16388
 dn_6007_6008_6009 | 16392
(6 rows)


-- Create a range-distributed table and insert data into the table.
gaussdb=# CREATE TABLE tbl_range(
  id       INT,
  name     VARCHAR(20),
  province VARCHAR(60),                    -- Province
  country   VARCHAR(30) DEFAULT 'China'        -- Country
)DISTRIBUTE BY RANGE(id)(
  SLICE s1 VALUES LESS THAN (100) DATANODE dn_6001_6002_6003,
  SLICE s2 VALUES LESS THAN (200) DATANODE dn_6004_6005_6006,
  SLICE s3 VALUES LESS THAN (MAXVALUE) DATANODE dn_6007_6008_6009
);
gaussdb=# INSERT INTO tbl_range VALUES (52,'Bob','Beijing','China'),
                                       (100,'Ben','Shanghai','China'),
                                       (150,'Scott','Guangzhou','China'),
                                       (300,'Jordan','Beijing','China');

-- Query the table data distributed on the dn_6007_6008_6009 node.
gaussdb=# EXECUTE DIRECT ON(dn_6007_6008_6009) 'SELECT * FROM tbl_range';
 id  |  name  | province | country
-----+--------+----------+---------
 300 | Jordan | Beijing  | China
(1 row)

-- Delete the tbl_range table.
gaussdb=# DROP TABLE tbl_range;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.10.3 EXPLAIN

Description

Shows the execution plan of an SQL statement.

The execution plan shows how the tables referenced by the SQL statement will be scanned, for example, by plain sequential scan or index scan. If multiple tables are referenced, the execution plan also shows what join algorithms will be used to bring together the required rows from each input table.

The most critical part of the display is the estimated statement execution cost, which is the plan generator's guess at how long it will take to run the statement.

The **ANALYZE** option causes the statement to be actually executed. The total elapsed time expended within each plan node (in milliseconds) and total number of rows it actually returned are added to the display. This is useful for determining whether the plan generator's estimated value is close to the actual one.

Precautions

- The statement is actually executed when the **ANALYZE** option is used. If you want to use EXPLAIN on an INSERT, UPDATE, DELETE, or EXECUTE statement without letting the statement affect your data, use this approach:
```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```
- After parameterized **Remotequery** is generated in a statement and **max_datanode_for_plan** is set to a value greater than 0, the execution plan of parameterized **Remotequery** on DNs cannot be viewed. To view the execution plan of parameterized **Remotequery** on DNs, perform the following steps:
```
-- 1. Use the auto_explain parameter to view a DN plan.
SET enable_auto_explain = on;
SET auto_explain_level = log;
SET enable_opfusion=off;
-- If you do not want to modify the data, do as follows:
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
-- Otherwise, do as follows:
EXPLAIN ANALYZE ...;
-- Then, the execution plan on each DN is recorded in gs_log of the DN.

-- 2. Use the statement_history view to view the DN plan.
-- Use the dynamic_func_control function to track.
SELECT * FROM dynamic_func_control('GLOBAL', 'STMT', 'TRACK', '{"unique_sql_id", "L2"}');
-- Run the SQL statement to be viewed.
SELECT ...;
-- View the statement plan on the corresponding DN.
SELECT unique_query_id,query_plan from statement_history WHERE unique_query_id='unique_sql_id';
```

Syntax

- Display the execution plan of an SQL statement, which supports multiple options and has no requirements for the order of options.
```
EXPLAIN [ (  option  [, ...] )  ] statement;
```
The syntax of the **option** clause is as follows:
```
ANALYZE [ boolean ] |
   VERBOSE [ boolean ] |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    COSTS [ boolean ] |
    CPU [ boolean ] |
    DETAIL [ boolean ] |
    NODES [ boolean ] |
    NUM_NODES [ boolean ] |
    BUFFERS [ boolean ] |
    TIMING [ boolean ] |
    PLAN [ boolean ] |
    BLOCKNAME [ boolean ] |
    PERFORMANCE |
    FORMAT { TEXT | XML | JSON | YAML }
```

- Display the execution plan of an SQL statement, where options are in order.

```
EXPLAIN  { [   ANALYZE    ] [ VERBOSE  ] } statement;
```

Parameters

- **statement**

  Specifies the SQL statement to explain.

- **ANALYZE boolean**

  Specifies whether to display actual run times and other statistics. When two parameters are used at the same time, the latter one in **option** takes effect.

  The options are as follows:

  – **TRUE** (default): displays them.

  – **FALSE**: does not display them.

- **VERBOSE boolean**

  Specifies whether to display additional information regarding the plan.

  The options are as follows:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **COSTS boolean**

  Specifies whether to display the estimated total cost of each plan node, estimated number of rows, estimated width of each row.

  The options are as follows:

  – **TRUE** (default): displays them.

  – **FALSE**: does not display them.

- **CPU boolean**

  Specifies whether to display CPU usage. This parameter must be used together with **ANALYZE**.

  The options are as follows:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

- **DETAIL boolean**

  Specifies whether to display information about database nodes. This parameter must be used together with **ANALYZE**.

  The options are as follows:

  – **TRUE** (default): displays it.

  – **FALSE**: does not display it.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **NODES boolean**

  Specifies whether to display information about the nodes executed by query.

  The options are as follows:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.

- **NUM_NODES boolean**

  Specifies whether to display the number of executing nodes.

  The options are as follows:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.

- **BUFFERS boolean**

  Specifies whether to display buffer usage. This parameter must be used together with **ANALYZE**.

  The options are as follows:

  - **TRUE**: displays it.
  - **FALSE** (default): does not display it.

- **TIMING boolean**

  Specifies whether to display the actual startup time and time spent on the output node. This parameter must be used together with **ANALYZE**.

  The options are as follows:

  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.

- **PLAN**

  Specifies whether to store the execution plan in **PLAN_TABLE**. If this parameter is set to **on**, the execution plan is stored in **PLAN_TABLE** and is not displayed on the screen. Therefore, this parameter cannot be used together with other parameters when it is set to **on**.

  The options are as follows:

  - **ON** (default): stores the execution plan in **PLAN_TABLE** and does not display it on the screen. If the plan is stored successfully, "EXPLAIN SUCCESS" is returned.
  - **OFF**: does not store the execution plan but displays it on the screen.

- **BLOCKNAME boolean**

  Specifies whether to display the query block where each operation of the plan is located. When this option is enabled, each name of the query block where each operation is performed is displayed in the **Query Block** column. This helps users obtain the query block name and use hints to modify the execution plan.

  - **TRUE** (default value): The name of the query block where each operation is located is displayed in the **Query Block** column. This option must be used in the pretty mode.
  - **FALSE**: The plan display is not affected.

- **FORMAT**

  Specifies the output format.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: **TEXT**, **XML**, **JSON**, and **YAML**

Default value: **TEXT**

- **PERFORMANCE**

  Prints all relevant information in execution. Some information is described as follows:

  - **ex c/r**: indicates the average number of CPU cycles used by each row, which is equal to **(ex cyc) / (ex row)**.
  - **ex row**: indicates the number of executed rows.
  - **ex cyc**: indicates the number of used CPU cycles.
  - **inc cyc**: indicates the total number of CPU cycles used by subnodes.
  - **shared hit**: indicates the shared buffer hits of the operator.
  - **loops**: indicates the number of operator loop execution times.
  - **total_calls**: indicates the total number of generated elements.
  - **remote query poll time stream gather**: indicates the operator used to listen to the network poll time when data on each DN reaches the CN.
  - **deserialize time**: indicates the time required for deserialization.
  - **estimated time**: indicates the estimated time.
  - **Network Poll Time**: indicates the duration for the libcomm receiver to wait for data during distributed stream network communication.
  - **Stream Send time**: indicates the time consumed by libcomm or libpq to send data during distributed stream network communication.
  - **OS Kernel Send time**: indicates the time required for the OS layer to send data during distributed stream network communication. This parameter is displayed only when the value is greater than **0**.
  - **Wait Quota time**: indicates the duration for libcomm to wait for the peer end to send the quota traffic control size during distributed stream network communication. This parameter is displayed only when the value is greater than **0**.
  - **Data Serialize time**: indicates the data serialization time during distributed stream network communication.
  - **Data Copy time**: indicates the data replication time during distributed stream network communication. This parameter is displayed only when the value is greater than **0**.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

  -- Create the tpcds.customer_address table.
m_db=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAABAAAAAAA'),(10000,
'AAAAAAAACAAAAAAA');

-- Create the tpcds.customer_address_p1 table.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# CREATE TABLE tpcds.customer_address_p1 LIKE tpcds.customer_address;

-- Change the value of explain_perf_mode to normal.
m_db=# SET explain_perf_mode=normal;

-- Display an execution plan for simple queries in the table.
m_db=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
                  QUERY PLAN
-----------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: All datanodes
(2 rows)

-- Generate an execution plan in JSON format (with explain_perf_mode being normal).
m_db=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
              QUERY PLAN
-------------------------------------
 [                                  +
   {                                +
     "Plan": {                        +
       "Node Type": "Data Node Scan",+
       "Startup Cost": 0.00,         +
       "Total Cost": 0.00,           +
       "Plan Rows": 0,               +
       "Plan Width": 0,              +
       "Nodes": "All datanodes"      +
     }                              +
   }                                +
 ]
(1 row)

-- If there is an index and we use a query with an indexable WHERE condition, a different plan may be
displayed.
m_db=# EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
                  QUERY PLAN
-----------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Node/s: datanode2
(2 rows)

-- Generate an execution plan in YAML format (with explain_perf_mode being normal).
m_db=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
           QUERY PLAN
---------------------------------
 - Plan:                        +
     Node Type: "Data Node Scan"+
     Startup Cost: 0.00         +
     Total Cost: 0.00           +
     Plan Rows: 0               +
     Plan Width: 0              +
     Nodes: "datanode2"
(1 row)

-- Suppress the execution plan of cost estimation.
m_db=# EXPLAIN(COSTS FALSE) SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
     QUERY PLAN
---------------------
 Data Node Scan
   Node/s: datanode2
(2 rows)

-- Generate an execution plan with aggregate functions for a query.
m_db=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=13.24..13.28 rows=1 width=12)
   ->  Streaming (type: GATHER)  (cost=13.24..13.28 rows=2 width=12)
```

```
      Node/s: All datanodes
      ->  Aggregate  (cost=13.17..13.18 rows=2 width=12)
            ->  Seq Scan on customer_address_p1  (cost=0.00..13.16 rows=7 width=4)
                  Filter: (ca_address_sk < 10000)
(6 rows)

-- Delete the tpcds.customer_address_p1 table.
m_db=# DROP TABLE tpcds.customer_address_p1;

-- Delete the tpcds.customer_address table.
m_db=# DROP TABLE tpcds.customer_address;

-- Delete the schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

ANALYZE

4.4.2.11 G

4.4.2.11.1 GRANT

Description

Grants permissions to roles and users.

GRANT is used in the following scenarios:

- **Granting system permissions to roles or users**

  System permissions are also called user attributes, including SYSADMIN, CREATEDB, CREATEROLE, AUDITADMIN, MONADMIN, OPRADMIN, POLADMIN, INHERIT, REPLICATION, and LOGIN.

  They can be specified only by the CREATE ROLE or ALTER ROLE statement. The SYSADMIN permissions can be granted and revoked using GRANT ALL PRIVILEGES and REVOKE ALL PRIVILEGES, respectively. System permissions cannot be inherited by a user from a role, and cannot be granted using PUBLIC.

- **Granting database object permissions to roles or users**

  Grant permissions on a database object (table, view, column, database, function, schema, or tablespace) to a role or user.

  GRANT gives specific permissions on a database object to one or more roles. These permissions are added to those already granted, if any.

  The keyword PUBLIC indicates that the permissions are to be granted to all roles, including those that might be created later. PUBLIC can be thought of as an implicitly defined group that always includes all roles. Any particular role will have the sum of permissions granted directly to it, permissions granted to any role it is presently a member of, and permissions granted to PUBLIC.

  If WITH GRANT OPTION is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. This option cannot be granted to PUBLIC, which is a unique M-compatible database attribute.

  M-compatible database grants the permissions on certain types of objects to PUBLIC. By default, permissions on tables, columns, sequences, foreign

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

servers, schemas, and tablespaces are not granted to PUBLIC users, but the following permissions are granted to PUBLIC users: CONNECT and CREATE TEMP TABLE permissions on databases, EXECUTE permission on functions, and USAGE permission on languages and data types (including domains). An object owner can revoke the default permissions granted to PUBLIC users and grant permissions to other users as needed. For security purposes, you are advised to create an object and set its permissions in the same transaction so that other users do not have time windows to use the object.

By default, an object owner has all permissions on the object. For security purposes, the owner can discard some permissions. However, the ALTER, DROP, COMMENT, INDEX, VACUUM, and re-grantable permissions of the object are inherent permissions implicitly owned by the owner.

- **Granting the permissions of one role or user to others**

  Grant the permissions of one role or user to others. In this case, every role or user can be regarded as a set of one or more database permissions.

  If WITH ADMIN OPTION is specified, the recipients can in turn grant the permissions to other roles or users or revoke the permissions they have granted to other roles or users. If recipients' permissions are changed or revoked later, the grantees' permissions will also change.

  When separation of duties is disabled, a SYSADMIN can grant or revoke the permissions of any non-permanent user, OPRADMIN, or private user role, and a user with the CREATEROLE permission can grant or revoke the permissions of any non-SYSADMIN, built-in role, permanent user, OPRADMIN, or private user role.

- **Granting ANY permissions to roles or users**

  Grant ANY permissions to a specified role or user. For details about the value range of the ANY permissions, see the syntax. If WITH ADMIN OPTION is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users. The ANY permissions can be inherited by a role but cannot be granted to PUBLIC. When separation of duties is disabled, an initial user and SYSADMIN can grant the ANY permissions to or revoke them from any role or user.

  Currently, the following ANY permissions are supported: CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, CREATE ANY SEQUENCE, ALTER ANY SEQUENCE, DROP ANY SEQUENCE, SELECT ANY SEQUENCE, ALTER ANY INDEX, and DROP ANY INDEX. For details about the ANY permission scope, see **Table 4-26**.

Precautions

- It is not allowed to grant the ANY permissions to PUBLIC or revoke the ANY permissions from PUBLIC.
- The ANY permissions are database permissions and are valid only for database objects that are granted the permissions. For example, SELECT ANY TABLE only allows a user to view all user table data in the current database, but the user does not have the permission to view user tables in other databases.
- The ANY permissions and the original permissions do not affect each other.
- If a user is granted the CREATE ANY TABLE permission, the owner of a table created in a schema with the same name as the user is the creator of the

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

schema. When the user performs other operations on the table, the user needs to be granted the corresponding operation permission. Similar to CREATE ANY SEQUENCE and CREATE ANY INDEX, the owner of an object created in a schema with the same name is the owner of the schema with the same name.

- When GRANT is used to grant a user the permission to use a table, if the permission is not properly used, ALTER may be used to add expressions to the default values and constraints of the table, or indexes may be created to add expressions to index. In this case, the permission may be exploited.

Syntax

- Grant the table or view access permission to a specified user or role.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
COMMENT | INDEX | VACUUM } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

  NOTE

  To perform an operation on a table, you must have the operation permission on the table and USAGE permission for the schema to which the table belongs. To create an index on the table, you must have the CREATE permission on the schema and INDEX permission on the table.

- Grant the column access permission to a user or role.

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )} [, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
  ON [ TABLE ] table_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

  NOTE

  If you have the permission to access a table, you have the permission to access all columns in the table by default. To grant only the access permission on a column in a table, you need to revoke the access permission on the table.

- Grant the sequence access permission to a specified user or role.

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ SEQUENCE ] sequence_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the subcluster access permission to a user or role.

```
GRANT { { CREATE | USAGE | COMPUTE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON NODE GROUP group_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

  NOTE

  When the CREATE permission of a sub-cluster is granted to a specified user or role, the USAGE and COMPUTE permissions are granted to the specified user or role by default.

- Grant the database access permission to a user or role.

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON DATABASE database_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> This syntax takes effect when the **grant_database_nomapping** option in the GUC parameter **m_format_behavior_compat_options** is enabled. It grants the database access permission to a specified user or role. If the **grant_database_nomapping** option is disabled, GRANT ON DATABASE is mapped to GRANT ON SCHEMA, granting the schema access permission to a specified user or role. GRANT ON DATABASE supports only the syntax options supported by GRANT ON SCHEMA.

- Grant the schema access permission to a user or role.

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON SCHEMA schema_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

> **NOTE**
>
> When you grant table or view permissions to other users, you also need to grant the USAGE permission on the schema that the tables and views belong to. Without the USAGE permission, the users with table or view permissions can only see the object names, but cannot access them. This syntax cannot be used to grant the permission to create tables in schemas with the same name, but you can use the syntax for granting permission of a role to another user or role to achieve the same effect.

- Grant a role's permissions to another user or role.

```
GRANT role_name [, ...]
  TO role_name [, ...]
  [ WITH ADMIN OPTION ];
```

- Grant the SYSADMIN permission to a role.

```
GRANT ALL PRIVILEGES
  TO role_name;
```

- Grant the ANY permissions to another user or role.

```
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT
ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX |
ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
  SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX |  } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

Parameters

GRANT permissions are classified as follows:

- **SELECT**

  Allows SELECT on a specified table, view, or sequence. The SELECT permission on the corresponding column is also required for UPDATE or DELETE.

- **INSERT**

  Allows INSERT of a new row into a table.

- **UPDATE**

  Allows UPDATE of any column of a table. Generally, UPDATE also requires the SELECT permission to query which rows need to be updated. SELECT ... FOR UPDATE and SELECT ... FOR SHARE also require this permission on at least one column, in addition to the SELECT permission.

- **DELETE**

  Allows DELETE of a row from a table. Generally, DELETE also requires the SELECT permission to query which rows need to be deleted.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- TRUNCATE

  Allows TRUNCATE on a table.

- **REFERENCES**

  Allows creation of a foreign key constraint referencing a table. This permission is required on both referencing and referenced tables.

- **CREATE**
  - For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have the CREATE permission on the schema of the object. For tablespaces, allows tables to be created within the tablespace, and allows databases and schemas to be created that have the tablespace as their default tablespace.
  - For subclusters, allows tables to be created within the subcluster.

- **EXECUTE**

  Allows calling a function, including use of any operators that are implemented on top of the function.

- **USAGE**
  - For procedural languages, allows use of the language for the creation of functions in that language.
  - For schemas, allows access to objects contained in the schema. Without this permission, it is still possible to see the object names.
  - For sequences, allows use of the nextval function.
  - For subclusters, allows users who can access objects contained in the schema to access tables in the subcluster.

- **ALTER**

  Allows users to modify the attributes of a specified object, excluding the owner and schema of the object.

- **DROP**

  Allows users to drop specified objects.

- **COMMENT**

  Allows users to define or modify comments of a specified object.

- **INDEX**

  Allows users to create indexes on specified tables, manage indexes on the tables, and perform REINDEX operations on the tables.

- **VACUUM**

  Allows users to perform ANALYZE and VACUUM operations on specified tables.

- **ALL PRIVILEGES**

  Grants all available permissions to a user or role at a time. Only a system administrator has the GRANT ALL PRIVILEGES permission.

GRANT parameters are as follows:

- **role_name**

  Specifies the username that has been created. For details about the username requirements, see **•user_name**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **table_name**

  Specifies the table name.

- **column_name**

  Specifies the column name.

- **group_name**

  Specifies the subcluster name.

- **schema_name**

  Specifies the schema name.

- **sequence_name**

  Specifies the sequence name.

- **WITH GRANT OPTION**

  If WITH GRANT OPTION is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to PUBLIC.

When a non-owner of an object attempts to grant permissions on the object:

- The statement will fail outright if the user has no permissions whatsoever on the object.
- As long as some permission is available, the statement will proceed, but it will grant only those permissions for which the user has grant options.
- The GRANT ALL PRIVILEGES forms will issue a warning message if no grant options are held, while the other forms will issue a warning if grant options for any of the permissions specifically named in the statement are not held.

NOTE

When separation of duties is disabled, database administrators can access all objects, regardless of object permission settings. This is comparable to the permissions of **root** in a Unix system. As with **root**, it is unwise to operate as a system administrator except when necessary.

- **WITH ADMIN OPTION**

  If WITH ADMIN OPTION is specified for a role, the grantee can grant the role to other roles or users or revoke the role from other roles or users.

  For the ANY permissions, if WITH ADMIN OPTION is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users.

Table 4-26 ANY permissions

System Permission	Description
CREATE ANY TABLE	Users can create tables or views in the public and user schemas.
ALTER ANY TABLE	Users' ALTER permission on tables or views in the public and user schemas. If the users want to modify the unique index of a table to add a primary key constraint or unique constraint to the table, the users must be granted the index permission on the table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

System Permission	Description
DROP ANY TABLE	Users' DROP permission on tables or views in the public and user schemas.
SELECT ANY TABLE	Users' SELECT permission on tables or views in the public and user schemas.
UPDATE ANY TABLE	Users' UPDATE permission on tables or views in the public and user schemas.
INSERT ANY TABLE	Users' INSERT permission on tables or views in the public and user schemas.
DELETE ANY TABLE	Users' DELETE permission on tables or views in the public and user schemas.
CREATE ANY SEQUENCE	Users can create sequences in the public and user schemas.
CREATE ANY INDEX	Users can create indexes in the public and user schemas. The users must be granted the permission to create tablespaces to create a partitioned table index in a tablespace.
ALTER ANY SEQUENCE	Users have the ALTER permission on sequences in public and user schemas, excluding modifying the owner of a sequence.
DROP ANY SEQUENCE	Users' DROP permission on sequences in the public and user schemas.
SELECT ANY SEQUENCE	Users' SELECT, USAGE, and UPDATE permissions on sequences in the public and user schemas.
ALTER ANY INDEX	Users' ALTER permission on indexes in the public and user schemas. To rename an index, users also need the permission to create objects in the schema where the index is located. If tablespace operations are involved, users need to have the corresponding operation permission on the tablespace. To set an index to **UNUSABLE**, users must have the DROP ANY INDEX permission.
DROP ANY INDEX	Users' DROP permission on indexes in the public and user schemas.

NOTE

When a user is granted any ANY permission, they automatically receive the USAGE permission on the public and user-defined schemas. However, they do not receive the USAGE permission on system schemas, except for the public schema explicitly supported by GaussDB. For details about the schemas supported by GaussDB, see section "Schema" in *Developer Guide*.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

Example 1: Grant system permissions to a user or role.

Create the **joe** user and grant the SYSADMIN permission to it.

```
m_db=# CREATE USER joe PASSWORD '********';
m_db=# GRANT ALL PRIVILEGES TO joe;
```

Then **joe** has the SYSADMIN permission.

Example 2: Grant object permissions to a user or role.

1. Revoke the SYSADMIN permission from the **joe** user. Grant the usage permission of the tpcds schema and all permissions on the **tpcds.reason** table to **joe**.

   ```
   m_db=# CREATE SCHEMA tpcds;
   CREATE SCHEMA
   m_db=# CREATE TABLE tpcds.reason
   (
   r_reason_sk         INTEGER      NOT NULL,
   r_reason_id         CHAR(16)     NOT NULL,
   r_reason_desc       VARCHAR(20)
   );
   CREATE TABLE
   m_db=# REVOKE ALL PRIVILEGES FROM joe;
   m_db=# GRANT USAGE ON SCHEMA tpcds TO joe;
   m_db=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
   ```

   Then **joe** has all permissions on the **tpcds.reason** table, including create, retrieve, update, and delete.

2. Grant the retrieve permission of **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns and the update permission of the **r_reason_desc** column in the **tpcds.reason** table to **joe**.

   ```
   m_db=# GRANT SELECT (r_reason_sk,r_reason_id,r_reason_desc),UPDATE (r_reason_desc) ON
   tpcds.reason TO joe;
   ```

   Then **joe** has the retrieve permission on the **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns in the **tpcds.reason** table. To enable **joe** to grant these permissions to other users, execute the following statement:

   ```
   m_db=# GRANT SELECT (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
   ```

   Create the **tpcds_manager** role, grant the access and object creation permissions of the tpcds schema to **tpcds_manager**, but do not allow **tpcds_manager** to grant these permissions to others.

   ```
   m_db=# CREATE ROLE tpcds_manager PASSWORD '********';
   m_db=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
   ```

Example 3: Grant a user's or role's permissions to other users or roles.

1. Create the **manager** role, grant **joe**'s permissions to **manager**, and allow **manager** to grant these permissions to others.

   ```
   m_db=# CREATE ROLE manager PASSWORD '********';
   m_db=# GRANT joe TO manager WITH ADMIN OPTION;
   ```

2. Create the **senior_manager** user and grant **manager**'s permissions to it.

   ```
   m_db=# CREATE ROLE senior_manager PASSWORD '********';
   m_db=# GRANT manager TO senior_manager;
   ```

3. Revoke permissions and delete users.

   ```
   m_db=# REVOKE joe FROM manager;
   m_db=# REVOKE manager FROM senior_manager;
   m_db=# DROP USER manager;
   ```

Example 4: Revoke the granted permissions and delete the roles and users.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
m_db=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
m_db=# DROP ROLE tpcds_manager;
m_db=# DROP ROLE senior_manager;
m_db=# DROP USER joe CASCADE;
m_db=# DROP TABLE tpcds.reason;
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

REVOKE

4.4.2.12 I

4.4.2.12.1 INSERT

Description

Inserts one or more rows of data into a table.

Precautions

- You must have the INSERT permission on a table to insert data into it. If a user is granted the INSERT ANY TABLE permission, the user has the USAGE permission on all schemas except system schemas and the INSERT permission on tables in these schemas.
- Use of the RETURNING clause requires the SELECT permission on all columns mentioned in RETURNING.
- If ON DUPLICATE KEY UPDATE is used, you must have the INSERT and UPDATE permissions on the table and the SELECT permission on the columns of the UPDATE clause.
- If you use the query clause to insert rows from a query, you need to have the SELECT permission on any table or column used in the query.
- The generated column cannot be directly written. You cannot specify values for generated columns in INSERT, but you can specify the keyword DEFAULT.

Syntax

```
INSERT [/*+ plan_hint */] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
   [(col_name [, col_name] ...)]
   {VALUES | VALUE} (value_list) [, (value_list)] ...
   [ON DUPLICATE KEY UPDATE column_name = {expression | DEFAULT}[, ...]]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];

INSERT [/*+ plan_hint */] [IGNORE] [INTO] tbl_name
   [PARTITION (partition_name [, partition_name] ...)]
   SET column_name = {expression | DEFAULT}[, ...]
   [ON DUPLICATE KEY UPDATE column_name = {expression | DEFAULT}[, ...]]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];

INSERT [/*+ plan_hint */] [IGNORE] [INTO] tbl_name
   [PARTITION (partition_name [, partition_name] ...)]
   [(col_name [, col_name] ...)]
   query
   [ON DUPLICATE KEY UPDATE column_name = {expression | DEFAULT}[, ...]]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
INSERT [/*+ plan_hint */] [IGNORE] [INTO] view_name
   [(col_name [, col_name] ...)]
   {VALUES | VALUE} (value_list) [, (value_list), ...]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];

INSERT [/*+ plan_hint */] [IGNORE] [INTO] view_name
   SET column_name = {expression | DEFAULT}[, ...]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];

INSERT [/*+ plan_hint */] [IGNORE] [INTO] view_name
   [(col_name [, col_name] ...)]
   query
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

Parameters

- plan_hint clause

  Follows the INSERT keyword in the **/*+ */** format. It is used to optimize the plan of an INSERT statement block. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **IGNORE**

  When the INSERT statement uses the IGNORE keyword, some ERROR-level errors can be degraded to WARNING-level errors, and invalid values can be adjusted to the closest values based on error scenarios. GaussDB supports the following error downgrade scenarios:

  - Violation of the NOT NULL constraint
  - UNIQUE KEY conflict
  - No partition found for the inserted value
  - Unmatch between the inserted data and the specified partition
  - Multiple rows returned for a subquery
  - **sql_mode** set to loose

- **table_name**

  Specifies the name of the target table where data will be inserted.

  Value range: existing table names.

- **partition_name**

  Specifies the name of the partition.

- **column_name**

  Specifies the name of a column in a table.

  - The column name can be qualified with a subcolumn name or array index, if needed.
  - Each column not present in the column list will be filled with a default value, either its declared default value or **NULL** if there is none. Inserting data into only some columns of a composite type leaves the other columns **NULL**.
  - The target column names (specified by **column_name**) can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.
  - The target columns are the first *N* column names, if there are only *N* columns supplied by the VALUE clause or query.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - The values provided by the VALUE clause and QUERY are joined with the corresponding columns from left to right in the table.

  Value range: existing column names.

- **expression**

  Specifies an expression or a value to assign to the corresponding column.

  - In the INSERT ON DUPLICATE KEY UPDATE statement, expression can be **VALUES(***column_name***)**, indicating that the value of **column_name** corresponding to the conflict row is referenced. **VALUES(***column_name***)** can be nested in a compound expression, for example, **VALUES(***column_name***) + 1**, **VALUES(***column_name***) + VALUES(***column_name***)**, and **function_name(VALUES(***column_name***))**.

    NOTE

    VALUES(*column_name***)** can be used only in the ON DUPLICATE KEY UPDATE clause.

  - If single-quotation marks (') are inserted into a column, the single-quotation marks (') need to be used for escape.
  - If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

- **DEFAULT**

  Specifies the default value of a column. The value is **NULL** if there is no default value.

- **query**

  Specifies a query statement (SELECT statement) that uses the query result as the inserted data.

- **RETURNING**

  Returns the inserted rows. The syntax of the RETURNING list is identical to that of the output list of SELECT.

- **ON DUPLICATE KEY UPDATE**

  For a table with a unique constraint (UNIQUE INDEX or PRIMARY KEY), if the inserted data violates the unique constraint, the UPDATE clause is executed on the conflicting row to complete the update. For a table without a unique constraint, only the INSERT operation is performed. When using UPDATE, you can use **VALUES()** to select the column corresponding to the source data.

  - If a table has multiple unique constraints and the inserted data violates multiple unique constraints, only the first row that has a conflict is updated. (The check sequence is closely related to index maintenance. Generally, the conflict check is performed on the index that is created first.)
  - If multiple rows are inserted and these rows conflict with the same row in the table, the system inserts or updates the first row and then updates other rows in sequence.
  - Primary keys and unique index columns cannot be updated.
  - **expression** does not support subquery expressions.
  - If **query** is a subquery, the column name in the subquery cannot be used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - When the ON DUPLICATE KEY UPDATE clause is used to update multiple columns, the result of the previous UPDATE statement does not affect the subsequent results. In addition, the same column cannot be set for multiple times.

- **view_name**

  Specifies the target view to be inserted.

  NOTE

  The restrictions on inserting views are as follows:

  1. Only columns that directly reference user columns in the base table can be inserted.
  2. A view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
  3. Views that contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause at the top layer are not supported.
  4. Views that contain set operations (UNION and EXCEPT) at the top layer are not supported.
  5. Views whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
  6. The ON DUPLICATE KEY UPDATE function is not supported.
  7. Table types supported in views include ordinary tables, temporary tables, global temporary tables, partitioned tables, Ustore tables, and Astore tables.
  8. In a multi-table join view, only one base table can be inserted at a time.
  9. When a view is joined, explicitly specified columns to be inserted or implicitly specified columns (columns specified during view creation) cannot reference columns of non-key-preserved tables For details about the key-preserved table, see **CREATE VIEW**.
  10. System views cannot be inserted.

Example

- **Insert a data record.**

  Example:

```
-- Create a table.
m_db=# CREATE TABLE test_t1(col1 INT,col2 VARCHAR(20));

-- Insert data.
m_db=# INSERT INTO test_t1 (col1, col2) VALUES (1,'AB');

-- Insert data into some columns of the table.
m_db=# INSERT INTO test_t1 (col1) VALUES (2);

-- There is no parenthesis on the left of the VALUES keyword. Values must be added to all columns in the parenthesis on the right according to the table structure sequence.
m_db=# INSERT INTO test_t1 VALUES (3,'AC');

-- Query the table.
m_db=# SELECT * FROM test_t1 ORDER BY col1;
 col1 | col2
------+------
    1 | AB
    2 |
    3 | AC
(3 rows)

-- Drop the table.
m_db=# DROP TABLE test_t1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **Insert multiple data records.**

  Example:

```
-- Create tables.
m_db=# CREATE TABLE test_t2(col1 INT,col2 VARCHAR(20));
m_db=# CREATE TABLE test_t3(col1 INT,col2 VARCHAR(20));

-- Insert multiple data records.
m_db=# INSERT INTO test_t2 (col1, col2) VALUES (10,'AA'),(20,'BB'),(30,'CC');

-- Query the table.
m_db=# SELECT * FROM test_t2;
 col1 | col2
------+------
   10 | AA
   20 | BB
   30 | CC
(3 rows)

-- Insert data in test_t2 into test_t3.
m_db=# INSERT INTO test_t3 SELECT * FROM test_t2;

-- Query the table.
m_db=# SELECT * FROM test_t3;
 col1 | col2
------+------
   10 | AA
   20 | BB
   30 | CC
(3 rows)

-- Drop tables.
m_db=# DROP TABLE test_t2;
m_db=# DROP TABLE test_t3;
```

- **ON DUPLICATE KEY UPDATE**

  Example:

```
-- Create a table.
m_db=# CREATE TABLE test_t4 (id INT PRIMARY KEY, info VARCHAR(10));
m_db=# INSERT INTO test_t4 VALUES (1, 'AA'), (2,'BB'), (3, 'CC');

-- Use the ON DUPLICATE KEY UPDATE keyword.
m_db=# INSERT INTO test_t4 VALUES (3, 'DD'), (4, 'EE') ON DUPLICATE KEY UPDATE info =
VALUES(info);

-- Query the table.
m_db=# SELECT * FROM test_t4;
 id | info
----+------
  1 | AA
  2 | BB
  4 | EE
  3 | DD

-- Drop the table.
m_db=# DROP TABLE test_t4;
```

- **INSERT IGNORE**

  Example 1: Violation of the NOT NULL constraint

```
-- Create a table.
m_db=# CREATE TABLE test_t5(f1 INT NOT NULL);
CREATE TABLE

-- Use the IGNORE keyword.
m_db=# INSERT IGNORE INTO test_t5 VALUES(NULL);
WARNING:  null value in column "f1" violates not-null constraint
DETAIL:  Failing row contains (null).
INSERT 0 1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Query the table.
m_db=# SELECT * FROM test_t5;
 f1
----
  0
(1 row)

-- Drop the table.
m_db=# DROP TABLE test_t5;
```

Example 2: UNIQUE KEY conflict

```
-- Create a table.
m_db=# CREATE TABLE test_t6(f1 INT PRIMARY KEY);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "test_t6_pkey" for table "test_t6"
CREATE TABLE

-- Insert data.
m_db=# INSERT INTO test_t6 VALUES(1);
INSERT 0 1

-- Use the IGNORE keyword.
m_db=# INSERT IGNORE INTO test_t6 VALUES(1);
WARNING:  duplicate key value violates unique constraint "test_t6_pkey"
INSERT 0 0

-- Query the table.
m_db=# SELECT * FROM test_t6;
 f1
----
  1
(1 row)

-- Drop the table.
m_db=# DROP TABLE test_t6;
DROP TABLE
```

Example 3: No partition found for the inserted value

```
-- Create a table.
m_db=# CREATE TABLE test_t7(f1 INT, f2 INT) PARTITION BY LIST(f1) (PARTITION p0 VALUES(1, 4,
7), PARTITION p1 VALUES (2, 5, 8));
CREATE TABLE

-- Use the IGNORE keyword.
m_db=# INSERT IGNORE INTO test_t7 VALUES(3, 5);
WARNING:  inserted partition key does not map to any table partition
INSERT 0 0

-- Query the table.
m_db=# SELECT * FROM test_t7;
 f1 | f2
----+----
(0 rows)

-- Drop the table.
m_db=# DROP TABLE test_t7;
DROP TABLE
```

Example 4: Unmatch between the inserted data and the specified partition

```
-- Create a table.
m_db=# CREATE TABLE test_t8(f1 INT NOT NULL, f2 TEXT, f3 INT) PARTITION BY RANGE(f1)
(PARTITION p0 VALUES LESS THAN(5), PARTITION p1 VALUES LESS THAN(10), PARTITION p2
VALUES LESS THAN(15), PARTITION p3 VALUES LESS THAN(MAXVALUE));
CREATE TABLE

-- Use the IGNORE keyword.
m_db=# INSERT IGNORE INTO test_t8 PARTITION(p2) VALUES(20, 'Jan', 1);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
WARNING:  inserted partition key does not map to the table partition
DETAIL:  N/A.
INSERT 0 0

-- Query the table.
m_db=# SELECT * FROM test_t8;
 f1 | f2 | f3
----+----+----
(0 rows)

-- Drop the table.
m_db=# DROP TABLE test_t8;
DROP TABLE
```

Example 5: Multiple rows returned for a subquery

```
-- Create a table.
m_db=# CREATE TABLE test_t9(f1 INT, f2 INT);
CREATE TABLE

-- Insert data.
m_db=# INSERT INTO test_t9 VALUES(1, 1), (2, 2), (3, 3);
INSERT 0 3

-- Use the IGNORE keyword.
m_db=# INSERT IGNORE INTO test_t9 VALUES((SELECT f1 FROM test_t9), 0);
WARNING:  more than one row returned by a subquery used as an expression
CONTEXT:  referenced column: f1
INSERT 0 1

-- Query the table.
m_db=# SELECT * FROM test_t9 WHERE f2 = 0;
 f1 | f2
----+----
    |  0
(1 row)

-- Drop the table.
m_db=# DROP TABLE test_t9;
DROP TABLE
```

- **Insert a view.**

  ### Example:

```
-- Create a schema.
m_db=# CREATE SCHEMA ins_view;
CREATE SCHEMA
m_db=# SET CURRENT_SCHEMA = 'ins_view';
SET

-- Create tables.
m_db=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
m_db=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE

-- Create a single table view.
m_db=# CREATE VIEW v_ins1 AS SELECT * FROM t1;
CREATE VIEW
m_db=# CREATE VIEW v_ins2 AS SELECT * FROM t1 WHERE y1 < 3;
CREATE VIEW

-- Insert data into t1 through a view.
m_db=# INSERT INTO v_ins1 VALUES (1, 1);
INSERT 0 1
m_db=# INSERT INTO v_ins2 VALUES (5, 5);
INSERT 0 1

-- Create a multi-table view.
m_db=# CREATE VIEW vv_ins AS SELECT * FROM t1, t2 WHERE x1 = x2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
CREATE VIEW

-- Insert data into t1 through a multi-table view.
m_db=# INSERT INTO vv_ins (x1, y1) VALUES (2, 2);
INSERT 0 1

-- Drop the schema.
m_db=# DROP SCHEMA ins_view;
NOTICE:  drop cascades to 5 other objects
DETAIL:  drop cascades to table t1
drop cascades to table t2
drop cascades to view v_ins1
drop cascades to view v_ins2
drop cascades to view vv_ins
DROP SCHEMA
```

Suggestions

- VALUES

  When you use the INSERT statement to insert data in batches, you are advised to combine multiple records into one statement to improve data loading performance.

  Example: **INSERT INTO sections VALUES (30, 'Administration', 31, 1900), (40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);**

4.4.2.13 L

4.4.2.13.1 LOCK

Description

LOCK TABLE obtains a table-level lock.

An M-compatible database always tries to select the lock mode with minimum constraints when automatically requesting a lock for a statement referenced by a table. Use **LOCK** if users need a more strict lock mode. For example, suppose an application runs a transaction at the Read Committed isolation level and needs to ensure that data in a table remains stable in the duration of the transaction. To achieve this, you could obtain READ lock mode over the table before the query. This will prevent concurrent data changes and ensure subsequent reads of the table see a stable view of committed data. It is because the READ lock mode conflicts with the ROW EXCLUSIVE mode required by writers, the statement will wait until any concurrent holders of ROW EXCLUSIVE mode locks commit or roll back. Therefore, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

Precautions

- **LOCK TABLE** is useless outside a transaction block: the lock would remain held only to the completion of the statement. If **LOCK TABLE** is out of any transaction block, an error is reported.
- If no lock mode is specified, then WRITE, the most restrictive mode, is used.
- There is no UNLOCK TABLE statement. Locks are always released at transaction end. Currently, M-compatible databases do not support the **UNLOCK TABLE** command.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- LOCK TABLE processes only table-level locks. For details about the rules for checking whether locks conflict with each other, see **Table 4-27**.
- If the **xc_maintenance_mode** parameter is not enabled, an error is reported when a WRITE lock is applied for a system catalog.
- Only the redistribution tool can use the automatic CANCEL service API.

Syntax

```
LOCK [ TABLE | TABLES ] [ONLY]
   tbl_name [lock_type]
    [, tbl_name [lock_type] ] ...
    [NOWAIT]
    [CANCELABLE];

lock_type: {
    READ
  | WRITE
  | IN ACCESS SHARE MODE
  | IN EXCLUSIVE MODE
}
```

Parameters

Table 4-27 Lock mode conflicts

Requested Lock Mode/ Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	READ	SHARE ROW EXCLUSIVE	EXCLUSIVE	WRITE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
READ	-	-	X	X	-	X	X	X

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Requested Lock Mode/ Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	READ	SHARE ROW EXCLUSIVE	EXCLUSIVE	WRITE
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
WRITE	X	X	X	X	X	X	X	X

LOCK parameters are as follows:

- **table_name**

  Specifies the name (optionally schema-qualified) of an existing table to lock.

  Tables are locked one-by-one in the order specified in the **LOCK TABLE** statement.

  Value range: an existing table name

- **ONLY**

  If **ONLY** is specified, only that table is locked. If it is not specified, the table and all its sub-tables are locked.

- **READ**

  Allows concurrent queries of a table but does not allow modification of the table.

  Acquired by **CREATE INDEX** (without **CONCURRENTLY**).

- **WRITE**

  Guarantees that the holder is the only transaction accessing the table in any way.

  The ALTER TABLE, DROP TABLE, TRUNCATE and REINDEX statements automatically request this lock.

  This is also the default lock mode for **LOCK TABLE** statements that do not specify a mode explicitly.

- **ACCESS SHARE**

  Allows only read operations on a table. In general, any SQL statements that only read a table and do not modify it will acquire this lock mode. The SELECT statement acquires an **ACCESS SHARE** lock on referenced tables.

- **EXCLUSIVE**

  Allows concurrent queries of the target table but does not allow any other operations.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

This mode allows only concurrent **ACCESS SHARE** locks; that is, only reads from the table can proceed in parallel with a transaction holding this lock mode.

No SQL statements automatically acquire this lock mode on user tables. However, it will be acquired on some system catalogs in case of some operations.

- **ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE ROW EXCLUSIVE**

  In M-compatible databases, LOCK TABLE cannot be used to obtain the locks of these modes.

- **NOWAIT**

  Specifies that LOCK TABLE does not wait for any conflicting locks to be released. If the lock cannot be obtained immediately, the command exits and an error message is displayed.

  If a table-level lock is obtained without specifying **NOWAIT** and other mutex locks exist, the system waits for other locks to be released.

- **CANCELABLE**

  Allows the waiting thread to send CANCEL signals to the holding threads and waiting threads.

  Only the redistribution tool can use this parameter. An error message is displayed when the parameter is used by users.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
m_db=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       INTEGER
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', '18'),(5, 'AAAAAAAACAAAAAAA',
'362'),(7, 'AAAAAAAADAAAAAAA', '585');

m_db=# START TRANSACTION;

m_db=# LOCK TABLE tpcds.reason WRITE;

m_db=# COMMIT;

-- Drop the table.
m_db=# DROP TABLE tpcds.reason;

-- Drop the schema.
m_db=# DROP SCHEMA tpcds;
```

4.4.2.13.2 LOAD DATA

Description

Imports data from a file to a specified table in the database.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- LOAD DATA requires the INSERT and DELETE permissions on tables.
- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to execute LOAD DATA. When the **enable_copy_server_files** parameter is enabled, a user with the SYSADMIN permission or a user who inherits the permissions of the built-in role gs_role_copy_files can execute LOAD DATA. By default, you cannot execute LOAD DATA on database configuration files, key files, certificate files, or audit logs. This prevents unauthorized users from viewing or modifying sensitive files.
- When the **enable_copy_server_files** parameter is enabled, an administrator can use the GUC parameter **safe_data_path** to set the path where common users can import or export data. However, the path must be a subpath of **safe_data_path**. If the **safe_data_path** parameter is not set (by default), the path used by common users is not blocked.
- If the data written to a table by executing LOAD DATA cannot be converted to the data type of the table, the import fails.
- LOAD DATA applies only to tables but not views.

Syntax

```
LOAD DATA
   [LOCAL]
   INFILE 'file_name'
   [REPLACE | IGNORE]
   INTO TABLE tbl_name
   [PARTITION (partition_name [, partition_name] ...)]
   [CHARACTER SET charset_name]
   [{FIELDS | COLUMNS}
      [TERMINATED BY 'string']
      [[OPTIONALLY] ENCLOSED BY 'char']
      [ESCAPED BY 'char']
   ]
   [LINES
      [STARTING BY 'string']
      [TERMINATED BY 'string']
   ]
   [IGNORE number {LINES | ROWS}]
   [(col_name_or_user_var
      [, col_name_or_user_var] ...)]
   [SET col_name={expr | DEFAULT}
      [, col_name={expr | DEFAULT}] ...]
```

Parameters

- **LOCAL**

  Specifies the location of the file to be imported.

  If **LOCAL** is not specified and **'file_name'** is a relative path, data is imported to the data directory by default.

  If the **LOCAL** parameter is specified, **file_name** must be set to an absolute path. If **file_name** is set to a relative path, data is imported to the location of the database binary file by default, that is, *$GAUSSHOME*/**bin/**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If the imported data conflicts with the table data or the number of columns in the file is less than the number of columns in the specified table, the function of specifying **LOCAL** is the same as that of specifying **IGNORE**.
- Currently, LOAD DATA can only be used to import files from the server to the database.

- **REPLACE | IGNORE**

  If the data to be imported conflicts with the original data in the table, specifying **REPLACE** replaces the conflicting data, and specifying **IGNORE** skips the conflicting data and continues the import. If data conflicts occur but none of **REPLACE**, **IGNORE**, or **LOCAL** is specified, the import stops and an error is reported.

  NOTE

  If the number of columns in the file is less than that in the specified table, specifying **IGNORE** or **LOCAL** will assign default values to the remaining columns. If **IGNORE** or **LOCAL** is not specified, an error is reported.

- **PARTITION**

  If the table to be imported is a partitioned table, this parameter specifies a partition. If the data is inconsistent with the specified partition range, an error is reported.

- **CHARACTER SET**

  Specifies the encoding format of a data file. The default value is the current client encoding format.

- **FIELDS | COLUMNS**
  - **TERMINATED BY**

    Specifies the delimiter between columns. The default value is **\t**.

    NOTE

    The specified newline character cannot be the same as the delimiter.

  - **[OPTIONALLY] ENCLOSED BY**

    Specifies the quotation mark character. The default value is **''**.

    The **OPTIONALLY** parameter is optional and does not take effect.

    The quotation mark can only be a single character and cannot be a character string.

  - **ESCAPED BY**

    Specifies the escape character. The default value is **'\'**.

    The escape character can only be a single character and cannot be a character string.

- **LINES**
  - **STARTING BY**

    Specifies the style of the starting field in the data file to be imported.

  - **TERMINATED BY**

    Specifies the newline character style of the imported data file.

- **IGNORE**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies that the first *number* rows of the data file are skipped during data import.

- **col_name_or_user_var**

  Specifies an optional list of columns to be copied.

  Value range: any columns. All columns will be copied if no column list is specified.

  NOTE

  - The parameter for specifying columns cannot be used to specify a column repeatedly.
  - When columns are specified using the LOAD DATA syntax, **col_name_or_user_var** can be specified as existing columns or user variables.

- **SET**

  Specifies the column value, which can be an expression or **DEFAULT**.

  NOTE

  - The expression does not support column names.
  - If no implicit conversion exists between the expression result type and the type of the assigned column, an error is reported.

Examples

```
-- Prepare the files to be imported.
m_db=# CREATE TABLE load_data_tmp1(a int, b int);
m_db=# INSERT INTO load_data_tmp1 VALUES(1,1),(2,2),(3,3);
m_db=# \copy load_data_tmp1 to '/home/omm/load1.csv';
m_db=# CREATE TABLE load_data_tmp2(a int, b int);
m_db=# INSERT INTO load_data_tmp2 VALUES(1,2);
m_db=# \copy load_data_tmp2 to '/home/omm/load2.csv';
m_db=# CREATE TABLE load_data_tmp3(a int, b int);
m_db=# INSERT INTO load_data_tmp3 VALUES(4,4),(5,5);
m_db=# \copy load_data_tmp3 to '/home/omm/load3.csv';
m_db=# CREATE TABLE load_data_tmp4(a char(50));
m_db=# INSERT INTO load_data_tmp4 VALUES('"load test quote"'), ('\'load test single_quote\'');
m_db=# \copy load_data_tmp4 to '/home/omm/load4.csv';

-- Create a table.
m_db=# CREATE TABLE load_data_tbl1(load_col1 INT UNIQUE, load_col2 INT, load_col3 CHAR(10));

-- Insert a data record into the table.
m_db=# INSERT INTO load_data_tbl1 VALUES(0,0,'load0');

-- Copy data from the /home/omm/load1.csv file to the load_data_tbl table, specify a column name, and set the value of the load_col3 column to load.
m_db=# LOAD DATA INFILE '/home/omm/load1.csv' INTO TABLE load_data_tbl1(load_col1, load_col2) SET load_col3 = 'load';

-- The values imported to the load_col3 column later are all 'load'.
m_db=# SELECT * FROM load_data_tbl1 ORDER BY load_col1;
 load_col1 | load_col2 | load_col3
-----------+-----------+-----------
         0 |         0 | load0
         1 |         1 | load
         2 |         2 | load
         3 |         3 | load
(4 rows)

-- Copy data from the /home/omm/load2.csv file to the load_data_tbl table and specify IGNORE to ignore conflicts.
m_db=# LOAD DATA INFILE '/home/omm/load2.csv' IGNORE INTO TABLE load_data_tbl1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Data in the load_data_tbl1 table remains unchanged, and conflicting data is skipped.
m_db=# SELECT * FROM load_data_tbl1 ORDER BY load_col1;
 load_col1 | load_col2 | load_col3
-----------+-----------+-----------
         0 |         0 | load0
         1 |         1 | load
         2 |         2 | load
         3 |         3 | load
(4 rows)

-- Create a partitioned table.
m_db=# CREATE TABLE load_data_tbl2
(
    load_col_col1 INT,
    load_col_col2 INT
) PARTITION BY RANGE (load_col_col2)
(
    PARTITION load_p1 VALUES LESS THAN(3),
    PARTITION load_p2 VALUES LESS THAN(9),
    PARTITION load_p3 VALUES LESS THAN(MAXVALUE)
);

-- Copy data from the /home/omm/load3.csv file to the load_data_tbl2 table and specify a partition.
m_db=# LOAD DATA INFILE '/home/omm/load3.csv' INTO TABLE load_data_tbl2 PARTITION (load_p2);

-- Import data to the specified partition in the load_data_tbl2 table.
m_db=# SELECT * FROM load_data_tbl2;
 load_col_col1 | load_col_col2
---------------+---------------
             4 |             4
             5 |             5
(2 rows)

-- Create a table.
m_db=# CREATE TABLE load_data_tbl3(load_col_col1 CHAR(30));

-- Copy data from the /home/omm/load4.csv file to the load_data_tbl3 table and specify FIELDS
ENCLOSED BY.
m_db=# LOAD DATA INFILE '/home/omm/load4.csv' INTO TABLE load_data_tbl3 FIELDS ENCLOSED BY '"';

-- The double quotation marks ("") of "load test quote" are removed, and the single quotation marks of
'load test single_quote' are retained.
m_db=# select * from load_data_tbl3;
         load_col_col1
--------------------------------
 load test quote
 'load test single_quote'
(2 rows)

-- Drop the table.
m_db=# DROP TABLE load_data_tmp1;
m_db=# DROP TABLE load_data_tmp2;
m_db=# DROP TABLE load_data_tmp3;
m_db=# DROP TABLE load_data_tmp4;
m_db=# DROP TABLE load_data_tbl1;
m_db=# DROP TABLE load_data_tbl2;
m_db=# DROP TABLE load_data_tbl3;
```

4.4.2.14 P

4.4.2.14.1 PREPARE

Description

Creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When the PREPARE statement is executed, the specified query is parsed, analyzed, and rewritten. When EXECUTE is executed, the prepared statement is planned and executed. This avoids repetitive parsing and analysis. After the PREPARE statement is created, it exists throughout the database session. Once it is created (even if in a transaction block), it will not be deleted when a transaction is rolled back. It can only be deleted by explicitly calling **DEALLOCATE** or automatically deleted when the session ends.

Precautions

The name of a PREPARE statement must be unique.

Syntax

```
PREPARE name FROM string
```

Parameters

- **name**

  Specifies an identifier for the prepared query.

- **string**

  Specifies a character string, which is a prepared SQL statement. The user variable **@var_name** is not supported. The following statements are supported:

  – SELECT
  – INSERT
  – UPDATE
  – DELETE

Examples

```
m_db=# PREPARE p1 FROM 'SELECT 10 < 52';
PREPARE
m_db=# EXECUTE p1;
 ?column?
----------
 t
(1 row)
```

4.4.2.14.2 PREPARE TRANSACTION

Description

Prepares the current transaction for two-phase commit.

After this statement, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a high probability that it can be committed successfully, even if a database crash occurs before the commit is requested.

Once prepared, a transaction can later be committed or rolled back with **COMMIT PREPARED** or **ROLLBACK PREPARED**, respectively. Those statements can be issued from any session, not only the one that executed the original transaction.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

From the point of view of the issuing session, **PREPARE TRANSACTION** is not unlike a **ROLLBACK** statement: after executing it, there is no active current transaction, and the effects of the prepared transaction are no longer visible. (The effects will become visible again if the transaction is committed.)

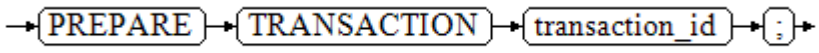
If the **PREPARE TRANSACTION** statement fails for any reason, it becomes a **ROLLBACK** and the current transaction is canceled.

Precautions

- The transaction function is maintained automatically by the database, and should be not visible to users.
- Currently, the distributed system does not allow users to call the user-defined PREPARE TRANSACTION operation. This syntax is reserved only for local two-phase cleanup on CNs in maintenance mode and does not ensure the global transaction capability.
- When running the **PREPARE TRANSACTION** statement, increase the value of **max_prepared_transactions** in configuration file **gaussdb.conf**. You are advised to set it to a value not less than that of **max_connections** so that one pending prepared transaction is available for each session.
- Currently, two-phase commit is not supported in distributed mode (not even in maintenance mode). This syntax is reserved only for local two-phase cleanup on CNs in maintenance mode and does not ensure the global transaction capability.

Syntax

```
PREPARE TRANSACTION transaction_id;
```



Parameters

- **transaction_id**

  Specifies an arbitrary identifier that later identifies this transaction for **COMMIT PREPARED** or **ROLLBACK PREPARED**. The identifier must be different from those for current prepared transactions.

  Value range: The identifier must be written as a string literal, and must be less than 200 bytes long.

Helpful Links

COMMIT PREPARED and **ROLLBACK PREPARED**

4.4.2.14.3 PURGE

Description

The PURGE statement can be used to:

- Clear tables or indexes from the recycle bin and release all space related to the objects.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Clear the recycle bin.

Precautions

- The PURGE operation supports tables (**PURGE TABLE**), indexes (**PURGE INDEX**), and recycle bins (**PURGE RECYCLEBIN**).
- The permission requirements for performing the PURGE operation are as follows:
  - **PURGE TABLE**: The user must be the owner of the table and must have the USAGE permission on the schema to which the table belongs. When the separation of duties is disabled, the system administrator has this permission by default.
  - **PURGE INDEX**: The user must be the owner of the index and must have the USAGE permission for the schema to which the index belongs. When the separation of duties is disabled, the system administrator has this permission by default.
  - **PURGE RECYCLEBIN**: Common users can clear only the objects owned by themselves in the recycle bin. Additionally, the user must have the USAGE permission of the schema to which the objects belong. When the separation of duties is disabled, the system administrator has this permission by default.

Prerequisites

- The **enable_recyclebin** parameter has been enabled to enable the recycle bin. Contact an administrator for details about how to use the parameter.
- The **recyclebin_retention_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires. Contact an administrator for details about how to use the parameter.

Syntax

```
PURGE { TABLE [schema_name.]table_name
      | INDEX index_name
      | RECYCLEBIN
    };
```

Parameters

- **schema_name**

  Specifies a schema name.

- **TABLE [schema_name.] table_name**

  Clears a specified table in the recycle bin. The table can be schema-qualified.

- **INDEX index_name**

  Clears a specified index in the recycle bin.

- **RECYCLEBIN**

  Clears the objects in the recycle bin.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Create the tpcds.reason_t1 table.
m_db=# CREATE TABLE tpcds.reason_t1
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) ;
-- Create the tpcds.reason_t2 table.
m_db=# CREATE TABLE tpcds.reason_t2
 (
  r_reason_sk    integer,
  r_reason_id    character(16),
  r_reason_desc  character(100)
  ) ;
-- Create an index on the tpcds.reason_t1 table.
m_db=# CREATE INDEX index_t1 on tpcds.reason_t1(r_reason_id);
m_db=# CREATE INDEX index_t2 on tpcds.reason_t2(r_reason_id);
m_db=# DROP TABLE tpcds.reason_t1;
m_db=# DROP TABLE tpcds.reason_t2;

-- Check the recycle bin.
m_db=# SELECT rcyname,rcyoriginname,rcytablespace FROM pg_catalog.gs_recyclebin;
           rcyname            | rcyoriginname | rcytablespace
------------------------------+---------------+---------------
 BIN$41512338419D$87EAEB8==$0 | reason_t1     |             0
 BIN$4151233841A3$87EBBE0==$0 | index_t1      |             0
 BIN$4151233841A0$87EC6A0==$0 | reason_t2     |             0
 BIN$4151233841A4$87ED0F0==$0 | index_t2      |             0
(4 rows)

-- Purge the table.
m_db=# PURGE TABLE tpcds.reason_t2;
m_db=# SELECT rcyname,rcyoriginname,rcytablespace FROM pg_catalog.gs_recyclebin;

-- Purge the index.
m_db=# PURGE INDEX tpcds.index_t1;
m_db=# SELECT rcyname,rcyoriginname,rcytablespace FROM pg_catalog.gs_recyclebin;
           rcyname            | rcyoriginname | rcytablespace
------------------------------+---------------+---------------
 BIN$41512338419D$87EAEB8==$0 | reason_t1     |             0
 BIN$4151233841A3$87EBBE0==$0 | index_t1      |             0
(2 rows)

-- Purge all objects
m_db=# PURGE recyclebin;
m_db=# SELECT rcyname,rcyoriginname,rcytablespace FROM pg_catalog.gs_recyclebin;
 rcyname | rcyoriginname | rcytablespace
---------+---------------+---------------
(0 rows)

-- Delete the schema.
 m_db=# DROP SCHEMA tpcds;
```

4.4.2.15 R

4.4.2.15.1 REINDEX

Description

Rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

There are several scenarios in which REINDEX can be used:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- An index has become corrupted, and no longer contains valid data.
- An index has become "bloated", that is, it contains many empty or nearly-empty pages.
- You have altered a storage parameter (such as a fill factor) for an index, and wish that the change takes full effect.

Precautions

- REINDEX DATABASE and REINDEX SYSTEM cannot be performed in transaction blocks.
- If the index contains the **lpi_parallel_method** option and the value is **PARTITION**, and the **parallel_workers** value of the index's table is greater than 0, the index cannot be rebuilt in parallel. If the index does not contain the **lpi_parallel_method** option or the value of the option is set to **AUTO**, page-level parallel index is rebuilt by default. For details, see **LPI_PARALLEL_METHOD**.

Syntax

- Rebuild an ordinary index.

```
REINDEX { INDEX |  TABLE | DATABASE} [CONCURRENTLY] name [ FORCE ];
REINDEX SYSTEM name [FORCE];
```

- Rebuild an index partition.

```
REINDEX { INDEX | TABLE }[CONCURRENTLY] name PARTITION partition_name [ FORCE ];
```

Parameters

- **INDEX**

  Rebuilds the specified index.

- **TABLE**

  Rebuilds all indexes of a specified table. The TOAST table (if any) of the table is reindexed as well. If an index on the table has been invalidated by running **alter unusable**, the index cannot be rebuilt. Indexes in the TOAST table cannot be rebuilt when specifying the **CONCURRENTLY** option.

- **DATABASE**

  Rebuilds all indexes within the current database. Indexes in the TOAST table within the current database cannot be rebuilt when specifying the CONCURRENTLY option.

- **SYSTEM**

  Rebuilds all indexes on system catalogs within the current database. Indexes on user tables are not processed.

- **CONCURRENTLY**

  Rebuilds an index (with ShareUpdateExclusiveLock) in non-blocking DML mode. When an index is rebuilt, other statements cannot access the table on which the index depends. If this keyword is specified, DML is not blocked during the recreation. Indexes in system catalogs cannot be rebuilt online. REINDEX INTERNAL TABLE CONCURRENTLY, REINDEX SYSTEM CONCURRENTLY, and REINDEX INVALID INDEX CONCURRENTLY are not supported. When REINDEX DATABASE CONCURRENTLY is executed, all indexes on user tables in the current database are rebuilt online (indexes on system catalogs are not processed). REINDEX CONCURRENTLY cannot be

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

executed within a transaction. Online index rebuilding supports only B-tree and UB-tree indexes, ordinary indexes, global indexes, and local indexes. Online index columns cannot be added, deleted, or modified. PCR UB-tree indexes, level-2 partitions, and GSIs are not supported. Online concurrent index rebuilding supports only ordinary indexes, global indexes, and local indexes of Astore and Ustore. Other online index rebuilding specifications are inherited from the current version. If online index rebuilding fails, the system automatically clears new indexes to prevent resource occupation in scenarios such as manual cancellation, duplicate unique index key values, insufficient resources, thread startup failure, and lock timeout. If the system cannot automatically clear invalid new indexes (for example, the database breaks down, FATAL, or PANIC), you need to manually clear invalid new indexes (using the DROP INDEX statement) and temporary tables (using the DROP TABLE statement) as soon as possible to prevent more resources from being occupied. Generally, the extension of an invalid index name is **_ccnew**. The execution of REINDEX INDEX CONCURRENTLY adds a four-level session lock to the table and its first several phases are similar to those of CREATE INDEX CONCURRENTLY. Therefore, the execution may be suspended or deadlocked, which is similar to that of CREATE INDEX CONCURRENTLY. For example, if two sessions perform the REINDEX CONCURRENTLY operation on the same index or table at the same time, a deadlock occurs.

NOTE

This keyword is specified when an index is rebuilt. For Astore, you need to complete two full table scans for building. During the first scan, a new index is created without blocking read and write operations. During the second scan, the changes in the first scan are merged and updated. For Ustore, you need to complete a full table scan. During the scan, data generated by concurrent DML operations is inserted into the temporary table named **index_oid_cctmp**. After the scan is complete, you merge the temporary table to the new index suffixed with **_ccnew{n}**, delete the temporary table, exchange the old and new indexes, mark the old index as dead, enable the new index, and rebuild the index.

- **name**

  Specifies the name of the index, table, or database whose index needs to be rebuilt. Tables and indexes can be schema-qualified.

  NOTE

  REINDEX DATABASE and SYSTEM can create indexes for only the current database. Therefore, **name** must be the same as the current database name.

- **FORCE**

  Specifies an invalid option, which will be ignored.

- **partition_name**

  Specifies the name of the partition or index partition to be rebuilt.

  Value range:

  – If REINDEX INDEX is used, specify the name of an index partition.
  – If REINDEX TABLE is used, specify the name of a partition.

NOTICE

REINDEX DATABASE and REINDEX SYSTEM cannot be performed in transaction blocks.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create the tpcds. customer table.
m_db=# CREATE TABLE tpcds.customer
(
c_customer_sk         INTEGER        NOT NULL,
c_customer_id         CHAR(16)       NOT NULL
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.customer VALUES (1, 'AAAAAAAABAAAAAAA'),(5, 'AAAAAAAACAAAAAAA'),(10,
'AAAAAAAADAAAAAAA');

-- Create a row-store table tpcds.customer_t1 and create an index on the c_customer_sk column in the
table.
m_db=# CREATE TABLE tpcds.customer_t1
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)
)
WITH (orientation = row);

m_db=# CREATE INDEX tpcds_customer_index1 ON tpcds.customer_t1 (c_customer_sk);

m_db=# INSERT INTO tpcds.customer_t1 SELECT * FROM tpcds.customer WHERE c_customer_sk < 10;

-- Rebuild a single index.
m_db=# REINDEX INDEX tpcds.tpcds_customer_index1;

-- Rebuild all indexes in the tpcds.customer_t1 table:
m_db=# REINDEX TABLE tpcds.customer_t1;

Delete the tpcds.customer_t1 table.
m_db=# DROP TABLE tpcds.customer_t1;

-- Drop the table.
m_db=# DROP TABLE tpcds.customer;

-- Delete the schema.
m_db=# DROP SCHEMA tpcds;
```

Suggestions

- INTERNAL TABLE

  This scenario is used for fault recovery. You are advised not to perform concurrent operations.
- DATABASE

  You are not allowed to reindex databases in transactions.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- SYSTEM

  You are not allowed to reindex system catalogs in transactions.

4.4.2.15.2 RELEASE SAVEPOINT

Description

Deletes a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of statements executed after the savepoint was established. To do that, use **ROLLBACK TO SAVEPOINT**. Destroying a savepoint when it is no longer needed allows the system to recycle some resources earlier than transaction end.

RELEASE SAVEPOINT also deletes all savepoints that were established after the named savepoint was established.

Precautions

- Releasing a savepoint name that was not previously defined will cause an error.
- It is not possible to release a savepoint when the transaction is in an aborted state.
- If multiple savepoints have the same name, only the most recently defined savepoint is released.

Syntax

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

Parameters

- **savepoint_name**

  Specifies the name of the savepoint to be destroyed.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table.
m_db=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
m_db=# START TRANSACTION;

--Insert data.
m_db=# INSERT INTO tpcds.table1 VALUES (3);

-- Create a savepoint.
m_db=# SAVEPOINT my_savepoint;

--Insert data.
m_db=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
m_db=# RELEASE SAVEPOINT my_savepoint;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Commit the transaction.
m_db=# COMMIT;

-- Query the table content. The table should contain both 3 and 4.
m_db=# SELECT * FROM tpcds.table1;

-- Drop the table.
m_db=# DROP TABLE tpcds.table1;

-- Delete the schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

SAVEPOINT and **ROLLBACK TO SAVEPOINT**

4.4.2.15.3 REPLACE

Description

Inserts data into a table or replaces existing data in a table. If the data to be inserted has the primary key or unique key conflicts with the existing data, the REPLACE statement deletes the existing data and then inserts the new data.

You can use REPLACE in the following ways:

- Replace or insert values. That is, VALUES or VALUE is used to construct a row of records and insert the row into the table.
- Replace or insert the query. One or more rows of records are constructed based on the result set returned by SELECT and inserted into a table.
- Set the value of a specified column. Similar to value insertion, the default value is used for columns that are not specified.

Precautions

- To execute this statement, you must have the DELETE and INSERT permissions on the table.
- If the primary key or unique key does not conflict, you can directly insert the data. If the primary key or unique key conflicts, delete the original data and then insert the new data.
- The return format of the REPLACE operation is **REPLACE 0** *X*, where *X* indicates the number of DELETE and INSERT operations.
- In the REPLACE...SELECT syntax, the number of columns in **select_list** must be the same as the number of columns to be inserted.
- In the REPLACE...SET syntax:
  - If **col_name** has a default value, **SET col_name = col_name + 1** is equivalent to **SET col_name =** *Default value of col_name* **+ 1**.
  - If **col_name** has neither default value nor NOT NULL constraint, **SET col_name = col_name + 1** is equivalent to **col_name = NULL**.
  - If **col_name** does not have a default value but has a NOT NULL constraint, the default zero value of each data type is displayed as follows:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Type	Default Zero Value
tinyint [unsigned | signed], smallint [unsigned | signed], mediumint [unsigned | signed], int [unsigned | signed], bigint [unsigned | signed], float4, float8, numeric, and bool	**0**. If **numeric** specifies a decimal place, the decimal place is displayed. For example, **NUMERIC (10, 3): 0.000**
char, varchar, tinytext, text, mediumtext, longtext, tinyblob, blob, mediumblob, longblob, bit, and varbinary	''
binary	''. The length of an empty string is specified. If the length is not specified, the length is **1**.
time	00:00:00.
date	0000-00-00.
timestamp	0000-00-00 00:00:00.
datetime	0000-00-00 00:00:00.
year	0000.

NOTE

- The zero value can be inserted only when date, timestamp, or datetime is in loose mode.
- If the default value of the function expression can be calculated during parsing, the default value is the calculated constant. Otherwise, the value is **NULL**.
- In other scenarios, the default value is **NULL**.

- REPLACE ... In the SET syntax, the current **col_name** value depends on the previous **col_name** value. If there is no previous **col_name** value, the default value is used. For example, in the **SET f1 = f1 + 1, f2 = f1** scenario, **f1** is equal to the default value of **f1** (assumed to **0**) plus 1, and **f2** is equal to the calculated value of **f1**, that is **1**.
- If a table has multiple unique constraints and the inserted data violates multiple unique constraints, all the data that violates the constraints is deleted and new data is inserted. In this scenario, data may be deleted by mistake. Therefore, exercise caution when performing this operation.
- If multiple rows are inserted and these rows have unique constraint conflicts with data in the same row in the table, the REPLACE operation is performed in sequence.
- In **SET col_name = col_name + 1**, the length of **col_name** cannot exceed 1. For example, in **SET col_name = table_name.col_name + 1**, the length of **table_name.col_name** is 2. Formats such as B.A, C.B.A, and D.C.B.A are not supported.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When comparing floating-point data, note that precision loss may occur.

Syntax

- Replace or insert values.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    [ ( col_name [, ... ] ) ]
    { VALUES | VALUE } ( value [, ... ] ) [, ... ]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];

-- The format of value is as follows:
{ expression | DEFAULT }
```

- Replace or insert the query.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    [ ( col_name [, ... ]) ]
    query
  [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

- Set the value of a specified column.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    SET col_name = value [, ... ];

-- The format of value is as follows:
{ expression | DEFAULT }
```

Parameters

- **table_name**

  Specifies the name of the target table where data will be inserted.

  Value range: an existing table name.

- **col_name**

  Specifies the name of a column in a table.

  - The column name can be qualified with a subcolumn name or array index, if needed.
  - Each column not present in the column list will be filled with a default value, either its declared default value or **NULL** if there is none. Inserting data into only some columns of a composite type leaves the other columns **NULL**.
  - The target column names (specified by **col_name**) can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.
  - The target columns are the first *N* column names, if there are only *N* columns provided by the VALUE clause and QUERY.
  - The values provided by the VALUE clause and QUERY are joined with the corresponding columns from left to right in the table.

  Value range: an existing column.

- **PARTITION (partition_name [, ...])**

  Inserts data to a specified partition. **partition_name** indicates the partition name.

  If the value of the VALUE clause is inconsistent with that of the specified partition, an exception is displayed.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **value**

  Specifies the value to be inserted. The value format is as follows:

  ```
  { expression | DEFAULT }
  ```

  a. **expression** indicates that a valid expression or value is assigned to the corresponding column.

     If single-quotation marks (') are inserted into a column, the single-quotation marks (') need to be used for escape.

     If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

  b. **DEFAULT** indicates the default value of the corresponding column name. The value is **NULL** if there is no default value.

- **query**

  Specifies a query statement (SELECT statement) that uses the query result as the inserted data.

- **RETURNING**

  Returns the inserted rows. The syntax of the RETURNING list is identical to that of the output list of SELECT.

Examples

```
-- Create a table.
m_db=# CREATE TABLE test(f1 int primary key, f2 int, f3 int);

-- Insert data.
m_db=# INSERT INTO test VALUES(1, 1, 1), (2, 2, 2), (3, 3, 3);
INSERT 0 3

-- Replace or insert values.
m_db=# REPLACE INTO test VALUES(1, 11, 11);
REPLACE 0 2

-- Query the result.
m_db=# SELECT * FROM test WHERE f1 = 1;
 f1 | f2 | f3
----+----+----
  1 | 11 | 11
(1 row)

-- Replace or insert the query.
m_db=# REPLACE INTO test SELECT 2, 22, 22;
REPLACE 0 2

-- Query the result.
m_db=# SELECT * FROM test WHERE f1 = 2;
 f1 | f2 | f3
----+----+----
  2 | 22 | 22
(1 row)

-- Replace or insert the specified column.
m_db=# REPLACE INTO test SET f1 = f1 + 3, f2 = f1 * 10 + 3, f3 = f2;
REPLACE 0 2

-- Query the result.
m_db=# SELECT * FROM test WHERE f1 = 3;
 f1 | f2 | f3
----+----+----
  3 | 33 | 33
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
m_db=# DROP TABLE test;
```

4.4.2.15.4 RESET

Description

Restores GUC parameters to their default values. The default values are parameter default values complied in the **gaussdb.conf** configuration file.

The current version does not support **RESET configuration_parameter**. Only **RESET ALL** is supported.

Precautions

- **RESET ALL** and **SET** have the same transaction behavior. Their impact will be rolled back.
- **RESET ALL** cannot be used to restore the default values of **role** and **session authorization**.

Syntax

```
RESET ALL;
```

Parameters

- **ALL**

  Specifies all GUC parameters.

Examples

```
-- Set all parameters to their default values.
m_db=# RESET ALL;
```

Helpful Links

SET and **SHOW**

4.4.2.15.5 REVOKE

Description

Revokes permissions from one or more roles.

Precautions

If a non-owner user of an object attempts to REVOKE permission on the object, the statement is executed based on the following rules:

- If the user has no permissions whatsoever on the object, the statement will fail outright.
- If an authorized user has some permissions, only the permissions with authorization options are revoked.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If the authorized user does not have the authorization option, the REVOKE ALL PRIVILEGES form will issue an error message. For other forms of statements, if the permission specified in the statement does not have the corresponding authorization option, the statement will issue a warning.

Syntax

- Revoke the permission on a specified table or view.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified column in a table.

```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified sequence.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ SEQUENCE ] sequence_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified database.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

  NOTE

  This syntax takes effect when the GUC parameter **m_format_behavior_compat_options** is set to **grant_database_nomapping**. It revokes the access permission of a specified user or role on database-level objects. If the **grant_database_nomapping** option is not set, REVOKE ON DATABASE is mapped to REVOKE ON SCHEMA, revoking the access permission of a specified user or role on schema-level objects. REVOKE ON DATABASE supports only the syntax options supported by REVOKE ON SCHEMA.

- Revoke the permission on a specified schema.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAPRIVILEGEGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified subcluster.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE | COMPUTE  | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON NODE GROUP group_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> When the CREATE permission on a sub-cluster is revoked, the USAGE and COMPUTE permissions are revoked by default.

- Revoke permissions from a role.

```
REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_name [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the SYSADMIN permission from a role.

```
REVOKE ALL PRIVILEGES FROM role_name;
```

- Revoke the ANY permissions.

```
REVOKE [ ADMIN OPTION FOR ]
  { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX |
ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
  SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX
} [, ...]
  FROM [ GROUP ] role_name [, ...];
```

Parameters

The keyword PUBLIC indicates an implicitly defined group that has all roles.

For details about permission types and parameters, see **Parameters** in section "GRANT."

Permissions of a role include the permissions directly granted to the role, permissions inherited from the parent role, and permissions granted to PUBLIC. Therefore, revoking the SELECT permission on an object from PUBLIC users does not necessarily mean that the SELECT permission on the object has been revoked from all roles, because the SELECT permission directly granted to roles and inherited from parent roles remains. Similarly, if the SELECT permission is revoked from a user but is not revoked from PUBLIC users, the user can still run the SELECT statement.

If GRANT OPTION FOR is specified, the permission cannot be granted to others, but permission itself is not revoked.

If user A holds the UPDATE permission on a table and the **WITH GRANT OPTION** option and has granted them to user B, the permission that user B holds is called dependent permission. When user A's permission or grant option is revoked, CASCADE must be declared to revoke all dependent permissions.

A user can only revoke permissions that were granted directly by that user. For example, if user A has granted permission with grant option (**WITH ADMIN OPTION**) to user B, and user B has in turn granted it to user C, then user A cannot revoke the permission directly from C. However, user A can revoke the grant option held by user B and use CASCADE. In this way, the permission of user C is automatically revoked. For another example, if both user A and user B have granted the same permission to C, A can revoke A's own grant but not B's grant, so C will still effectively have the permission.

If the role executing REVOKE holds permissions indirectly by using more than one role membership path, it is unspecified which containing role will be used to execute the statement. In such cases, you are advised to use SET ROLE to become the specific role, and then execute REVOKE. Failure to do so may lead to deleting permissions not intended to delete, or not deleting any permissions at all.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

See **Examples** in section "GRANT."

Helpful Links

GRANT

4.4.2.15.6 ROLLBACK

Description

Rolls back the current transaction and backs out all updates in the transaction.

If a fault occurs during the running of a transaction, the transaction cannot be executed. The system cancels all completed operations on the database in the transaction, and the database status returns to the time when the transaction starts.

Precautions

If a ROLLBACK statement is executed out of a transaction, no error occurs, but a notice is displayed.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ];
```

Parameters

- **WORK | TRANSACTION**

  Specifies the optional keyword that more clearly illustrates the syntax.

Examples

```
-- Start a transaction.
m_db=# START TRANSACTION;

-- Roll back all changes.
m_db=# ROLLBACK;
```

Helpful Links

COMMIT

4.4.2.15.7 ROLLBACK PREPARED

Description

Cancels a transaction ready for two-phase committing.

Precautions

- The function is only available in maintenance mode (when the GUC parameter **xc_maintenance_mode** is set to **on**). Exercise caution when

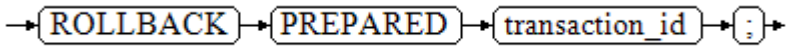
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.

- Only the user who initiates a transaction or a system administrator can roll back the transaction.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
ROLLBACK PREPARED transaction_id;
```



NOTE

The distributed environment does not support the ROLLBACK PREPARED syntax.

Parameters

transaction_id

Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

Helpful Links

COMMIT PREPARED and **PREPARE TRANSACTION**

4.4.2.15.8 ROLLBACK TO SAVEPOINT

Description

Rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Rolls back all statements that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

Precautions

- Rolling back to a savepoint name that has not been specified is an error.
- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the commands executed after the savepoint was established.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

Parameters

- **savepoint_name**

  Specifies the savepoint to roll back to.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Undo the effects of the statements executed after my_savepoint was established:
m_db=# START TRANSACTION;
m_db=# SAVEPOINT my_savepoint;
m_db=# ROLLBACK TO SAVEPOINT my_savepoint;
m_db=# RELEASE SAVEPOINT my_savepoint;
m_db=# COMMIT;
```

Helpful Links

SAVEPOINT and **RELEASE SAVEPOINT**

4.4.2.16 S

4.4.2.16.1 SAVEPOINT

Description

SAVEPOINT establishes a new savepoint in the current transaction.

A savepoint is a special mark in a transaction. It allows all commands that are executed after the savepoint was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

Precautions

- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the commands executed after the savepoint was established.
- Savepoints can only be established when inside a transaction block. Multiple savepoints can be defined within a transaction.
- In the case of an unexpected termination of a thread or process caused by a node or connection failure, or of an error caused by the inconsistency between source and destination table structures in a COPY FROM operation, the transaction cannot be rolled back to the established savepoint. Instead, the entire transaction will be rolled back.
- According to the SQL standard, when a savepoint with the same name is created, the previous savepoint with the same name is automatically deleted. In an M-compatible database, the old savepoint is retained, but only the latest one is used during rollback or release. If the latest savepoint is released, the previous savepoint will again become accessible to ROLLBACK TO SAVEPOINT and RELEASE SAVEPOINT. Except for this, SAVEPOINT fully complies with the SQL standard.

Syntax

```
SAVEPOINT savepoint_name;
```

Parameter

- **savepoint_name**

  Specifies the name of the new savepoint.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples

```
-- Create a table.
m_db=# CREATE TABLE table1(a int);

-- Start a transaction.
m_db=# START TRANSACTION;

-- Insert data.
m_db=# INSERT INTO table1 VALUES (1);

-- Create a savepoint.
m_db=# SAVEPOINT my_savepoint;

-- Insert data.
m_db=# INSERT INTO table1 VALUES (2);

-- Roll back to the savepoint.
m_db=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Insert data.
m_db=# INSERT INTO table1 VALUES (3);

-- Commit the transaction.
m_db=# COMMIT;

-- Query the table content. The table should contain both 1 and 3 but not 2, because 2 is rolled back.
m_db=# SELECT * FROM table1;

-- Drop the table.
m_db=# DROP TABLE table1;

-- Create a table.
m_db=# CREATE TABLE table2(a int);

-- Start a transaction.
m_db=# START TRANSACTION;

-- Insert data.
m_db=# INSERT INTO table2 VALUES (3);

-- Create a savepoint.
m_db=# SAVEPOINT my_savepoint;

-- Insert data.
m_db=# INSERT INTO table2 VALUES (4);

-- Delete the savepoint.
m_db=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
m_db=# COMMIT;

-- Query the table content. The table should contain both 3 and 4.
m_db=# SELECT * FROM table2;

-- Drop the table.
m_db=# DROP TABLE table2;
```

Helpful Links

RELEASE SAVEPOINT and **ROLLBACK TO SAVEPOINT**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.16.2 SELECT

Description

SELECT retrieves data from tables or views.

Serving as an overlaid filter on database tables, the SELECT statement uses SQL keywords to filter data tables and extract the required data.

Precautions

- The owner of the table, users granted the SELECT permission on the table, and users granted the SELECT ANY TABLE permission can read data in the table or view. System administrators have the permission to read data in the table or view by default when separation of duties is disabled.
- SELECT can join ordinary tables, but cannot join ordinary and GDS foreign tables. That is, the SELECT statement cannot contain both an ordinary table and a GDS foreign table.
- You must have the SELECT permission on all columns in the SELECT statement.
- The UPDATE permission is required for the use of FOR UPDATE or FOR SHARE.

Syntax

Query data.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */]
    [ALL | DISTINCT | DISTINCTROW]
    [SQL_NO_CACHE]
    select_expr [, select_expr] ...
    [FROM from_item [,...]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}]
    [HAVING condition]
    [ORDER BY {col_name | expression | position}
      [ASC | DESC][ NULLS { FIRST | LAST } ], ...]
    [[LIMIT {[offset,] row_count | row_count OFFSET offset}] | FETCH {FIRST | NEXT} [(expr)] {ROW| ROWS}
ONLY]
    [into_option]
    [ FOR READ ONLY |
        [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT N | SKIP LOCKED ]} [...] ] |
        LOCK IN SHARE MODE]

select_expr:
    expression [ [AS] output_name ]

where_condition:
expr
```

SELECT ... UNION:

```
SELECT ...
    UNION [ALL | DISTINCT] SELECT ...
    [UNION [ALL | DISTINCT] SELECT ...]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

Restrictions on using aliases in the SELECT clause in the WHERE, HAVING, and GROUP BY clauses are as follows:

- Table creation statements are initialized.
  ```
  m_db=# DROP TABLE IF EXISTS t0,t1;
  m_db=# CREATE TABLE t0 (c0 INT, c1 INT);
  m_db=# CREATE TABLE t1 (c0 INT, c1 INT);
  ```
- Reference only within the same level.
  ```
  m_db=# SELECT r FROM t0 WHERE (select c0 as r from t0) = 1; -- error
  ```
- Reference only the aliases in the target list.
  ```
  m_db=# SELECT * FROM t0 WHERE (SELECT c0 AS r FROM t0) = 1 AND r = 2; -- error
  ```
- Reference a prior expression in a subsequent expression.
  ```
  m_db=# SELECT r AS d, c0 AS r FROM t0; -- error
  ```
- The **volatile** function cannot be used.
  ```
  m_db=# SELECT uuid() AS r FROM t0 WHERE r = 1; -- error
  ```
- The **Window** function cannot be used.
  ```
  m_db=# SELECT pg_catalog.rank() over(PARTITION BY c0 ORDER BY c1) AS r FROM t0 WHERE r = 1; -- error
  ```
- Aliases cannot be referenced in the condition of JOIN ON.
  ```
  m_db=# SELECT uuid() AS r FROM t0 JOIN t1 ON r = 1; -- error
  ```
- An error is reported if the target list contains multiple aliases to be referenced.
  ```
  m_db=# SELECT c0 AS r AS u AS v FROM t0; -- error
  ```

- The subquery **with_query** is as follows:
  ```
  with_query_name [ ( column_name [, ...] ) ]
      AS ( {select} )
  ```
- The specified query source **from_item** is as follows:
  ```
  { table_name  [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
  |with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
  |function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
  |joined_table
  ```

  NOTE

  table_name contains the virtual table **dual**. An M-compatible database allows you to specify **dual** as the virtual table name when no table is referenced.

- **joined_table** is as follows:
  ```
  joined_table: {
    table_reference [INNER | CROSS] JOIN table_factor [join_specification]
   | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_specification
   | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor
  }
  join_specification: {
     ON join_condition
   | USING (join_column_list)
  }
  join_column_list:
      column_name [, column_name] ...
  ```
- The GROUP clause is as follows:
  ```
  | expression
  | ( expression [, ...] )
  ```
- The specified partition **partition_clause** in **from_item** is as follows:
  ```
  PARTITION { ( { partition_name | subpartition_name } [, ...] ) | FOR (  partition_value [, ...] ) }
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- Specifying partitions applies only to partitioned tables.
- If PARTITION specifies multiple partition names, the partition names can be the same. The union set of the partition ranges is used.

Parameters

- **plan_hint**

  Follows the SELECT keyword in the **/*+ */** format. It is used to optimize the plan of a SELECT statement block based on hints. In each statement, only the first **/*+ plan_hint */** comment block takes effect as a hint. Multiple hints can be written.

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which are equivalent to temporary tables. This subquery statement structure is called the common table expression (CTE) structure. When this structure is used, the execution plan contains the CTE SCAN content.

  If RECURSIVE is specified, it allows a SELECT subquery to reference itself by name.

  The detailed format for **with_query** is as follows:

  ```
  with_query_name [ ( column_name [, ...] ) ] AS ( {select} )
  ```

  - **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.
  - **column_name** specifies the column names displayed in the subquery result set.
  - You can use SELECT in the subqueries.
  - RECURSIVE can appear only after WITH. If there are multiple CTEs, you only need to specify RECURSIVE at the first CTE.
  - When using RECURSIVE, the data types of the subquery results on the left and right of UNION ALL or UNION [DISTINCT] in the CTE subquery must be converted to the same data type using the CAST function. For example, **WITH RECURSIVE cte (n) AS (SELECT cast(id as signed int) from table_1 UNION ALL SELECT cast((n + 1) as signed int) FROM cte WHERE n < 5) SELECT * FROM cte**. For details about the type conversion generated by operators, see **Specification Constraints of Logical Operators**, **Specification Constraints of Bitwise Operators**, and **Specification Constraints of Arithmetic Operators**.

- **ALL**

  Specifies that all rows that meet the conditions are returned. The keyword can be omitted because all rows that meet the conditions are returned by default.

- **DISTINCT | DISTINCTROW**

  Removes all duplicate rows from the result set of the SELECT statement so that each row in the result set is unique.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

The **DISTINCT|DISTINCTROW** expression is interpreted using the same rules as ORDER BY. Unless you use ORDER BY to guarantee that the required row appears first, you cannot know what the first row is.

- **SELECT list**

  Specifies the column names in the queried table. It can be the names of some or all of the columns. To specify all columns, use the wildcard (*).

  You may use the AS *output_name* clause to assign an alias to the output column. The alias is usually used for displaying the output column. The keywords NAME, VALUE, and TYPE can be used as column aliases.

  Column names can be:

  – Manually input column names, where multiple names are separated by commas (,).

  – Columns computed by the FROM clause.

- **FROM clause**

  Specifies one or more source tables for SELECT.

  The FROM clause contains the following elements:

  – table_name

    Specifies the name of a table or view. The schema name can be added before the table name or view name, for example, **schema_name.table_name**.

  – alias

    Assigns a temporary alias to a table or a complex table reference so that the reference of the table by other queries is easier.

    An alias is used for abbreviation or eliminating ambiguity in self-joins. If an alias is assigned, it completely hides the actual name of the table.

  – column_alias

    Specifies the column alias.

  – PARTITION

    Queries data in the specified partition in a partitioned table.

  – partition_name

    Specifies the name of the partition.

  – partition_value

    Specifies the partition key value. If a partitioned table has multiple partition keys, you can use PARTITION FOR to uniquely identify a partition.

  – subquery

    Performs a subquery in the FROM clause. A temporary table is created to save the output of the subquery.

  – with_query_name

    The WITH clause can also be the source of the FROM clause and can be referenced by the name of the WITH query.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- function_name

  Specifies the function name. Functions can be used in a FROM clause.

- The types of JOIN are as follows:

  - [INNER] JOIN

    A JOIN clause combines two FROM items. You can use parentheses to determine the order of nesting. In the absence of parentheses, JOIN nests left-to-right.

    In any case, JOIN binds more tightly than FROM items that are separated by commas.

  - LEFT [OUTER] JOIN

    Returns all rows that meet the join conditions in the Cartesian product, as well as the rows in the left table that do not match the right table rows according to join conditions. Therefore, the rows on the left represent the full length of the joined table, where the corresponding columns in the right table are filled with **NULL**. Note that the condition of the JOIN clause is used only when matches are being calculated. The condition of the outer layer is applied after the calculation is complete.

  - RIGHT [OUTER] JOIN

    Returns all result rows of INNER JOIN, as well as all rows on the right that do not have matches (rows on the left are filled with **NULL**).

    This is just a notational convenience because you can always convert it to a LEFT OUTER JOIN by switching the left and right inputs.

  - CROSS JOIN

    CROSS JOIN is equivalent to INNER JOIN ON(TRUE), that is, rows that are not deleted by the conditions. This type of JOIN has the same effect as FROM and WHERE, and is more convenient in symbol usage.

    > **NOTE**
    >
    > You must specify a join condition for INNER JOIN and OUTER JOIN, that is, one of the following: **NATURAL ON**, **join_condition**, and **USING (join_column [, ...])**.

  CROSS JOIN and INNER JOIN produce a simple Cartesian product, the same result as you get from listing the two items at the top level of FROM.

  You can use multiple JOIN clauses consecutively.

- ON join_condition

  Specifies the join condition, defining which rows match. Example: **ON left_table.a = right_table.a** You are advised not to use numeric types such as int for **join_condition**, because such types can be implicitly converted to Boolean values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- USING(join_column[, ...])

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Abbreviation for **ON left_table.a = right_table.a AND left_table.b = right_table.b ...** Corresponding columns must have the same name.

- NATURAL

  A shorthand for a USING list that contains all columns in the two tables that have the same names.

- from item

  Specifies the name of the query source object used for the join.

- **WHERE clause**

  The WHERE clause forms an expression for row selection to narrow down the query range of SELECT. **condition** indicates any expression that returns a value of Boolean type. Rows that do not meet this condition will not be retrieved. You are advised not to use numeric types such as int for **condition**, because such types can be implicitly converted to Boolean values (non-zero numeric values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

  NOTICE

  When using a LIKE operator in a WHERE clause, add a slash (\) before each special character if any of the following is queried in LIKE: %_\

- **GROUP BY clause**

  Groups the query results based on the values of one or more columns. The query results with the same value are in the same group.

  NOTICE

  If a SELECT list expression references ungrouped columns and no aggregate function is used, an error is displayed. This is because multiple values may be returned for ungrouped columns.

- **HAVING clause**

  Used together with the GROUP BY clause to select special groups. The HAVING clause compares some attributes of groups with a constant. Only groups that match the logical expression in the HAVING clause are extracted. HAVING can only reference columns in the GROUP BY clause or columns used in aggregate functions.

- **UNION clause**

  Computes the set union of the rows returned by multiple SELECT statements.

  The UNION clause has the following constraints:

  - The default result of UNION does not contain any duplicate rows unless the ALL clause is specified.
  - UNION operators in the same SELECT statement are evaluated from left to right unless specified by parentheses.
  - FOR UPDATE cannot be specified either for a UNION result or for any input of a UNION.

  General expression:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
select_statement UNION [ALL] select_statement
```

- **select_statement** can be any **SELECT** statement without an **ORDER BY**, **LIMIT**, or **FOR UPDATE** statement.
- ORDER BY and LIMIT can be attached to subexpressions if enclosed in parentheses.

- **ORDER BY clause**

  Sorts data retrieved by SELECT in ascending or descending order. If the ORDER BY expression contains multiple columns:

  - The values in the leftmost column are sorted first. If the values in this column are the same, they are then compared based on the next expression until the comparison is complete.
  - If the values are equal for all specified expressions, they are returned in a random order.
  - When used with the DISTINCT keyword, the columns to be sorted in ORDER BY must be included in the columns of the result set retrieved by the SELECT statement.
  - The columns to be sorted by the ORDER BY clause where the GROUP BY clause is also used must be included in the columns grouped by GROUP BY.

  If the SELECT result set contains aggregate functions, the ORDER BY clause, not used together with GROUP BY, takes effect only for the columns where the functions that return the set are located.

  ```
  m_db=# CREATE TABLE t1(c1 int, c2 int);
  m_db=# EXPLAIN SELECT sum(c1) c FROM t1 ORDER BY c, c2; -- Ignores ORDER BY.
  m_db=# EXPLAIN SELECT sum(c1) c, generate_series (1,5) s FROM t1 ORDER BY c, c2; -- Ignores ORDER BY.
  The m_db=# EXPLAIN SELECT sum(c1) c, generate_series (1,5) s FROM t1 ORDER BY c, s; -- Sorts the s column in the result.
  ```

  - **NULLS FIRST**

    Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

  - **NULLS LAST**

    Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

  NOTICE

  To support Chinese Pinyin sorting, you need to specify the gb18030 and GBK character sets when initializing the M-compatible database. For details, see **Database-level Character Sets and Collations**. In addition, you need to specify the gb18030_chinese_ci and gbk_chinese_ci collations during comparison.

- **LIMIT clause**

  Consists of two independent clauses:

  LIMIT { count | ALL } limits the number of rows to be returned. **count** specifies the number of rows to be returned. The effect of **LIMIT ALL** is the same as that of omitting the LIMIT clause.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

LIMIT start count specifies the maximum number of rows to be returned, while **start** specifies the number of rows to skip before starting to return rows. When both are specified, **start** rows are skipped before **count** rows to be returned start to be counted. ROWNUM cannot be used as **count** or **offset** in the LIMIT clause.

- **FETCH {FIRST | NEXT} [(expr)] {ROW| ROWS} ONLY**

  If **count** is omitted in a **FETCH** clause, it defaults to **1**, indicating that the clause specifies the total number of rows counting from the first row of the returned query result.

  NOTE

  This attribute is not supported when the version compatibility control parameter **m_format_dev_version** of an M-compatible database is set to **'s1'** or later.

- **OFFSET clause**

  SQL: A different syntax was proposed in 2008.

  OFFSET start { ROW | ROWS }

  start specifies the number of rows to skip before starting to return rows.

- **Locking clause**

  The FOR UPDATE clause locks the rows retrieved by SELECT. This ensures that the rows cannot be modified or deleted by other transactions until the current transaction ends, that is, other transactions that attempt to perform UPDATE, DELETE, SELECT FOR UPDATE, or SELECT FOR SHARE on these rows will be blocked until the current transaction ends. Any DELETE on the same row will also acquire the FOR UPDATE lock mode, as will UPDATE that modifies values on the primary key column. Conversely, SELECT FOR UPDATE will wait for concurrent transactions that are running the preceding commands on the same row, and will continue to lock and return the updated rows (there may be no row because the rows may be deleted).

  FOR SHARE behaves similarly, except that it requires a shared lock rather than an exclusive lock on each retrieved row. A shared lock prevents other transactions from performing UPDATE, DELETE, or SELECT FOR UPDATE on the rows, but does not prevent them from performing SELECT FOR SHARE.

  To prevent the operation from waiting for the commit of other transactions, you can use **NOWAIT**. If the selected row cannot be locked immediately, an error is reported immediately. If you use **WAIT n** and the selected row cannot be locked immediately, the operation needs to wait for *n* seconds (the value of *n* is an integer ranging from 0 to 2147483). If the lock is obtained within *n* seconds, the operation is performed normally. Otherwise, an error is reported. If you use **SKIP LOCKED**, locked rows are skipped when a table is locked. Only rows that are locked using a row lock can be skipped. In scenarios where locks do not block each other in different transactions, for example, SELECT FOR SHARE - SELECT FOR SHARE, rows that are locked with preceding locks are not skipped though **SKIP LOCKED** is specified.

  If table names are specified in a locking clause, then only those tables will be locked. Other tables used in SELECT will not be locked. Otherwise, all tables used in the command will be locked.

  If a locking clause is applied to a view or subquery, it will lock all tables used in the view or subquery.

  Multiple locking clauses can be used to specify different lock modes for different tables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If a table appears (or implicitly appears) in multiple clauses, the strongest lock is used. Similarly, if there is **NOWAIT** in any of the clauses that affect a table, then **NOWAIT** will apply to the table.

NOTICE

- The LOCK IN SHARE MODE clause is equivalent to the FOR SHARE clause in which no other options are specified.
- Only FOR SHARE/FOR UPDATE/FOR READ ONLY/LOCK IN SHARE MODE can be used in querying the Ustore table.
- For the FOR UPDATE or FOR SHARE statements whose subquery is a stream plan, the same row that is locked cannot be concurrently updated.
- SELECT FOR UPDATE and SELECT FOR SHARE cannot be used together with UNION, EXCEPT, DISTINCT, GROUP BY, or HAVING.

- **PARTITION clause**

  Queries data in the corresponding partition in a partitioned table.

Simple Queries

A simple query retrieves one or more columns of data from one or more tables or views.

```
-- Create a table and insert data into the table.
m_db=# CREATE TABLE student(
    sid INT PRIMARY KEY,
    class INT,
    name  VARCHAR(50),
    gender INT CHECK(gender = 0 OR gender = 1) -- Gender. 1: male; 0: female
);
m_db=# INSERT INTO student (sid, class, name, gender) VALUES (1, 1, 'Michael', 0);
m_db=# INSERT INTO student (sid, class, name, gender) VALUES (2, 2, 'Bob', 1);
m_db=# INSERT INTO student (sid, class, name, gender) VALUES (3, 2, 'Gary', 0);

-- Query some columns.
m_db=# SELECT sid,name FROM student;
 sid |  name
-----+---------
   1 | michael
   2 | bob
   3 | Gary
(3 rows)

-- Query all columns.
m_db=# SELECT * FROM student;
 sid | class |  name   | gender
-----+-------+---------+-----
   1 |     1 | michael |   0
   2 |     2 | bob     |   1
   3 |     2 | Gary    |   0
(3 rows)

-- Assign an alias to a column.
m_db=# SELECT sid student_id, name FROM student;
 student_id |  name
------------+---------
          1 | michael
          2 | bob
          3 | Gary
(3 rows)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop.
m_db=# DROP TABLE student;
```

Conditional Queries

A conditional query retrieves data that meets the conditions.

```
-- Create a table and insert data into the table.
m_db=# CREATE TABLE test_grade(
    id INT,
    name VARCHAR(20),
    score FLOAT       -- Score
);
m_db=# INSERT INTO test_grade VALUES (1,'Scott',90),(2,'Jack',87.5),(3,'Ben',48);

-- Query information about students who fail to pass the test.
m_db=# SELECT * FROM test_grade WHERE score < 60;
 id | name | score
----+------+-------
  3 | Ben  |    48
(1 row)

-- Drop.
m_db=# DROP TABLE test_grade;
```

NOTE

Column aliases can be used in conditional queries.

Group Queries

A group query is usually used together with aggregate functions to query category statistics. Common aggregate functions include count(), sum(), avg(), min(), and max(). GROUP BY is often used in group queries.

```
-- Create a table and insert data into the table.
m_db=# CREATE TABLE student (id INT,name VARCHAR(20),class INT);
m_db=# INSERT INTO student VALUES (1, 'Scott', 1), (2, 'Ben', 1);
m_db=# INSERT INTO student VALUES (3, 'Jack', 2),(4, 'Anna', 2),(5, 'Judy', 2);
m_db=# INSERT INTO student VALUES (6, 'Sally', 3), (7, 'Jordan', 3);

-- Use the aggregate function COUNT() to count the number of people in each class.
m_db=# SELECT class, count(*) AS cnt FROM student GROUP BY class;
 class | cnt
-------+-----
     2 |   3
     1 |   2
     3 |   2
(3 rows)

-- Use the HAVING clause to retrieve data of classes with fewer than three people.
m_db=# SELECT class, COUNT(*) AS num FROM student GROUP BY class HAVING COUNT(*) < 3;
 class | num
-------+-----
     1 |   2
     3 |   2
(2 rows)

-- Drop.
m_db=# DROP TABLE student;
```

Pagination Queries

The syntax is as follows:

```
SELECT query_list FROM table_name [ LIMIT { [offset,] count | ALL } ]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **offset**: indicates the number of rows to skip before starting to return rows.
- **count**: indicates the number of data records to be retrieved after the skipped rows.
- **ALL**: to retrieve all data after the skipped rows.

```
-- Create a table and insert 100 data records into the table.
m_db=# CREATE TABLE testl(id int PRIMARY KEY, flag varchar(10));
m_db=# INSERT INTO testl (id, flag) VALUES (generate_series(1,100),generate_series(1,100));

-- Query the first five data records.
m_db=# SELECT * FROM testl ORDER BY 1  LIMIT 5;
 id | flag
----+-------
  1 | 1
  2 | 2
  3 | 3
  4 | 4
  5 | 5
(5 rows)

-- Query four data records after the twentieth record.
m_db=# SELECT * FROM testl ORDER BY 1 LIMIT 20,4;
 id |  flag
----+--------
 21 | 21
 22 | 22
 23 | 23
 24 | 24
(4 rows)

-- Query all data after the ninety-sixth record.
m_db=# SELECT * FROM testl ORDER BY 1 LIMIT 96,ALL;
 id  |  flag
-----+---------
  97 | 97
  98 | 98
  99 | 99
 100 | 100
(4 rows)

-- Drop.
m_db=# DROP TABLE testl;
```

Partition Queries

Queries data in a specified partition.

```
-- Create a range partitioned table.
m_db=# CREATE TABLE test_range1(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION p2 VALUES LESS THAN (400),
    PARTITION p3 VALUES LESS THAN (600),
    PARTITION p4 VALUES LESS THAN (800),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);
-- Insert 1000 data records.
m_db=# INSERT INTO test_range1 VALUES(GENERATE_SERIES(1,1000),'abcd');

-- Query the number of data records in the p1 partition.
m_db=# SELECT COUNT(*) FROM test_range1 PARTITION (p1);
 count
-------
   199
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop.
m_db=# DROP TABLE test_range1;
```

Join Queries

A join query may also be referred to as a cross-table query, where multiple tables need to be associated.

```
-- Create a table and insert data into the table.
m_db=# CREATE TABLE emp(
    id int,             -- Employee ID
    name varchar(20),   -- Employee name
    deptno int          -- Department ID of the employee
);

m_db=# CREATE TABLE dept(
    deptno int,         -- Department ID
    depname varchar(20) -- Department name
);

m_db=# INSERT INTO emp VALUES (1, 'Joe', 10), (2, 'Scott', 20), (3, 'Ben', 999);   -- The department for Ben has not been confirmed yet. Therefore, the department ID is 999.
m_db=# INSERT INTO dept VALUES (10, 'hr'), (20, 'it'), (30, 'sal');                -- There is no employee in the sal department.
```

- INNER JOIN

  ```
  m_db=# SELECT t1.id,t1.name,t2.depname FROM emp t1 JOIN dept t2 ON t1.deptno = t2.deptno;
   id | name  | depname
  ----+-------+---------
    1 | Joe   | hr
    2 | Scott | it
  (2 rows)
  ```

- LEFT JOIN

  ```
  m_db=# SELECT t1.id,t1.name,t2.depname FROM emp t1 LEFT JOIN dept t2 ON t1.deptno = t2.deptno;
   id | name  | depname
  ----+-------+---------
    1 | Joe   | hr
    2 | Scott | it
    3 | Ben   |
  (3 rows)
  ```

- RIGHT JOIN

  ```
  m_db=# SELECT t1.id,t1.name,t2.depname FROM emp t1 RIGHT JOIN dept t2 ON t1.deptno =
  t2.deptno;
   id | name  | depname
  ----+-------+---------
    1 | Joe   | hr
    2 | Scott | it
      |       | sal
  (3 rows)
  ```

Subqueries

- Subqueries can be classified into single-row subqueries and multi-row subqueries based on the number of records returned.
- Subqueries can be classified into correlated subqueries and non-correlated subqueries based on whether a subquery is executed for multiple times.

  A query can be nested in another query, and its result is used as the data source or condition of the other query. Outer queries are also called parent queries, and inner queries are also called subqueries.
- Alias cannot be assigned in a subquery.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Single-Row Subqueries

Single-row subquery operators include >=, >, <=, <, and <>.

```
-- Create a student table and insert data into the table.
m_db=# CREATE TABLE student(
    sid VARCHAR(5),     -- Student ID
    grade INT,          -- Grade
    name VARCHAR(20),   -- Name
    height INT          -- Height
);
m_db=# INSERT INTO student VALUES ('00001',1,'Scott',135),('00002',1,'Jack',95),('00003',1,'Ben',100);
m_db=# INSERT INTO student VALUES ('00004',2,'Henry',115),('00005',2,'Jordan',130),('00006',2,'Bob',126);
m_db=# INSERT INTO student VALUES ('00007',3,'Bella',128),('00008',3,'Alicia',136);
-- Create a teacher table and insert data into the table.
m_db=# CREATE TABLE teacher (
    name VARCHAR(20),    -- Teacher name
    grade INT            -- Class
);
-- Insert data.
m_db=# INSERT INTO teacher VALUES ('Bill',1),('Sally',2),('Luke',3);

-- Query the students who are taller than Bella.
m_db=# SELECT * FROM student
WHERE height > (SELECT height FROM student WHERE name = 'Bella');
  sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00008 |     3 | Alicia |    136
(3 rows)
```

Multi-Row Subqueries

Multi-row subquery operators:

- IN: equal to any member in the list.
- ANY: used together with single-row comparison operators to compare with any value returned by the subquery.
- ALL: used together with single-row comparison operators to compare with all values returned by the subquery.
- SOME: alias of ANY and both operators have the same effect.

Example: Query the students of Sally and Luke.

```
m_db=# SELECT * FROM student t1 WHERE t1.grade IN (
    SELECT grade FROM teacher WHERE name = 'Sally' OR name = 'Luke'
);
  sid  | grade |  name  | height
-------+-------+--------+--------
 00004 |     2 | Henry  |    115
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00007 |     3 | Bella  |    128
 00008 |     3 | Alicia |    136
(5 rows)
```

Example: Query the students in grade 2 who are taller than any students in grade 3.

```
m_db=# SELECT * FROM student
WHERE grade = 2 AND
      height > ANY (SELECT height FROM student WHERE grade = 3);
  sid  | grade |  name  | height
-------+-------+--------+--------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 00005 |     2 | Jordan |    130
(1 row)

-- The query result is equivalent to that of the following query:
m_db=# SELECT * FROM student
WHERE grade = 2 AND
      height > (SELECT MIN(height) FROM student WHERE grade = 3);
```

Example: Query the students in grade 1 who are taller than all students in grade 2.

```
m_db=# SELECT * FROM student
WHERE grade = 1 AND
      height > ALL (SELECT height FROM student WHERE grade = 2);
  sid  | grade | name  | height
-------+-------+-------+--------
 00001 |     1 | Scott |    135
(1 row)

-- The query result is equivalent to that of the following query:
m_db=# SELECT * FROM student
WHERE grade = 1 AND
      height > (SELECT MAX(height) FROM student WHERE grade = 2);
```

Correlated Subqueries

Characteristics: Subqueries cannot run independently and are related to parent queries. Execute the parent query and then the subquery. Each time the parent query is executed, the subquery is recalculated.

Example: Identify students whose height is above the class average.

```
m_db=# SELECT * FROM student out
WHERE height > (SELECT AVG(height) FROM student
                WHERE grade = out.grade);
  sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00008 |     3 | Alicia |    136
(4 rows)
```

Non-correlated Subqueries

Characteristics: The subquery queries the value and then returns the result to the outer layer for query.

Example: Identify students whose height is above the class average.

```
m_db=# SELECT t1.*
FROM student t1,
     (SELECT grade, AVG(height) avg_hei FROM student GROUP BY grade) t2
WHERE t1.grade = t2.grade AND
      t1.height > t2.avg_hei;
  sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00008 |     3 | Alicia |    136
(4 rows)
-- Drop.
m_db=# DROP TABLE student;
m_db=# DROP TABLE teacher;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Compound Queries

A query that contains a compound operator is a compound query. All compound queries have the same priority. The number of columns and expressions in each query result in a set operation must be the same, and the types must be compatible.

Common set operations include:

- UNION: a union set of the two query result sets where duplicates are removed.
- UNION ALL: a union set of the two queries where the results of the two queries are simply combined.

Examples

```
-- Create a table and insert data into the table.
m_db=# CREATE TABLE test1(c11 INT, c12 VARCHAR(20));
m_db=# INSERT INTO test1 VALUES (1,'a'),(2,'b'),(4,'d');

m_db=# CREATE TABLE test2(c21 INT, c22 VARCHAR(20));
m_db=# INSERT INTO test2 VALUES (1,'a'),(3,'c');
```

- UNION

```
m_db=# SELECT * FROM test1 UNION SELECT * FROM test2;
 c11 | c12
-----+-----
   4 | d
   3 | c
   1 | a
   2 | b
(4 rows)
```

- UNION ALL

```
m_db=# SELECT * FROM test1 UNION ALL SELECT * FROM test2;
 c11 | c12
-----+-----
   1 | a
   2 | b
   4 | d
   1 | a
   3 | c
(5 rows)
```

4.4.2.16.3 SELECT INTO

Description

SELECT INTO creates a table based on the query result and inserts data retrieved by the query into the new table.

Different from SELECT, data is not returned to the client. The columns of the new table have the same names and data types as the output columns of SELECT.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [SQL_NO_CACHE]
    select_expr [, select_expr] ...
    INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY ]  ] [ TABLE ] new_table
    [FROM from_item]
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
[WHERE where_condition]
[GROUP BY {col_name | expr | position}]
[HAVING condition]
[ORDER BY {col_name | expression | position}
  [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY ]  ] [ TABLE ] new_table
[ FOR READ ONLY |
    [ {FOR { UPDATE | SHARE | } [ OF table_name [, ...] ] [ NOWAIT | WAIT N ]} [...] ] |
    LOCK IN SHARE MODE];
```

Parameters

- **new_table**

  Specifies the name of the new table.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify **GLOBAL** or **LOCAL** before **TEMP** or **TEMPORARY**. If **GLOBAL** is specified, a global temporary table will be created for an M-compatible database. Otherwise, a local temporary table will be created.

- **TEMPORARY**

  If **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If **GLOBAL** is specified when creating a temporary table, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of a global temporary table is visible to all sessions. After sessions end, the metadata still exists. The user data, indexes, and statistics between sessions are isolated. Each session can only view and modify the data committed by itself. Global temporary tables can be created using ON COMMIT PRESERVE ROWS or ON COMMIT DELETE ROWS. The former creates session-level tables, where user data is automatically cleared when a session ends; the latter creates transaction-level tables, where user data is automatically cleared when a commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session-based schema is used by default. Different from local temporary tables, you can specify a schema that does not start with pg_temp_ when creating a global temporary table.

  A local temporary table is visible only in the current session and is automatically dropped at the end of the session. Therefore, you can create and use the temporary tables in the current session as long as the database node that the session connects to is normal. Using DDL statements on temporary tables will cause a DDL failure error because temporary tables are created only in the current session. Therefore, you are advised not to perform operations on temporary tables using DDL statements. TEMP is equivalent to TEMPORARY.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

- Local temporary tables use independent schemas starting with **pg_temp** for each session to ensure that they are visible only to the current session. Therefore, you are advised not to manually delete schemas starting with **pg_temp** or **pg_toast_temp** during routine operations.
- If TEMPORARY or TEMP is not specified when creating the table and the schema of the table is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables or indexes are being used by other sessions, do not perform ALTER or DROP on the tables or indexes.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, TRUNCATE, REINDEX, and ANALYZE are valid only for the current session.

NOTE

For details about other SELECT INTO parameters, see **Parameters** in section "SELECT".

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table named tpcds.reason.
m_db=# CREATE TABLE tpcds.reason
(
  r_reason_sk      integer,
  r_reason_id      character(16),
  r_reason_desc    character(100)
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2'),(3,'AAAAAAAABAAAAAAA','reason 3'),
(4,'AAAAAAAABAAAAAAA','reason 4'),(4,'AAAAAAAABAAAAAAA','reason 5'),
(4,'AAAAAAAACAAAAAAA','reason 6'),(5,'AAAAAAAACAAAAAAA','reason 7');

-- Add the values of r_reason_sk that are less than 5 in the tpcds.reason table to a new table.
m_db=# SELECT * INTO tpcds.reason_t1 FROM tpcds.reason WHERE r_reason_sk < 5;
INSERT 0 6

-- Drop the tpcds.reason_t1 table.
m_db=# DROP TABLE tpcds.reason_t1;

-- Drop the table.
m_db=# DROP TABLE tpcds.reason;

-- Drop the schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

SELECT

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.4.2.16.4 SET

Description

Modifies GUC parameters.

Precautions

Most GUC parameters can be modified at runtime by using SET, but some parameters cannot be modified after a service or session starts.

Syntax

- Set the time zone.
```
SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
```
- Set the schema.
```
SET [ SESSION | LOCAL ]
  {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
  | SCHEMA 'schema'};
```
- Set the client character set.
```
SET [ SESSION | LOCAL ] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};
```
- Set other GUC parameters.
```
SET [ SESSION | LOCAL ]
    {config_parameter { TO | = } { value | DEFAULT } | FROM CURRENT }};
```

Parameters

- **SESSION**

  Specifies that the specified parameters take effect only for the current session. If neither **SESSION** nor **LOCAL** appears, **SESSION** is the default value.

  If this command is executed in a transaction, the effect of the command disappears after the transaction is rolled back. Once the surrounding transaction is committed, the effect will persist until the end of the session, unless the parameters are reset by another SET statement.

- **LOCAL**

  Specifies that the specified parameters take effect only for the current transaction. After COMMIT or ROLLBACK, session-level settings will take effect again.

  The effect of this command lasts only until the end of the current transaction, whether the transaction is committed or not. A special case is where SET is followed by SET LOCAL within a single transaction. In this case, SET LOCAL will take effect until the end of the transaction, but after the transaction is committed, SET will take effect.

- **TIME ZONE**

  Specifies the local time zone for the current session.

  Value range: a valid local time zone. The corresponding GUC parameter is **TimeZone**. The default value is **PRC**.

- **CURRENT_SCHEMA**

  Specifies the current schema.

  Value range: an existing schema name.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- **SCHEMA**

  Same as **CURRENT_SCHEMA**. The schema here is a string.

  Example: **set schema 'public';**

- **NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};**

  Specifies the client character encoding name. It is equivalent to **set client_encoding to charset_name**.

  It is equivalent to:

  ```
  set client_encoding = charset_name; set character_set_connection = charset_name;
  set collation_connection = collation_name; set character_set_results = charset_name;
  ```

  Value range: character sets and collations supported in M-compatible mode. Currently, **charset_name** must be one of the character sets specified for the database.

- **config_parameter**

  Specifies the name of the configurable GUC parameter. You can use SHOW ALL to view the available GUC parameters.

  NOTE

  Some parameters viewed by SHOW ALL cannot be set by using SET. For example, **max_datanodes**.

- **value**

  Specifies the new value of **config_parameter**. You can specify the parameter as a string constant, identifier, number, or comma-separated list. When **DEFAULT** is specified, the parameter is set to its default value.

Examples

```
-- Set the schema search path.
m_db=# SET search_path TO tpcds, public;

-- Set the date style to the traditional POSTGRES style (date placed before month).
m_db=# SET datestyle TO postgres,dmy;
```

Helpful Links

RESET and **SHOW**

4.4.2.16.5 SET ROLE

Description

Sets the current user identifier for the current session.

Precautions

- Users of the current session must be the role members specified by **rolename**, but system administrators can choose any roles.
- This statement may add permissions to or restrict permissions of a user. If the role of a session user has the INHERITS attribute, then the user is automatically granted all permissions of the roles that SET ROLE enables the user to be. In this case, SET ROLE deletes all permissions directly granted to the session user, as well as the permissions of the role it belongs to, that is,

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the user is left with the permissions of only the specified role. On the other hand, if the role of a session user has the NOINHERITS attribute, SET ROLE deletes permissions directly granted to the session user and obtains permissions of the specified role.

Syntax

- Set the current user identifier of the current session.
  ```
  SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
  ```
- Reset the current user identifier to that of the current session.
  ```
  SET ROLE = DEFAULT;
  ```

Parameters

- **SESSION**

  Specifies that the command takes effect only for the current session. This parameter is used by default.
- **LOCAL**

  Specifies that the command takes effect only for the current transaction.
- **role_name**

  Value range: an existing role name. For details, see **•role_name**.
- **password**

  Specifies the password for the role. It must comply with the password convention. Encrypted passwords are not supported.

Examples

```
-- Create a role named paul.
m_db=# CREATE ROLE paul IDENTIFIED BY '********';

-- Set the current user to paul.
m_db=# SET ROLE paul PASSWORD '********';

Query the current user.
m_db=# SELECT CURRENT_USER;

-- Reset the current user.
m_db=# SET role = DEFAULT;

-- Drop the user.
m_db=# DROP USER paul;
```

4.4.2.16.6 SET SESSION AUTHORIZATION

Description

Sets the session user identifier and the current user identifier of the current session to the specified user.

Precautions

The session identifier can be changed only when the initial session user has the system administrator permissions. Otherwise, the system executes the command only when an authenticated username is specified.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax

- Set the session user identifier and current user identifier for the current session.
  ```
  SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
  ```
- Reset the session and current user identifiers to the originally authenticated username.
  ```
  {SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
  | SET SESSION_AUTHORIZATION = DEFAULT};
  ```

Parameters

- **SESSION**

  Specifies that the command takes effect only for the current session.
- **LOCAL**

  Specifies that the command takes effect only for the current transaction.
- **role_name**

  Specifies the username.

  Value range: a string. For details about the username requirements, see **•user_name**.
- **password**

  Specifies the password for the role. The password must comply with the password convention. Encrypted passwords are not supported.
- **DEFAULT**

  Resets the identifiers of the session and current users to the initially authenticated usernames.

Examples

```
-- Create a role named paul.
m_db=# CREATE ROLE paul IDENTIFIED BY '********';

-- Set the current user to paul.
m_db=# SET SESSION AUTHORIZATION paul password '********';

Query the current user.
m_db=# SELECT CURRENT_USER;

-- Reset the current user.
m_db=# SET SESSION_AUTHORIZATION = DEFAULT;

-- Drop the user.
m_db=# DROP USER paul;
```

Helpful Links

SET ROLE

4.4.2.16.7 SET TRANSACTION

Description

Sets the transaction characteristics. Transaction characteristics include transaction isolation level and transaction access mode (read/write or read-only). You can set

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the current transaction characteristics using LOCAL or the default transaction characteristics of a session using SESSION.

Precautions

The characteristics of the current transaction must be set in a transaction, that is, START TRANSACTION or BEGIN must be executed before SET TRANSACTION is executed. Otherwise, the settings do not take effect.

Syntax

Set the transaction isolation level and access mode.

```
{ SET [ LOCAL | SESSION ] TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } };
```

Parameters

- **LOCAL**

  Specifies that the command takes effect only for the current transaction.

- **SESSION**

  Specifies that the command takes effect only for the current session.

- **ISOLATION LEVEL**

  Specifies a transaction isolation level, which determines what data is visible to a transaction if there are other concurrent transactions.

  NOTE

  The isolation level cannot be changed after data is modified using SELECT, INSERT, DELETE, UPDATE, or COPY in the current transaction.

  Value range:

  - **READ COMMITTED**: Only committed data can be read. This is the default value.
  - **READ UNCOMMITTED**: After this isolation level is set, its behavior is the same as that of **READ COMMITTED**.
  - **REPEATABLE READ**: Only the data committed before the transaction starts can be read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported in an M-compatible database. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read-only).

  NOTE

  The access mode of the default transaction feature of the session can be set only when the database is started or by sending the HUP signal.

Examples

```
-- Start a transaction and set its isolation level to READ COMMITTED and access mode to READ ONLY:
m_db=# START TRANSACTION;
m_db=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# COMMIT;

-- Set the transaction isolation level and access mode for the current session.
m_db=# SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
m_db=# SET SESSION TRANSACTION READ ONLY;
```

4.4.2.16.8 SHOW

Description

SHOW shows the current value of a GUC parameter.

Syntax

```
SHOW
 {
   [VARIABLES LIKE] configuration_parameter |
   TIME ZONE |
   TRANSACTION ISOLATION LEVEL |
   SESSION AUTHORIZATION |
   ALL
 };

SHOW [DATABASES | SCHEMAS] [like_or_where]

like_or_where:
 {
   LIKE 'pattern' |
   WHERE expr
 };
```

Parameters

- **configuration_parameter**

  Specifies the name of the GUC parameter.

  Value range: GUC parameters. You can view the GUC parameters by using the SHOW ALL statement.

  NOTE

  SHOW CHARSET, SHOW COLLATION, and SHOW PROCESSLIST are not supported.

- **TIME ZONE**

  Specifies the time zone.

- **TRANSACTION ISOLATION LEVEL**

  Specifies the transaction isolation level.

- **SESSION AUTHORIZATION**

  Specifies the user identifier of the current session.

- **DATABASES | SCHEMAS**

  Specifies all schemas in the current database.

- **ALL**

  Specifies all GUC parameters.

SHOW syntax examples

```
-- Show the value of timezone.
m_db=# SHOW timezone;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Show all parameters.
m_db=# SHOW ALL;

-- Show all parameters whose names contain var.
m_db=# SHOW VARIABLES LIKE var;
```

Helpful Links

SET and **RESET**

4.4.2.16.9 START TRANSACTION

Description

START TRANSACTION starts a transaction. If an isolation level and the read/write mode are specified, a new transaction will have those characteristics. You can also specify them using **SET TRANSACTION**.

Syntax

Format 1: START TRANSACTION

```
START TRANSACTION
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [ ...]
 ];
```

Format 2: BEGIN

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
  } [ ...]
 ];
```

Parameters

- **WORK | TRANSACTION**

  Specifies an optional keyword in the BEGIN format and does not have any actual effect.

- **ISOLATION LEVEL**

  Specifies a transaction isolation level, which determines what data is visible to a transaction if there are other concurrent transactions.

  NOTE

  The isolation level cannot be changed after data is modified using SELECT, INSERT, DELETE, UPDATE, or COPY in the current transaction.

  Value range:

  - **READ COMMITTED**: Only committed data can be read. This is the default value.
  - **READ UNCOMMITTED**: After this isolation level is set, its behavior is the same as that of **READ COMMITTED**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

  - **REPEATABLE READ**: Only the data committed before the transaction starts can be read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported in an M-compatible database. It is equivalent to **REPEATABLE READ**.
- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read-only).

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table named tpcds.reason.
m_db=# CREATE TABLE tpcds.reason (c1 int, c2 int);

-- Start a transaction in default mode.
m_db=# START TRANSACTION;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Start a transaction in default mode.
m_db=# BEGIN;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ WRITE:
m_db=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
m_db=# SELECT * FROM tpcds.reason;
m_db=# COMMIT;

-- Drop the tpcds.reason table.
m_db=# DROP TABLE tpcds.reason;

-- Drop the schema.
m_db=# DROP SCHEMA tpcds;
```

Helpful Links

COMMIT, **ROLLBACK**, and **SET TRANSACTION**

4.4.2.17 T

4.4.2.17.1 TRUNCATE

Description

TRUNCATE quickly removes all rows from a database table.

It has the same effect as an unqualified DELETE statement on a table, but TRUNCATE is faster because it does not scan the tables. This is most useful on large tables.

Precautions

- TRUNCATE TABLE has the same function as a DELETE statement without WHERE clause, where both delete all rows in a table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- TRUNCATE TABLE uses less system and transaction log resources than DELETE.
  - DELETE deletes a row each time and records the deletion of each row in the transaction log.
  - TRUNCATE TABLE deletes data by releasing data pages storing the table data. It only records the releasing of the data page in the transaction log.
- The differences between TRUNCATE, DELETE, and DROP are as follows:
  - TRUNCATE TABLE deletes content, releases space, but does not delete definitions.
  - DELETE TABLE deletes content, but does not delete definitions nor release space.
  - DROP TABLE deletes content and definitions, and releases space.

Syntax

- Truncate table data.

```
TRUNCATE [ TABLE ] [ONLY] {table_name [ * ]} [, ... ]
   [CASCADE | RESTRICT][ PURGE ]};
```

- Truncate partition data.

```
ALTER TABLE { [ ONLY  ] table_name
          | table_name *
          | ONLY ( table_name )  }
   TRUNCATE PARTITION { { ALL | partition_name [, ...] }
                  | FOR (  partition_value  [, ...] )  } [ UPDATE GLOBAL INDEX ];
```

Parameters

- **ONLY**

  If **ONLY** is specified, only the specified table is truncated. If **ONLY** is not specified, the table and all its subtables (if any) are truncated. Currently, the syntax of **ONLY** with the ***** option is reserved, but the function is not supported.

- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: an existing table name.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as other views) that depend on the view.
  - **RESTRICT**: refuses to delete the view if any objects depend on it. This is the default action.

  NOTE

  This attribute is not supported when the version compatibility control parameter **m_format_dev_version** of an M-compatible database is set to **'s1'** or later.

- **PURGE**

  Removes table data which is moved to the recycle bin by default.

- **{ ALL | partition_name [, ...] }**

  ALL: clears all partition data.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

partition_name: specifies a partition in the target partitioned table. Level-1 and level-2 partition names are supported.

Value range: an existing partition name.

- **partition_value**

  Specifies the partition key value.

  You can use PARTITION FOR to uniquely identify a partition.

  Value range: value range of partition keys of the partition to be truncated.

NOTICE

1. If a table is a foreign table of another table, running TRUNCATE to clear data will fail and an error will be reported.

2. When the PARTITION FOR clause is used, the entire partition that **partition_value** corresponds to is truncated.

- **UPDATE GLOBAL INDEX**

  If this parameter is specified, all global indexes in the partitioned table will be updated to ensure that data can be queried properly using global indexes. If this parameter is not specified, all global indexes in the partitioned table will become invalid.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table named tpcds.reason.
m_db=# CREATE TABLE tpcds.reason
(
  r_reason_sk      integer,
  r_reason_id      character(16),
  r_reason_desc    character(100)
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAABAAAAAAA','reason 2'),(15,'AAAAAAAABAAAAAAA','reason 3'),
(25,'AAAAAAAABAAAAAAA','reason 4'),(35,'AAAAAAAABAAAAAAA','reason 5'),
(45,'AAAAAAAACAAAAAAA','reason 6'),(55,'AAAAAAAACAAAAAAA','reason 7');

-- Create a table.
m_db=# CREATE TABLE tpcds.reason_t1  LIKE tpcds.reason;

-- Truncate the tpcds.reason_t1 table.
m_db=# TRUNCATE TABLE tpcds.reason_t1;

-- Drop the table.
m_db=# DROP TABLE tpcds.reason_t1;
-- Create a partitioned table.
m_db=# CREATE TABLE tpcds.reason_p
(
  r_reason_sk integer,
  r_reason_id character(16),
  r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)
(
  partition p_05_before values less than (05),
  partition p_15 values less than (15),
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
  partition p_25 values less than (25),
  partition p_35 values less than (35),
  partition p_45_after values less than (MAXVALUE)
);

-- Insert data.
m_db=# INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;

-- Truncate the p_05_before partition.
m_db=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;

-- Truncate the p_15 partition.
m_db=# ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (15);

-- Truncate the partitioned table.
m_db=# TRUNCATE TABLE tpcds.reason_p;

-- Drop the table.
m_db=# DROP TABLE tpcds.reason_p;

-- Drop the table.
m_db=# DROP TABLE tpcds.reason;

-- Drop the schema.
m_db=# DROP SCHEMA tpcds;
```

4.4.2.18 U

4.4.2.18.1 UPDATE

Description

UPDATE updates the data in a table, modifying the values of the specified columns in all rows that meet the conditions. The WHERE clause specifies the conditions. Only the columns specified in the SET clause will be modified.

Precautions

- The table owner, users with the UPDATE permission on the current table, and users with the UPDATE ANY TABLE permission can update data in a table. System administrators automatically have this permission when separation of duties is disabled.
- UPDATE..... LIMIT row_count supports only some scenarios of execution plan pushdown. (Table replication is not supported.) The prerequisites are that the filter criteria must contain the equivalent distribution key and the filter criteria should be relatively simple. Do not use forcible type conversion. If the command fails to be executed, simplify the filter criteria.
- You must have the SELECT permission on all tables involved in **expression** or **condition**.
- The distribution keys can be updated only when **enable_update_distkey** is set to **on** and constraints are met. Otherwise, distribution keys do not support the UPDATE operation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

The constraints on updating distribution keys are as follows:

- Distribution keys can be updated only when **enable_update_distkey** is set to **on**.
- The UPDATE statement cannot be pushed down to DNs for execution. The PGXC plan is directly generated and is not changed based on the values before and after the distribution key is updated.
- Tables with row-level UPDATE TRIGGER are not supported. Otherwise, the execution fails and an error message is displayed. The row-level INSERT/DELETE TRIGGER does not take effect, and the update-statement-level TRIGGER is executed normally.
- Concurrent update of the same row is not supported. You need to obtain a lock first. The result (returning **0** or reporting an error) of obtaining a lock on DNs depends on the setting of the GUC parameter **concurrent_dml_mode**. An error will be reported in the following cases: (1) The error message "update distribute column conflict" is displayed. (2) When the time spent for obtaining a lock exceeds the threshold, an error message is displayed, indicating that the lock times out.
- Tables with global secondary indexes (GSIs) are not supported. Otherwise, an error is reported.
- Only Hash distribution is supported. LIST/RANGE distributed tables are not supported. Otherwise, an error is reported.
- MERGE INTO and UPSERT cannot be used to update distribution keys. Otherwise, an error is reported.
- GTM_FREE is not supported. Otherwise, an error is reported.
- Statements with joined tables are not supported. Otherwise, an error is reported.
- UPDATE+LIMIT is not supported. Otherwise, an error is reported.

- When the UPDATE statement uses a STREAM plan as a subquery, concurrent updates to the same row are not allowed.
- You cannot modify the character set encoding of a database through an UPDATE on a system catalog. This action may lead to exceptions in existing data or some other operations. If you need to modify the encoding, follow the database switching process for data migration.

Syntax

```
UPDATE [/*+ plan_hint */] [IGNORE] [ ONLY ] {table_name [ partition_clause ] | view_name} [ * ] [ [ AS ] alias ]
SET {column_name = { expression | DEFAULT }
}[, ...]
   [ WHERE condition ]
   [ORDER BY {col_name | expression | position}
      [ASC | DESC][ NULLS { FIRST | LAST } ], ...]
    [ LIMIT { count } ]
    [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

Parameters

- **plan_hint**

  Follows the UPDATE keyword in the **/*+ */** format. It is used to optimize the plan of an UPDATE statement block. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **IGNORE**

  When the UPDATE statement uses the IGNORE keyword, some ERROR-level errors can be degraded to WARNING-level errors, and invalid values can be

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

adjusted to the closest values based on error scenarios. GaussDB supports the following error downgrade scenarios:

- Violation of the NOT NULL constraint
- UNIQUE KEY conflict
- No partition found for the inserted value
- Unmatch between the inserted data and the specified partition
- Multiple rows returned for a subquery
- **sql_mode** set to loose

- **table_name**

  Specifies the name (optionally schema-qualified) of the table to be updated. If **ONLY** is specified before the table name, only the matching rows in that specific table will be updated. If **ONLY** is not specified, the update will also affect matching rows in other tables inheriting data from the specified table. Currently, the syntax of **ONLY** with the * option is reserved, but the function is not supported.

  Value range: existing table names.

- **partition_clause**

  Updates the specified partition.

  PARTITION { ({ partition_name | subpartition_name } [, ...]) | FOR (partition_value [, ...]) }

  For keywords, see **SELECT**.

  NOTE

  If PARTITION specifies multiple partition names, the partition names can be the same. The union set of the partition ranges is used.

- **view_name**

  Specifies the target view to be updated.

  NOTE

  The restrictions on updating views are as follows:

  - The UPDATE operation can be performed only on columns that directly reference user columns in the base table.
  - A view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
  - Views that contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause at the top layer are not supported.
  - Views that contain set operations (UNION and EXCEPT) at the top layer are not supported.
  - Views whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
  - Table types supported in views include ordinary tables, temporary tables, global temporary tables, partitioned tables, Ustore tables, and Astore tables.
  - Only one base table can be updated at a time in a multi-table join view.
  - Join views can only update key-preserved tables. For details about the key-preserved table, see **CREATE VIEW**.
  - System views cannot be updated.

- **alias**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Specifies the alias of the target table.

Value range: character strings complying with **Identifier Description**.

- **table_list**

  Specifies a list of table expressions. It is similar to **from_list**, but you can specify both the target table and joined tables. This parameter is used only in the syntax for updating multiple tables.

- **column_name**

  Specifies the name of the column to be modified.

  You can reference this column by using the target table alias and column name. Example: **UPDATE foo AS f SET f.col_name = 'namecol'**

  The column can be referenced in the format of *Database name*.*Alias* or *Database name*.*Table name*+*Column name*. Example: **UPDATE foo AS f SET public.f.col_name = 'namecol'**.

  Value range: existing column names.

- **expression**

  Specifies a value or expression for a column.

- **DEFAULT**

  Specifies the default value of a column.

  The value is **NULL** if there is no default value.

- **from_list**

  Specifies a list of table expressions. You can use columns of other tables in the WHERE condition. It is similar to specifying a table list in a FROM clause of a SELECT statement.

  NOTICE

  Note that the target table cannot appear in **from_list**, unless you intend a self-join (in which case it must appear with an alias in **from_list**).

- **condition**

  Specifies an expression that returns a value of type Boolean. Only rows for which this expression returns **true** are updated. To prevent unexpected outcomes, avoid using numeric types such as int for **condition**, because these types can be implicitly converted into Boolean values: **0** becomes **false**, while other values become **true**.

- **ORDER BY**

  For keywords, see **SELECT**.

- **LIMIT**

  For keywords, see **SELECT**.

- **RETURNING**

  Returns the inserted rows. The syntax of the RETURNING list is identical to that of the output list of SELECT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example

- Modify all data in the table.

```
-- Create the tbl_test1 table and insert data into it.
m_db=# CREATE TABLE tbl_test1(id int, info varchar(10));
m_db=# INSERT INTO tbl_test1 VALUES (1, 'A'), (2, 'B');

-- Query.
m_db=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | A
  2 | B
(2 rows)

-- Modify information in the info column in the tbl_test1 table.
m_db=# UPDATE tbl_test1 SET info = 'aa';

-- Query the tbl_test1 table.
m_db=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | aa
  2 | aa
(2 rows)
```

- Modify some data in the table.

```
-- Modify the data with an id value of 2 in the tbl_test1 table.
m_db=# UPDATE tbl_test1 SET info = 'bb' WHERE id = 2;

-- Query the tbl_test1 table.
m_db=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | aa
  2 | bb
(2 rows)
```

- Modify the data and return the modified data.

```
-- Modify the data with an id value of 1 in the tbl_test1 table and specify that the info column
should be returned.
m_db=# UPDATE tbl_test1 SET info = 'ABC' WHERE id = 1;
UPDATE 1

-- Drop the tbl_test1 table.
m_db=# DROP TABLE tbl_test1;
```

- Update a view.

```
-- Create a schema.
m_db=# CREATE SCHEMA upd_view;
CREATE SCHEMA
m_db=# SET CURRENT_SCHEMA = 'upd_view';
SET

-- Create a table and insert data into the table.
m_db=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
m_db=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
m_db=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
m_db=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4

-- Create a single table view.
m_db=# CREATE VIEW v_upd1 AS SELECT * FROM t1;
CREATE VIEW

-- Update t1 through a view.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# UPDATE v_upd1 SET y1 = 13 where y1 = 3;
UPDATE 1

-- Create a multi-table view.
m_db=# CREATE VIEW vv_upd AS SELECT * FROM t1, t2 WHERE x1 = x2;
CREATE VIEW

-- Update t1 through the join view.
m_db=# UPDATE vv_upd SET y1 = 1 WHERE y2 = 1;
UPDATE 1

-- Drop the schema.
m_db=# DROP SCHEMA upd_view;
NOTICE:  drop cascades to 4 other objects
DETAIL:  drop cascades to table t1
drop cascades to table t2
drop cascades to view v_upd1
drop cascades to view vv_upd
DROP SCHEMA
```

4.4.2.18.2 USE

Description

USE specifies the current schema so that the database uses the specified schema as the default (current) schema.

Precautions

To use USE to specify a schema, you must have the USAGE permission on the schema. You can run the **SHOW databases;** command to view the schemas on which you have the USAGE permission. If you do not have the USAGE permission on a schema and perform operations on the schema, the current schema is set to null and no error message is displayed. You can call the database function to check whether the specified schema takes effect.

```
m_db=> USE test;
SET
m_db=> SELECT database();
ERROR:  function returned NULL
CONTEXT:  referenced column: database
```

Syntax

```
USE schema_name
```

Parameters

schema_name

Specifies the name of the schema to be specified.

Value range: an existing schema name.

Examples

```
m_db=# SELECT database();
 database
----------
 public
(1 row)

m_db=# SHOW DATABASES;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    Database
--------------------
 information_schema
 blockchain
 db4ai
 dbe_perf
 dbe_pldebugger
 dbe_pldeveloper
 dbe_sql_util
 m_schema
 pg_catalog
 pg_toast
 pkg_service
 public
 snapshot
 sqladvisor
 sys
(16 rows)

m_db=# USE m_schema;
SET
m_db=# SELECT database();
 database
----------
 m_schema
(1 row)
```

4.4.2.19 V

4.4.2.19.1 VACUUM

Description

VACUUM recycles storage space occupied by deleted rows in tables or B-Tree indexes. In normal database operation, rows that have been deleted are not physically removed from their table; they remain present until a VACUUM is done. Therefore, it is necessary to execute VACUUM periodically, especially on frequently-updated tables.

Precautions

- If no table is specified, VACUUM processes the tables on which the user has the corresponding permission in the current database. If a table is specified, VACUUM processes only the specified table.
- To vacuum a table, you must be the owner of the table or a user granted the VACUUM permission for the specified table. When separation of duties is disabled, a system administrator has this permission by default. However, database owners are allowed to perform VACUUM on all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide VACUUM can only be executed by system administrators). VACUUM skips over any tables that the calling user does not have permission to vacuum.
- VACUUM cannot be executed inside a transaction block.
- It is recommended that production databases be vacuumed frequently (at least nightly) to ensure that dead rows are removed. It might be a good idea to always perform VACUUM ANALYZE on the affected table after adding or deleting a large number of rows. This will update the system catalogs with

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the results of all recent changes, and allow the query planner to make better choices in planning queries.

- FULL is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table. VACUUM FULL usually shrinks a table more than VACUUM does. The **FULL** option does not clear indexes. You are advised to periodically use the **REINDEX** statement. If the occupied physical space does not decrease after you run the statement, check whether there are other active transactions (that started before the data deleting transaction and has not finished before you run VACUUM FULL). If there are such transactions, wait for them to finish and run the statement again.
- VACUUM FULL returns the free space in the table to the tablespace by rebuilding the table. During the rebuilding, extra storage space equivalent to the valid data size in the table is required. For a non-segment-page table, after VACUUM FULL is executed, the physical files occupied by the source table are deleted, and the physical file space occupied by the source table is returned to the OS. For a segment-page table, after VACUUM FULL is executed, the physical space occupied by the source table is returned to the segment-page data file instead of the OS.
- VACUUM causes a significant increase in I/O traffic, which may affect the performance of other active sessions. Therefore, it is sometimes advisable to use the cost-based VACUUM delay feature.
- When **VERBOSE** is specified, VACUUM prints progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well.
- When the option list is surrounded by parentheses, the options can be written in any order. Without parentheses, the options must be specified in the order given in the syntax.
- VACUUM and VACUUM FULL clear deleted tuples after the delay specified by **vacuum_defer_cleanup_age**.
- VACUUM ANALYZE executes a VACUUM operation and then an ANALYZE operation for each selected table. This is a handy combination form for routine maintenance scripts.
- Plain VACUUM (without FULL) simply recycles space and makes it available for reuse. This form of statement can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained. VACUUM FULL executes wider processing, including moving rows across blocks to compress tables so they occupy the minimum number of disk blocks. This form is much slower and requires an exclusive lock on each table while it is being processed.
- A deadlock may occur when multiple VACUUM FULL statements are executed simultaneously.
- If the **xc_maintenance_mode** parameter is not enabled, the VACUUM FULL operation will skip all system catalogs.
- If you run **VACUUM FULL** immediately after running **DELETE**, the space will not be recycled. After running **DELETE**, execute 1000 non-SELECT transactions, or wait for 1s and then execute one transaction. Then, run **VACUUM FULL** to recycle the space.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- During the execution of VACUUM FULL, the table will be locked exclusively. You are advised not to run VACUUM FULL during peak hours because it may result in long waiting time or service interruption.

Syntax

- Recycle space and update statistics. The keyword order does not matter.
```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE  }} [,...] ) ]
   [ table_name [ (column_name [, ...] ) ] [ PARTITION ( partition_name ) ] ];
```
- Recycle space, without updating statistics information.
```
VACUUM [ FULL ] [FREEZE] [ VERBOSE ] [ table_name
[ PARTITION ( partition_name )  ] ];
```
- Recycle space and update statistics. The keywords need to be in order.
```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE } [ VERBOSE ]
   [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

Parameters

- **FULL**

  Selects "FULL" vacuum, which can recycle more space, but takes much longer and exclusively locks the table.

  NOTE

  Using FULL will result in statistics missing. To collect statistics, specify the keyword ANALYZE in the VACUUM FULL statement.

- **FREEZE**

  Specifying FREEZE is equivalent to running **VACUUM** with the **vacuum_freeze_min_age** parameter set to **0**.

- **VERBOSE**

  Prints a detailed vacuum activity report for each table.

- **ANALYZE**

  Updates statistics used by the planner to determine the most efficient way to execute a query.

  NOTE

  Ustore partitioned tables will trigger VACUUM.

- **table_name**

  Specifies the name (optionally schema-qualified) of a specific table to vacuum.

  Value range: name of a specific table to vacuum. The default value indicates all tables in the current database.

- **column_name**

  Specifies the name of the column to be analyzed. This parameter must be used together with ANALYZE.

  Value range: name of a specific column to analyze. The default value indicates all columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

> **NOTE**
>
> The mechanism of the VACUUM ANALYZE statement is to execute VACUUM and ANALYZE in sequence. Therefore, if **column_name** is incorrect, VACUUM may be successfully executed but ANALYZE may fail to be executed. For a partitioned table, ANALYZE may fail to be executed after VACUUM is successfully executed on a partition.

- **partition_name**

  Specifies the names of the level-1 table partitions to vacuum. The default value is all level-1 partitions.

Examples

```
-- Create a schema.
m_db=# CREATE SCHEMA tpcds;

-- Create a table named tpcds.reason.
m_db=# CREATE TABLE tpcds.reason
(
  r_reason_sk       integer,
  r_reason_id       character(16),
  r_reason_desc     character(100)
);

-- Insert multiple records into the table.
m_db=# INSERT INTO tpcds.reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(2,'AAAAAAAABAAAAAAA','reason 2');

-- Create an index in the tpcds.reason table.
m_db=# CREATE UNIQUE INDEX ds_reason_index1 ON tpcds.reason(r_reason_sk);

-- Vacuum the tpcds.reason table that has indexes.
m_db=# VACUUM (VERBOSE, ANALYZE) tpcds.reason;

-- Drop the index.
m_db=# DROP INDEX tpcds.ds_reason_index1 CASCADE;
m_db=# DROP TABLE tpcds.reason;
m_db=# DROP SCHEMA tpcds;
```

Suggestions

- vacuum
  - VACUUM cannot be executed inside a transaction block.
  - It is recommended that production databases be vacuumed frequently (at least nightly) to ensure that dead rows are removed. It is strongly recommended that you perform VACUUM ANALYZE on corresponding tables after adding or deleting a large number of records.
  - **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table.

4.5 Functions and Operators

This section describes the functions and operators in M-compatible databases.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTICE

The functions and operators in "SQL Reference > Functions and Operators" in *Developer Guide* are internal APIs in M-compatible databases and are prohibited. Using these functions and operators may produce unexpected results.

Functions listed in the following tables are not supported in M-compatible databases. For details, see "SQL Reference > Functions and Operators" in *Developer Guide*.

Table 4-28 Functions with the same names as those in MySQL and not supported by M-compatible databases

cot	isEmpty	json_append	json_array	json_array_append
json_array_insert	json_contains	json_contains_path	json_depth	json_extract
json_insert	json_keys	json_length	json_merge	json_merge_patch
json_merge_preserve	json_object	json_quote	json_remove	json_valid
json_replace	json_search	json_set	json_type	json_unquote
last_insert_id	md5	mod	octet_length	overlaps
point	radians	stddev_pop	stddev_samp	var_pop
var_samp	variance	lag	lead	-

Table 4-29 System information functions

adm_hist_snapshot_func	adm_hist_sqlstat_func	adm_hist_sqltext_func	session_context

Table 4-30 Statistics information functions

get_global_transactions_prepared_xacts	get_global_transactions_running_xacts

Table 4-31 System administration functions

dbe_perf.get_full_sql_by_parent_id_and_timestamp	dbe_perf.get_active_session_profile	dbe_perf.get_global_slow_sql_by_timestamp	dbe_perf.get_global_full_sql_by_timestamp

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-32 Other functions

information_schema_pg_truetypid	information_schema._pg_truetypmod	summary_create	sys_context

Table 4-33 Internal functions

db4ai.archive_snapshot	db4ai.create_snapshot	db4ai.create_snapshot_internal	db4ai.manage_snapshot_internal	db4ai.prepare_snapshot
db4ai.prepare_snapshot_internal	db4ai.publish_snapshot	db4ai.purge_snapshot	db4ai.purge_snapshot_internal	db4ai.sample_snapshot

Table 4-34 Deprecated functions

pgxc_get_stat_dirty_tables	gs_wlm_persistent_user_resource_info

Table 4-35 User-defined GaussDB functions

pg_catalog.ts_debug	pg_catalog.gs_check_logic_cluster_consistency	pg_catalog.gs_check_tables_distribution	pg_catalog.gs_wlm_persistent_user_resource_info	pg_catalog.create_wlm_instance_statistics_info
pg_catalog.pgxc_get_wlm_current_instance_info	pg_catalog.pgxc_get_wlm_history_instance_info	pg_catalog.pgxc_get_wlm_session_info_bytime	pg_catalog.create_wlm_session_info	pg_catalog.to_char
pg_catalog.to_nchar	pg_catalog.report_application_error	pg_catalog.to_text	pg_catalog.time_text	pg_catalog.timetz_text
pg_catalog.reltime_text	pg_catalog.abstime_text	pg_catalog.int1	pg_catalog.int2	pg_catalog.int4
pg_catalog.int8	pg_catalog.float4	pg_catalog.float8	pg_catalog.to_numeric	pg_catalog.to_integer
pg_catalog.rawtohex	pg_catalog.to_clob	pg_catalog.trunc	pg_catalog.to_varchar2	pg_catalog.to_interval
pg_catalog.to_ts	pg_catalog.timestamp_to_smalldatetime	pg_catalog.smalldatetime_to_timestamp	pg_catalog.smalldatetime_to_varchar2	pg_catalog.varchar2_to_smalldatetime

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

pg_catalog.bpchar_to_smalldatetime	pg_catalog.abstime_to_smalldatetime	pg_catalog.smalldatetime_to_abstime	pg_catalog.smalldatetime_to_time	pg_catalog.smalldatetime_to_timestamptz
pg_catalog.timestamptz_to_smalldatetime	pg_catalog.regexp_like	pg_catalog.intervaltonum	pg_catalog.to_nvarchar2	pg_catalog.invalidate_plsql_object
pg_catalog.table_skewness	pg_catalog.pgxc_xacts_iscommitted	pg_catalog.pgxc_parse_clog	pg_catalog.pgxc_get_senders_catchup_time	pg_catalog.global_transaction
pg_catalog.gs_stat_reset	pg_catalog.pgxc_fenced_udf_process	pg_catalog.pgxc_terminate_all_fenced_udf_process	pg_catalog.create_wlm_operator_info	pg_catalog.copy_error_log_create
pg_catalog.get_delta_info	pg_catalog.pgxc_stat_bad_block_clear	pg_catalog.pgxc_query_audit	pg_catalog.is_oid_in_group_members	pg_catalog.get_shard_oids_byname
pg_catalog.reload_active_coordinator	pg_catalog.lock_cluster_ddl	pg_catalog.unlock_cluster_ddl	pg_catalog.pgxc_get_stat_dirty_tables	pg_catalog.pgxc_get_stat_dirty_tables
pg_catalog.get_large_table_name	pg_catalog.global_segment_space_info	dbe_perf.get_global_full_sql_by_timestamp	dbe_perf.get_global_slow_sql_by_timestamp	dbe_perf.get_full_sql_by_parent_id_and_timestamp
dbe_perf.global_plancache_clean	dbe_perf.global_slow_query_info_bytime	dbe_perf.get_global_gs_asp	dbe_perf.get_datanode_active_session	dbe_perf.get_datanode_active_session_hist
dbe_perf.global_stat_activity_timeout	dbe_perf.get_active_session_profile	dbe_perf.generate_asp_report	-	-

NOTE

Functions listed in **Table 4-35** are user-defined GaussDB functions and are not recommended because their behavior in M compatibility is unknown.

4.5.1 Operator Overview

Operators are used to concatenate operands or parameters and return results. In terms of syntax, operators can appear before operands, after operands, or between two operands. Therefore, operators can be classified into unary operators and binary operators.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Unary operator
  Operator Operand
- Binary operator
  Operand 1 Operator Operand 2

Table 4-36 describes the operators supported in M-compatible databases.

Table 4-36 Operator description

Operation Type	Operator Name	Operand Type	Description
Logical operation	NOT (!)	Unary	NOT.
	AND (&&)	Binary	AND.
	OR (||)	Binary	OR.
	XOR	Binary	XOR.
Bitwise operation	&	Binary	Bitwise AND.
	|	Binary	Bitwise OR.
	^	Binary	Bitwise XOR.
	<<	Binary	Left shift.
	>>	Binary	Right shift.
	~	Unary	Bitwise NOT.
Pattern matching	[NOT] LIKE	Binary	Simple pattern matching.
Comparison operation	<	Binary	Less than.
	>	Binary	Greater than.
	<=	Binary	Less than or equal to.
	>=	Binary	Greater than or equal to.
	=	Binary	Equal to.
	<> or !=	Binary	Not equal to.
	<=>	Binary	NULL-safe equal.
	IS	Binary	Determines whether a value is equivalent to a Boolean value.
	IS NOT	Binary	Determines whether a value is not equivalent to a Boolean value (negation of IS).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operation Type	Operator Name	Operand Type	Description
	IS NOT NULL	Unary (operand operator)	Checks whether a value is not **NULL**.
	IS NULL	Unary (operand operator)	Checks whether a value is **NULL**.
Arithmetic operation	+	Unary/Binary	Unary indicates a positive number, and binary indicates addition.
	-	Unary/Binary	Unary indicates negative, and binary indicates subtraction.
	*	Binary	Multiplication operator.
	/	Binary	Division operator.
	% or MOD	Binary	Returns the remainder after integer division.
	DIV	Binary	Returns the quotient value after integer division.

Data Types Supported by Operators

Table 4-37 Data types

General Data Type	Type
Boolean Types	BOOLEAN/BOOL
Binary Types	BINARY, VARBINARY, TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB
Character Types	CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT
Date and Time Types	DATE, TIME, TIMESTAMP, DATETIME, and YEAR
Bit String Types	BIT
Numeric Types	TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, NUMERIC/DECIMAL, DEC, FIXED, FLOAT, FLOAT4, FLOAT8, DOUBLE, TINYINT UNSIGNED, SMALLINT UNSIGNED, MEDIUMINT UNSIGNED, INT/INTEGER UNSIGNED, and BIGINT UNSIGNED

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

For details about data types, see **Data Types**.

Operator Priority

Operators have higher priorities from top to bottom. Operators in the same row have the same priority. For details, see **Table 4-38**.

Table 4-38 Operator priority list

Associativity	Symbol
left	OR
left	XOR
left	AND
right	NOT
right	=
nonassoc	<, >, >=, and <=
nonassoc	LIKE
nonassoc	BETWEEN
nonassoc	IN
left	|
left	&
left	<< and >>
left	+ and -
left	*, /, %, DIV, and MOD
left	^
left	~
right	!

4.5.2 Logical Operators

M-compatible databases support four logical operators: AND (&&), OR (||), NOT (!), and XOR. For details, see **Table 4-39**. There are three types of operation results: **TRUE**, **FALSE**, and **NULL**.

Table 4-39 Logical operator details

Operator	Operand	Description
NOT/!	Unary	NOT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Operator	Operand	Description
AND/&&	Binary	AND.
OR/||	Binary	OR.
XOR	Binary	XOR.

NOT and **AND, OR, and XOR** list the calculation rules, where *a* and *b* represent logical expressions.

NOT

If the operand is **TRUE**, the calculation result is **FALSE**. If the operand is **FALSE**, the calculation result is **TRUE**.

Value range: Boolean

The following lists calculation rules.

Table 4-40 NOT calculation rules (calculation based on TRUE, FALSE, and NULL)

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	NULL

NOTE

The operators ! and NOT have the same function, but the priority of the operator ! is higher than that of NOT.

Examples:

```
m_db=# SELECT NOT TRUE;
 ?column?
----------
 f
(1 row)

m_db=# SELECT NOT FALSE;
 ?column?
----------
 t
(1 row)

m_db=# SELECT NOT NULL;
 ?column?
----------

(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

AND, OR, and XOR

Table 4-41 describes the calculation logic of AND, OR, and XOR.

Table 4-41 Operation rules of AND, OR, and XOR (calculation based on TRUE, FALSE, and NULL)

a	b	Result of a AND b	Result of a OR b	Result of a XOR b
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	NULL	NULL	TRUE	NULL
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL	NULL
NULL	NULL	NULL	NULL	NULL

NOTE

The operators AND, XOR, and OR are commutative, that is, you can switch the left and right operand without affecting the result.

Examples:

```
m_db=# SELECT TRUE AND TRUE;
 ?column?
----------
 t
(1 row)

m_db=# SELECT TRUE OR FALSE;
 ?column?
----------
 t
(1 row)

m_db=# SELECT TRUE XOR NULL;
 ?column?
----------

(1 row)
```

Specification Constraints of Logical Operators

Logical operations support all data types supported by the current M-compatible database. During data operations between different major types, the data is promoted to the major type first, and then the calculation is performed based on whether the result of the promoted major type is 0. Table 4-42 describes the promotion rules.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-42 Logical operation type classification

Operator	Left and Right Input Parameter Type	Classification Type
XOR	All types	BIGINT
AND, OR, and NOT	TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, MEDIUMINT UNSIGNED, INT/INTEGER UNSIGNED, BIGINT UNSIGNED, BIT, YEAR, and BOOL/LBOOLEAN	BIGINT
	Others	DOUBLE

4.5.3 Bitwise Operators

Bitwise operators supported by M-compatible databases include bitwise AND (&), bitwise OR (|), bitwise XOR (^), left shift (<<), right shift (>>), and bitwise exclusive (~). For details, see **Table 4-36**.

Priority: bitwise exclusive > bitwise XOR > left shift = right shift > bitwise AND > bitwise OR.

&

Description: Performs the bitwise AND operation.

Examples:

```
m_db=# SELECT 1 & 1;
 ?column?
----------
 1
(1 row)
```

|

Description: Performs the bitwise OR operation.

Examples:

```
m_db=# SELECT 1 | 1;
 ?column?
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
----------
 1
(1 row)
```

^

Description: Performs the bitwise XOR operation.

Examples:

```
m_db=# SELECT -1 ^ -1;
 ?column?
----------
 0
(1 row)
```

<<

Description: Performs the left shift operation.

Examples:

```
m_db=# SELECT 1 << 1;
 ?column?
----------
 2
(1 row)
```

>>

Description: Performs the right shift operation.

Examples:

```
m_db=# SELECT -1 >> -1;
 ?column?
----------
 0
(1 row)
```

~

Description: Performs the bitwise NOT operation.

Examples:

```
m_db=# SELECT ~1;
      ?column?
----------------------
 18446744073709551614
(1 row)
```

Specification Constraints of Bitwise Operators

Bitwise operators support all data types supported by the current M-compatible database. If none of the operands is **NULL**, the type of the final operation result is BIGINT UNSIGNED. If any operand is **NULL**, the operation result is **NULL**.

4.5.4 Pattern Matching Operators

The database provides the [NOT] LIKE operator matching and regular expression pattern matching.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

[NOT] LIKE

Description: Specifies whether a string matches the pattern string following LIKE. The **LIKE** expression returns **true** if the string matches the provided pattern. (As expected, the **NOT LIKE** expression returns **false** if the **LIKE** expression returns **true**, and vice versa.)

Matching rules:

1. This operator can succeed only when the pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign (%).
2. The underscore (_) represents (matches) any single character. The percent sign (%) indicates the wildcard character of any string.
3. To match a literal underscore or percent sign, the respective character in pattern must be preceded by the escape character. The default escape character is one backslash but a different one can be selected by using the ESCAPE clause.
4. To match with escape characters, enter two escape characters. For example, to write a pattern constant containing a backslash (\), you need to enter two backslashes in SQL statements.
5. Whether a value is case-sensitive depends on the collation. For example, the default value **utf8mb4_general_ci** is case-insensitive.
6. The operator ~~ is equivalent to LIKE and ! ~~ is equivalent to NOT LIKE.

Examples:

- During string matching, the collation is **utf8mb4_general_ci**, which is case-insensitive.

```
m_db=# CREATE TABLE t
(
    COL1 TEXT,
    COL2 TEXT,
    COL3 TEXT,
    COL4 TEXT,
    COL5 TEXT,
    COL6 TEXT
);
m_db=# INSERT INTO t VALUES
('ab^CD&eF','ab^CD&eF','ab^CD&eF','ab^CD&eF','ab^CD&eF','ab^CD&eF');
m_db=# SELECT COL1 FROM t WHERE COL1 LIKE '%B^c%';
   COL1
----------
 ab^CD&eF
(1 row)
m_db=# DROP TABLE t;
```

- For fields of the BINARY or BLOB type, the collation is **utf8mb4_bin**, which is case-sensitive.

```
m_db=# CREATE TABLE t
(
    COL1 BLOB,
    COL2 BLOB
) DISTRIBUTE BY REPLICATION;
m_db=# INSERT INTO t VALUES ('ab^CD&eF', 'ab^CD&eF');
m_db=# SELECT COL1 FROM t WHERE COL1 LIKE 'ab%';
   COL1
----------
 ab^CD&eF
(1 row)

m_db=# DROP TABLE t;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Use the escape character to match _ and %.

```
m_db=# CREATE TABLE t
(
    COL1 TEXT,
    COL2 TEXT,
    COL3 TEXT,
    COL4 TEXT,
    COL5 TEXT,
    COL6 TEXT
);
m_db=# INSERT INTO t VALUES ('ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD
%eF');
m_db=# SELECT COL1 FROM t WHERE COL1 LIKE '%B\_c%';
   COL1
----------
 ab_CD%eF
(1 row)

m_db=# DROP TABLE t;
```

- Specify the escape character.

```
m_db=# CREATE TABLE t
(
    COL1 TEXT,
    COL2 TEXT,
    COL3 TEXT,
    COL4 TEXT,
    COL5 TEXT,
    COL6 TEXT
);
m_db=# INSERT INTO t VALUES ('ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD%eF','ab_CD
%eF');
m_db=# SELECT COL1 FROM t WHERE COL1 LIKE '%B^_c%' ESCAPE '^';
   COL1
----------
 ab_CD%eF
(1 row)

m_db=# DROP TABLE t;
```

[NOT] REGEXP

Description: The REGEXP operator is used for regular expression matching. True or false is returned based on whether the pattern matches the given string, as described in **Table 4-43**.

Table 4-43 Regular expression operators

Operator Name	Description	Syntax
REGEXP	Specifies whether a string matches the regular expression.	expr REGEXP pat
NOT REGEXP	Specifies whether a string does not match the regular expression. The value is negative of REGEXP.	expr NOT REGEXP pat

Regular expression syntax:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- ^: matches the start of a string.
- $: matches the end of a string.
- .: matches any character (including the carriage return and newline characters).
- a*: matches zero or more "a" characters.
- a+: matches one or more "a" characters.
- a?: matches zero or one "a" character.
- de | abc: matches the "de" or "abc" character.
- (abc)*: matches zero or more "abc" characters.
- a{1} or a{2,3}: a{1} indicates that character "a" is repeated once, and a{2,3} indicates that character "a" is repeated twice or three times.
- [a-dX] or [^a-dX]: indicates range matching. [a-dX] matches a, b, c, d, or X. [^a-dX] matches characters that are not a, b, c, d, or X.
- [.characters.]: matches the collation used to verify elements. **Table 4-44** lists the allowed character names and matched characters.

Table 4-44 Character names and matched characters

Name	Character	Name	Character
NUL	0	SOH	1
STX	2	ETX	3
EOT	4	ENQ	5
ACK	6	BEL	7
alert	7	BS	10
backspace	\b'	HT	11
tab	\t'	LF	12
newline	\n'	VT	13
vertical-tab	\v'	FF	14
form-feed	\f'	CR	15
carriage-return	\r'	SO	16
SI	17	DLE	20
DC1	21	DC2	22
DC3	23	DC4	24
NAK	25	SYN	26
ETB	27	CAN	30
EM	31	SUB	32
ESC	33	IS4	34

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Character	Name	Character
FS	34	IS3	35
GS	35	IS2	36
RS	36	IS1	37
US	37	space	'
exclamation-mark	!'	quotation-mark	"'
number-sign	#'	dollar-sign	$'
percent-sign	%'	ampersand	&'
apostrophe	\''	left-parenthesis	('
right-parenthesis	)'	asterisk	*'
plus-sign	+'	comma	,'
hyphen	-'	hyphen-minus	-'
period	.'	full-stop	.'
slash	/'	solidus	/'
zero	0'	one	1'
two	2'	three	3'
four	4'	five	5'
six	6'	seven	7'
eight	8'	nine	9'
colon	:'	semicolon	;'
less-than-sign	<'	equals-sign	='
greater-than-sign	>'	question-mark	?'
commercial-at	@'	left-square-bracket	['
backslash	\\'	reverse-solidus	\\'
right-square-bracket	]'	circumflex	^'
circumflex-accent	^'	underscore	_'
low-line	_'	grave-accent	`'
left-brace	{'	left-curly-bracket	{'

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Character	Name	Character
vertical-line	|'	right-brace	}'
right-curly-bracket	}'	tilde	~'
DEL	177	-	-

- [=character_class=]: specifies the equivalence class. A character matches all characters with the same sort proofreading value, including itself. For example, if o and (+) are of the same type, [[=o=]], [[=(+)=]], and [o(+)] are synonyms.
- [:character_class:]: matches all character classes that belong to the class, as described in **Table 4-45**.

Table 4-45 Description of standard class names

Character Class	Description	Character Range
alnum	Alphanumeric numeric character.	0–9, a–z, and A–Z
alpha	Alphanumeric character.	a–z and A–Z
blank	Blank character.	Blank character[\t]
cntrl	Control character.	[\x01-\x1F]
digit	Digit character.	[0-9]
graph	Graphic character.	[^\x01-\x20]
lower	Lowercase character.	[a-z]
print	Graphic character.	[^\x01-\x20]
punct	Punctuation character.	[-!"#$%&'( )*+,./:;<=>?@[\ \]^_`{|}~]
space	Spaces, tabs, new lines, and carriage returns.	[\n\r\t\x0B]
upper	Uppercase character.	[A-Z]
xdigit	Hexadecimal numeric character.	[0-9a-fA-F]

- [[:<:]] and [[:>:]]: matches the start and end of a string respectively.

Examples:

```
m_db=# SELECT 'fo\nfo' REGEXP '^fo$';
 ?column?
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
----------
 f
(1 row)

m_db=# SELECT 'fofo' REGEXP '^fo';
 ?column?
----------
 t
(1 row)

m_db=# SELECT 'Ban' REGEXP '^Ba*n';
 ?column?
----------
 t
(1 row)

m_db=# SELECT 'Bn' REGEXP '^Ba?n';
 ?column?
----------
 t
(1 row)

m_db=# SELECT 'pi' REGEXP 'pi|apa';
 ?column?
----------
 t
(1 row)

m_db=# SELECT 'aXbc' NOT REGEXP '^[a-dXYZ]$';
 ?column?
----------
 t
(1 row)
```

When **enable_escape_string** is enabled for **m_format_behavior_compat_options**, the escape characters described in **Table 4-46** are supported.

Table 4-46 Escape characters

Escape Character	Description
\'	A single quotation mark.
\"	A double quotation mark.
\b	A backspace.
\n	A newline.
\r	A carriage return.
\t	A tab.
\Z	ASCII 26 (Control Z).
\\	A backslash.
\%	A % character.
_	An underscore.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Escape Character	Description
\0	GaussDB reports the error: invalid byte sequence for encoding "UTF8": 0x00.

When **standard_conforming_strings** is set to **'off'**, **escape_string_warning** is set to **'off'**, and **backslash_quote** is set to **'on'**, escape characters listed in **Table 4-47** are supported.

Table 4-47 Escape Character

Escape Character	Description
\'	A single quotation mark.
\b	Backspace.
\f	Form feed, for example, C language.
\n	Newline, for example, C language.
\r	Carriage return, for example, C language.
\t	Horizontal tab, for example, C language.
\uwxyz	Character whose hexadecimal value is **0xwxyz**, where **wxyz** is four hexadecimal digits.
\xhhh	Character whose hexadecimal value is **0xhhh**, where **hhh** is any sequence of hexadecimal digits.
\0	GaussDB reports the error: invalid byte sequence for encoding "UTF8": 0x00.
\xy	Character whose octal value is **0xy**, where **xy** is two octal digits.
\xyz	Character whose octal value is **0xyz**, where **xyz** is three octal digits.

NOTICE

The lexical syntax is parsed based on byte streams. If a multi-byte character contains code that is consistent with symbols such as '\', '\'', and '\\', the behavior of the multi-byte character is inconsistent with that in MySQL. In this case, you are advised to disable the escape character function temporarily.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.5.5 Comparison Functions and Comparison Operators

Comparison operators are used to compare the sizes of two pieces of data and return a value of the Boolean type.

Comparison operators are binary operators. For details, see **Table 4-36**. For details about the comparison operators provided in M-compatible databases, see **Table 4-36**.

NOTE

- The calculation priority of the inequality sign is higher than that of the equality sign.
- In M-compatible databases, only B-tree and UB-tree indexes are supported.
- <> and <=> do not support index scan.
- <=> does not support hash join or merge join.
- <=> does not support row comparison.
- <, <=, =, >=, >, and <> operators support comparison between row expressions and NULL values.
- Row expressions can be used as parameters for the IS NULL and ISNULL operations.

```
m_db=# SELECT (1,2) <=> row(2,3);
ERROR:  could not determine interpretation of row comparison operator <=>
LINE 1: select (1,2) <=> row(2,3);
                     ^
HINT:  unsupported operator.
m_db=# SELECT (1,2) < NULL;
 ?column?
----------

(1 row)

m_db=# SELECT (1,2) <> NULL;
 ?column?
----------

(1 row)
m_db=# SELECT (1, 2)  IS NULL;
 ?column?
----------
 f
(1 row)

m_db=# SELECT ISNULL((1, 2));
 ?column?
----------
 f
(1 row)
```

Comparison Operators

- <

  Description: Performs the "less than" operation.

  Examples:

  ```
  m_db=# SELECT 2 < 2;
   ?column?
  ----------
   f
  (1 row)
  ```

- >

  Description: "greater than" operator.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT 2 > 2;
 ?column?
----------
 f
(1 row)
```

- <=

  Description: "less than or equal to" operator.

  Examples:

  ```
  m_db=# SELECT 0.1 <= 0.2;
   ?column?
  ----------
   t
  (1 row)
  ```

- >=

  Description: "greater than or equal to" operator.

  Examples:

  ```
  m_db=# SELECT 2 >= 2;
   ?column?
  ----------
   t
  (1 row)
  ```

- =

  Description: "equal to" operator.

  Examples:

  ```
  m_db=# SELECT 2 = 2;
   ?column?
  ----------
   t
  (1 row)
  ```

- <> or !=

  Description: "not equal to" operator.

  Examples:

  ```
  m_db=# SELECT 1 <> 2;
   ?column?
  ----------
   t
  (1 row)
  ```

- <=>

  Description: NULL-safe equal operator. It performs an equality comparison like the = operator, but returns **t** rather than **f** if both operands are NULL, and **f** rather than **t** if one operand is NULL.

  Examples:

  ```
  m_db=# SELECT 1 <=> 1, 1 <=> NULL, NULL <=> NULL;
   ?column? | ?column? | ?column?
  ----------+----------+----------
   t        | f        | t
  (1 row)
  ```

- IS

  ```
  expr IS boolean_value
  ```

  Description: Checks whether **expr** is equivalent to **boolean_value**. If they are equivalent, **TRUE (t)** is returned. Otherwise, **FALSE (f)** is returned. **boolean_value** can be **TRUE**, **FALSE**, **NULL**, or **UNKNOWN**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
 is true | is false | ?column?
---------+----------+----------
 t       | t        | t
(1 row)
```

- IS NOT

```
expr IS NOT boolean_value
```

Description: Checks whether **expr** is not equivalent to **boolean_value**. If **expr** is not equivalent to **boolean_value**, **TRUE (t)** is returned. Otherwise, **FALSE (f)** is returned. **boolean_value** can be **TRUE**, **FALSE**, **NULL**, or **UNKNOWN**.

Examples:

```
m_db=# SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
 ?column? | ?column? | ?column?
----------+----------+----------
 t        | t        | f
(1 row)
```

- IS NOT NULL

```
expr IS NOT NULL
```

Description: Checks whether **expr** is not **NULL**. If so, true **t** is returned. If not, false **f** is returned.

Examples:

```
m_db=# SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
 ?column? | ?column? | ?column?
----------+----------+----------
 t        | t        | f
(1 row)
```

- IS NULL

```
expr IS NULL
```

Description: Checks whether **expr** is **NULL**. If so, true **t** is returned. If not, false **f** is returned.

Examples:

```
m_db=# SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
 ?column? | ?column? | ?column?
----------+----------+----------
 f        | f        | t
(1 row)
```

Comparison Functions

- COALESCE(value1, value2, ...)

Description: Returns the first non-null value in the parameter list. If there is no non-null value, the function returns **NULL**.

Return value type: Depends on the type derivation matrix. The return value type is determined by the input parameter type.

Examples:

```
m_db=# SELECT COALESCE(NULL, 1);
 coalesce
----------
        1
(1 row)

m_db=# SELECT COALESCE(NULL, NULL, NULL);
 coalesce
----------

(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- GREATEST(value1, value2, ...)

  Description: Returns the maximum value in the parameter list.

  Parameters: At least two parameters are required.

  Example:

```
m_db=# SELECT GREATEST(34.0, 3.0, 5.0, 767.0);
 greatest
----------
    767.0
(1 row)

m_db=# SELECT GREATEST('B', 'A', 'C');
 greatest
----------
 C
(1 row)
```

- INTERVAL(N, N1, N2, N3, ...)

  Description: Returns the result of comparison between N and a list (N1, N2, N3, ...) consisting of other parameters.

  Parameter: At least two parameters are required. All input parameters are considered as numeric types.

  Return value type: SMALLINT

  – If N is **NULL**, **-1** is returned.
  – If the value of N is less than N1, **0** is returned.
  – If there is a positive integer j, and for any positive integer i≤j, there is $N_i \le N \le N_{j+1}$, and **j** is returned.

  Examples:

```
m_db=# SELECT INTERVAL(24, 2, 5, 14, 26, 28, 46);
 interval
----------
        3
(1 row)

m_db=# SELECT INTERVAL(24, 29, 5, 14);
 interval
----------
        0
(1 row)
```

- ISNULL(expr)

  Description: If parameter **expr** is **NULL**, **t** is returned. Otherwise, **f** is returned.

  Parameter: any input parameter.

  Return type: Boolean

  Example:

```
m_db=# SELECT ISNULL('2023-01-01');
 isnull
--------
 f
(1 row)

m_db=# SELECT ISNULL(NULL);
 isnull
--------
 t
(1 row)
```

- LEAST(value1, value2, ...)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Returns the minimum value in the parameter list.

Parameter: At least two parameters are required.

Examples:

```
m_db=# SELECT LEAST(34.0, 3.0, 5.0, 767.0);
 least
-------
   3.0
(1 row)

m_db=# SELECT LEAST('B', 'A', 'C');
 least
-------
 A
(1 row)
```

4.5.6 Numeric Operation Functions and Arithmetic Operators

Arithmetic Operators

Arithmetic operators in M-compatible databases include +, -, *, /, %, - (negative operation), and DIV. For details, see **Table 4-36**.

- +

  Description: Performs the addition operation.

  Example:

  ```
  /* plus constant */
  m_db=# SELECT 2+3;
   ?column?
  ----------
         5
  (1 row)

  m_db=# SELECT +3;
   ?column?
  ----------
         3
  (1 row)
  ```

- -

  Description: Performs the subtraction operation or represents a negative number.

  NOTE

  If both operands are integers and either of them is unsigned, the result is an unsigned integer. For subtraction, if the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is a signed number even if any operand is an unsigned number.

  Example:

  ```
  /* minus constant */
  m_db=# SELECT 3-2;
   ?column?
  ----------
         1
  (1 row)

  m_db=# SELECT -3;
   ?column?
  ----------
        -3
  (1 row)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- *

  Description: Performs the multiplication operation.

  Example:

```
/* multiply constant */
m_db=# SELECT 2*3;
 ?column?
----------
        6
(1 row)

m_db=# SELECT -2*3;
 ?column?
----------
       -6
(1 row)
```

- /

  Description: Performs the division operation.

  NOTE

  When / is used for division, the number of decimal places in the result is the number of decimal places in the first operand plus the value of **div_precision_increment**. The default value is **4**. For example, the result of expression 8.85/0.093 has six decimal places (95.161290).

  Example:

```
/* division constant */
m_db=# SELECT 4/2;
 ?column?
----------
  2.0000
(1 row)

m_db=# SELECT 8.85/0.093;
 ?column?
------------
 95.161290
(1 row)
```

- % or MOD

  Description: Performs the modulo operation.

  Example:

```
/* mod constant */
m_db=# SELECT 7%4;
 ?column?
----------
        3
(1 row)

m_db=# SELECT 7 MOD 4;
 ?column?
----------
        3
(1 row)
```

- DIV

  Description: Performs the exact division operation.

  Example:

```
/* DIV test */
m_db=# SELECT 2.1 DIV 1;
 ?column?
----------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
      2
(1 row)

m_db=# SELECT 1 DIV 2;
 ?column?
----------
      0
(1 row)
```

Specification Constraints of Arithmetic Operators

Operators support operations between different data types. Before calculating results, they first promote data to appropriate types. **Table 4-48** describes the operand promotion rules and result type specifications associated with operators.

Table 4-48 Operand promotion rules for operators

Left and Right Input Parameter Type	Classification Type
TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, and DATE	BIGINT
TINYINT UNSIGNED, SMALLINT UNSIGNED, MEDIUMINT UNSIGNED, INT/INTEGER UNSIGNED, BIGINT UNSIGNED, BIT, and YEAR	BIGINT UNSIGNED
TIME, TIMESTAMP, and DATETIME	BIGINT (without milliseconds) NUMERIC (with milliseconds)
UNKNOWN, CHAR, VARCHAR, BINARY, VARBINARY, TINYTEXT TEXT, MEDIUMTEXT, and LONGTEXT TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB, FLOAT, FLOAT4, and FLOAT8	FLOAT8
NUMERIC	NUMERIC
Use the default settings of other parameters.	BIGINT

Table 4-49 Addition, multiplication, and modulo result types

Classification Type Combination (from Top to Bottom in Descending Order of Priority)	Final Result Type
FLOAT8 + any	FLOAT8
NUMERIC + any	NUMERIC
BIGINT UNSIGNED + any	BIGINT UNSIGNED
Others	BIGINT

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-50 Subtraction result types

Classification Type Combination (from Top to Bottom in Descending Order of Priority)	Final Result Type
FLOAT8 + any	FLOAT8
NUMERIC + any	NUMERIC
BIGINT UNSIGNED + any	BIGINT UNSIGNED or BIGINT (For GUC parameter **SQL_MODE**, if NO_UNSIGNED_SUBTRACTION is enabled, the result type is BIGINT. Otherwise, BIGINT UNSIGNED is returned.)
Others	BIGINT

Table 4-51 Division result types

Classification Type Combination (from Top to Bottom in Descending Order of Priority)	Final Result Type
FLOAT8 + any	float8
Others	numeric

Table 4-52 Negative result types

Input Parameter Type	Result Type
TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, TINYINT UNSIGNED, SMALLINT UNSIGNED, MEDIUMINT UNSIGNED, INT/INTEGER UNSIGNED, BIGINT UNSIGNED, BIT, and YEAR	BIGINT
UNKNOWN, CHAR, VARCHAR, BINARY, VARBINARY, TINYTEXT TEXT, MEDIUMTEXT, and LONGTEXT TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, DATE, DATETIME, TIMESTAMP, TIME, FLOAT, FLOAT4, and FLOAT8	FLOAT8
NUMERIC	NUMERIC
Use the default settings of other parameters.	FLOAT8

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-53 Exact division result types

Input Parameter Type	Result Type
TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, and BIGINT TINYINT UNSIGNED, SMALLINT UNSIGNED, MEDIUMINT UNSIGNED, INT/INTEGER UNSIGNED, BIGINT UNSIGNED, BIT, and YEAR	BIGINT (Convert the input parameter to the BIGINT type, perform the division operation, round the result, and return the BIGINT type.)
UNKNOWN, CHAR, VARCHAR, BINARY, VARBINARY, TINYTEXT TEXT, MEDIUMTEXT, and LONGTEXT TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, DATE, DATETIME, TIMESTAMP, TIME, FLOAT, FLOAT4, FLOAT8, and NUMERIC	BIGINT (Convert the input parameter to the DOUBLE type, perform the division operation, round the result, and return the BIGINT type.)

NOTE

When the operand of an arithmetic operator is a function whose return value is of the datetime, timestamp, or time type, or a literal value of the datetime or time type, for example, timestamp '2000-01-01 00:00:00' or time '08:00:00', the content after the decimal point is ignored, that is, the millisecond information is ignored.

Numeric Operation Functions

- ABS(expr)

  Description: Returns the absolute value of a numeric expression.

  Parameter: **expr**, which is a numeric expression.

  Return type: BIGINT UNSIGNED, BIGINT, DOUBLE, or DECIMAL. The data type is the same as that of the numeric expression.

  Example:

```
m_db=# SELECT ABS(-1.3);
 abs
-----
 1.3
(1 row)
```

- ACOS(X)

  Description: Returns the arc cosine value of X, that is, the value whose cosine is X.

  If the value of parameter **X** is not in the range [-1, 1], the function returns **NULL**.

  Return type: DOUBLE.

  Example:

```
m_db=# SELECT ACOS(0);
       acos
--------------------
 1.5707963267948966
(1 row)
```

- ASIN(X)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Returns the arc sine value of X, that is, the value whose sine is X.

If the value of parameter **X** is not in the range [-1, 1], the function returns **NULL**.

Return type: DOUBLE.

Example:

```
m_db=# SELECT ASIN(0);
 asin
------
    0
(1 row)
```

- ATAN(X)

  Description: Returns the arc tangent value of X, that is, the value whose tangent value is X.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT ATAN(-10);
          atan
  ---------------------
   -1.4711276743037347
  (1 row)
  ```

- ATAN(Y, X)

  Description: Returns the arc tangent value of X and Y. It is similar to calculating the arc tangent value of Y/X. The symbols of the two parameters are used to determine the quadrant where the result is located.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT ATAN(-2, 2);
          atan
  ---------------------
   -0.7853981633974483
  (1 row)
  ```

- ATAN2(Y, X)

  Description: This function has the same function and usage as **ATAN(Y, X)**.

  Example:

  ```
  m_db=# SELECT ATAN2(-2, 2);
          atan2
  ---------------------
   -0.7853981633974483
  (1 row)
  ```

- CEILING(X)

  Description: Returns the smallest integer greater than or equal to X.

  Return type: BIGINT, DOUBLE, or DECIMAL.

  Example:

  ```
  m_db=# SELECT CEILING(-5.5);
   ceiling
  ---------
        -5
  (1 row)
  ```

- COS(X)

  Description: Returns the cosine value of the specified radian X.

  Return type: DOUBLE.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Example:

```
m_db=# SELECT COS(0);
 cos
-----
   1
(1 row)
```

- DEGREES(X)

  Description: Converts parameter **X** from a radian to a degree and returns the value.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT DEGREES(PI());
   degrees
  ---------
      180
  (1 row)
  ```

- EXP(X)

  Description: Returns the X^{th} power of the natural base e (2.7182818...).

  Return type: DOUBLE.

  ```
  m_db=# SELECT EXP(2);
         exp
  ------------------
   7.38905609893065
  (1 row)
  ```

- FLOOR(X)

  Description: Returns the largest integer less than or equal to X.

  Return type: BIGINT UNSIGNED, BIGINT, DOUBLE, or DECIMAL.

  Example:

  ```
  m_db=# SELECT FLOOR(5.5);
   floor
  -------
      5
  (1 row)
  ```

- LN(X)

  Description: Returns the natural logarithm of X, that is, the logarithm of X to base e (2.7182818...).

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT LN(2);
          ln
  --------------------
   0.6931471805599453
  (1 row)
  ```

- LOG([B,]X)

  Description: Returns the logarithm of X to base B. When parameter **B** is set to the default value, e (2.7182818...) is used as the base.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT LOG(2);
         log
  --------------------
   0.6931471805599453
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(1 row)

m_db=# SELECT LOG(10, 2);
        log
--------------------
 0.30102999566398114
(1 row)
```

- LOG10(X)

  Description: Returns the logarithm of X to base 10.

  Return type: DOUBLE.

```
m_db=# SELECT LOG10(2);
       log10
--------------------
 0.3010299956639812
(1 row)
```

- LOG2(X)

  Description: Returns the logarithm of X to base 2.

  Return type: DOUBLE.

```
m_db=# SELECT LOG2(2);
 log2
------
    1
(1 row)
```

- PI()

  Description: Returns the value of π (circumference).

  Return type: DOUBLE.

  Example:

```
m_db=# SELECT PI();
        pi
-------------------
 3.141592653589793
(1 row)
```

- POW(X, Y)

  Description: Returns X raised to the power of Y.

  Return type: DOUBLE.

  Example:

```
m_db=# SELECT POW(2, 3);
 pow
-----
   8
(1 row)
```

- POWER(X, Y)

  Description: Returns X raised to the power of Y. This function is the alias of the **POW(X, Y)** function.

  Return type: DOUBLE.

  Example:

```
m_db=# SELECT POWER(2, 3);
 power
-------
     8
(1 row)
```

- RAND([seed])

  Description: Returns the random floating-point number. The value range is [0, 1).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter: **seed**, which is a random number seed. If the same seed value is specified, the function generates the same random number.

Return type: DOUBLE.

Example:

```
m_db=# SELECT RAND();
        rand
--------------------
 0.5320404642261565
(1 row)

m_db=# SELECT RAND(1);
        rand
---------------------
 0.40540353712197724
(1 row)
```

- SIGN(X)

  Description: Returns the value corresponding to the symbol of X. That is, **1** is returned for a positive value, **0** is returned for a zero value, and **-1** is returned for a negative value.

  Return type: BIGINT.

  Example:

  ```
  m_db=# SELECT SIGN(5.2);
   sign
  ------
     1
  (1 row)

  m_db=# SELECT SIGN(0);
   sign
  ------
     0
  (1 row)

  m_db=# SELECT SIGN(-5.2);
   sign
  ------
    -1
  (1 row)
  ```

- SIN(X)

  Description: Returns the sine value of the specified radian X.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT SIN(PI());
           sin
  ------------------------
   1.2246467991473532e-16
  (1 row)
  ```

- SQRT(X)

  Description: Returns the square root of X.

  Return type: DOUBLE.

  Example:

  ```
  m_db=# SELECT SQRT(0.64);
   sqrt
  ------
    0.8
  (1 row)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- TAN(X)

  Description: Returns the tangent value of the specified radian X.

  Return type: DOUBLE.

  Example:

```
m_db=# SELECT TAN(PI());
          tan
-------------------------
 -1.2246467991473532e-16
(1 row)
```

- TRUNCATE(X, D)

  Description: Returns the result of **X** with **D** decimal places.

  Parameter: If parameter **D** is **0**, the integer part of X is returned. If parameter **D** is a negative number, **D** places to the left of the decimal point in **X** will be adjusted to **0**.

  Return type: BIGINT, DOUBLE, or DECIMAL.

  Example:

```
m_db=# SELECT TRUNCATE(3.2312, 1);
 truncate
----------
      3.2
(1 row)

m_db=# SELECT TRUNCATE(123.2312, -1);
 truncate
----------
      120
(1 row)
```

Extended Scenarios

- Precedence of arithmetic operations: Negation > Multiplication = Division = Modulo = Exact division > Addition = Subtraction.

  Example: The multiplication operation takes precedence over the addition operation.

```
m_db=# SELECT 1 + 2 * 3;
 ?column?
----------
        7
(1 row)
```

  Example: The modulo operator (%) takes precedence over the addition operation.

```
m_db=# SELECT 1 + 2 % 3;
 ?column?
----------
        3
(1 row)
```

  Example: The modulo operation (MOD) takes precedence over the addition operation.

```
m_db=# SELECT 1 + 2 MOD 3;
 ?column?
----------
        3
(1 row)
```

  Example: The exact division operation takes precedence over the addition operation.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT 1 + 7 DIV 3;
 ?column?
----------
        3
(1 row)
```

4.5.7 String Functions

M-compatible databases provide string functions to process strings.

- For a function that operates on the position of a string, the number of the first position is 1.
- For functions that use the length as the parameter, the non-integer parameter is rounded off to the nearest integer.
- If the character set of the database is SQL_ASCII, the database interprets byte values 0 to 127 according to the ASCII standard, and byte values 128 to 255 are regarded as characters that cannot be parsed. If SQL_ASCII is used, the database cannot convert or verify non-ASCII characters. As a result, string functions such as CHAR_LENGTH, FIND_IN_SET, LEFT, LOWER, REVERSE, RIGHT, STRCMP, and TRIM generate unexpected results. If any non-ASCII data is used, you are advised not to set the character set of the database to SQL_ASCII.

ASCII

ASCII(str)

Description: Returns the ASCII code of the leftmost character of **str**. If **str** is empty, **0** is returned. If **str** is **NULL**, **NULL** is returned.

Return type: INT

Examples:

```
m_db=# SELECT ASCII('');
 ascii
-------
     0
(1 row)

m_db=# SELECT ASCII('abc');
 ascii
-------
    97
(1 row)

m_db=# SELECT ASCII(NULL);
 ascii
-------

(1 row)
```

BIT_LENGTH

BIT_LENGTH(str)

Description: Returns the bit length of **str**.

Return type: BIGINT

Examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT BIT_LENGTH('text');
 bit_length
------------
         32
(1 row)
```

CHAR_LENGTH

CHAR_LENGTH(str)

Description: Returns the number of characters contained in **str**. A multi-byte character is counted as one character. For example, if you enter two 2-byte characters, **LENGTH()** returns **4**, and **CHAR_LENGTH()** returns **2**.

Return type: BIGINT

Examples:

```
m_db=# SELECT CHAR_LENGTH('text');
 char_length
-------------
           4
(1 row)
```

CHARACTER_LENGTH

CHARACTER_LENGTH(str)

Description: The function and usage are the same as those of **CHAR_LENGTH**.

Return type: BIGINT

Examples:

```
m_db=# SELECT CHARACTER_LENGTH('text');
 character_length
------------------
                4
(1 row)
```

CONCAT

CONCAT(str1, str2, ..., strN)

Description: Concatenates *N* character strings based on the input parameter sequence and returns the concatenated character string. If any string is **NULL**, **NULL** is returned.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT CONCAT('hello', 'world', '!');
   concat
-------------
 helloworld!
(1 row)

m_db=# SELECT CONCAT('hello', NULL, '!');
 concat
--------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(1 row)
```

CONCAT_WS

CONCAT_WS(separator, str1, str2, ..., strN)

Description: Concatenates multiple strings in sequence and separates adjacent strings using **separator**. If **separator** is **NULL**, **NULL** is returned. If the string to be concatenated is **NULL**, it is ignored.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT CONCAT_WS(',', 'hello', NULL, 'world', '!');
   concat_ws
---------------
 hello,world,!
(1 row)

m_db=# SELECT CONCAT_WS(NULL, 'hello', NULL, 'world', '!');
 concat_ws
-----------

(1 row)
```

FIELD

FIELD(str, str1, str2, str3, ...)

Description: The field function returns the position of **str** in the {str, str1, str2, str3, ...} list. The position increases from 1. If **0** is returned, **str** is not in the {str, str1, str2, str3, ...} list.

Return type: BIGINT

- If **str** is **NULL** or is not in the {str, str1, str2, str3, ...} list, **0** is returned.
- If the input parameters of the function are all digits, the comparison is performed based on digits. If the input parameters are all non-digits, the comparison is performed based on character strings. If the input parameters contain both digits and non-digits, the comparison is performed based on the DOUBLE type.

Examples:

```
m_db=# SELECT FIELD(1,2,1);
 field
------
     2
(1 row)

m_db=# SELECT FIELD('a','b','a');
 field
------
     2
(1 row)

m_db=# SELECT FIELD('a',1,'b','a');
WARNING:  Truncated incorrect double value: 'a'
CONTEXT:  referenced column: field
WARNING:  Truncated incorrect double value: 'b'
CONTEXT:  referenced column: field
 field
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

FIND_IN_SET

FIND_IN_SET(str, strlist)

Description: Returns the position of **str** in **strlist**, starting from 1 in ascending order. The string list **strlist** is a string that contains multiple sub-strings. The sub-strings are separated by commas (,).

Return type: INT

- If **strlist** does not contain **str**, **0** is returned.
- If **str** or **strlist** is **NULL**, **NULL** is returned.
- If **str** contains commas (,), **0** is returned.

Examples:

```
m_db=# SELECT FIND_IN_SET('e', 'a,b,c;d,e');
 find_in_set
-------------
           4
(1 row)

m_db=# SELECT FIND_IN_SET('a,', 'a,b,c;d,e');
 find_in_set
-------------
           0
(1 row)

m_db=# SELECT FIND_IN_SET(NULL, 'a,b,c;d,e');
 find_in_set
-------------

(1 row)
```

HEX

- HEX(str)

  Description: Converts **str** into a hexadecimal string. Each byte of each character in **str** is converted into two hexadecimal digits. The reverse operation is **UNHEX**.

  Return value type: TEXT

  Examples:

  ```
  m_db=# SELECT HEX('abc'), UNHEX(HEX('abc'));
    hex   | unhex
  --------+-------
   616263 | abc
  (1 row)
  ```

- HEX(N)

  Description: Converts the number *N* into a hexadecimal string.

  Return value type: TEXT

  Examples:

  ```
  m_db=# SELECT HEX(255);
   hex
  -----
   FF
  (1 row)
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

LCASE

LCASE(str)

Description: Converts uppercase characters in a string to lowercase letters. The functions and usage of LCASE are the same as those of **LOWER**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT LCASE('Hello,world!');
    lcase
--------------
 hello,world!
(1 row)
```

LEFT

LEFT(str, len)

Description: Returns the leftmost **len** characters of **str**. If **str** or **len** is **NULL**, **NULL** is returned. The number of characters is specified by **len**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT LEFT('abcdef', 3);
 left
------
 abc
(1 row)

m_db=# SELECT LEFT('abcdef', NULL);
 left
------

(1 row)

m_db=# SELECT LEFT('abcdef', 11);
  left
--------
 abcdef
(1 row)
```

LENGTH

LENGTH(str)

Description: Returns the number of bytes contained in **str**. A multi-byte character is counted as multiple bytes. For example, if you enter two 2-byte characters, **LENGTH()** returns **4**, and **CHAR_LENGTH()** returns **2**.

Return type: BIGINT

Examples:

```
m_db=# SELECT LENGTH('text');
 char_length
-------------
           4
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

LOWER

LOWER(str)

Description: Converts uppercase letters in a string to lowercase letters. The function and usage of LOWER are the same as those of **LCASE**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT LOWER('Hello,world!');
    lower
--------------
 hello,world!
(1 row)
```

LPAD

LPAD(str, len, padstr)

Description: Uses **padstr** to pad **str** from the left until the length is the value of **len**. The length of the return value is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is as follows: Min(max_allowed_packet/Maximum length of a single character in the character set returned by the function, 1073741819).

Parameters: **Table 4-54** describes the parameters.

Table 4-54 LPAD parameters

Parameter	Type	Description
str	TEXT	String to be padded.
len	INT	Length of the string after padding.
padstr	TEXT	String used for padding.

Return value type: TEXT or LONGBLOB

- If the length of **str** is greater than **len**, **len** characters on the left of **str** are returned.
- If **str**, **len**, or **padstr** is **NULL**, **NULL** is returned.
- If the value of **len** is negative, **NULL** is returned.

Examples:

```
m_db=# SELECT LPAD('hello', 10, 'abc');
    lpad
------------
 abcabhello
(1 row)

m_db=# SELECT LPAD('hello', 3, 'abc');
 lpad
------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 hel
(1 row)

m_db=# SELECT LPAD('hello', 10, NULL);
 lpad
------

(1 row)

m_db=# SELECT LPAD('hello', -10, 'abc');
 lpad
------

(1 row)
```

LTRIM

LTRIM(str)

Description: Deletes spaces on the left of **str**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT LTRIM('  hello   ');
  ltrim
----------
 hello
(1 row)
```

REPEAT

REPEAT(str, count)

Description: Returns a string consisting of **str** repeated **count** times. If **count** is less than or equal to **0**, **NULL** is returned. If **str** or **count** is **NULL**, **NULL** is returned. The length of the return value is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is as follows: Min(max_allowed_packet, 1073741819).

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT REPEAT('abc', 3);
  repeat
-----------
 abcabcabc
(1 row)

m_db=# SELECT REPEAT('abc', NULL);
 repeat
--------

(1 row)

m_db=# SELECT REPEAT('abc', -1);
 repeat
--------

(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

REPLACE

REPLACE(str, from_str, to_str)

Description: Replaces the string **from_str** in the string **str** with the string **to_str**. Case sensitivity is required. If any parameter is **NULL**, **NULL** is returned. The length of the return value is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is as follows: Min(max_allowed_packet, 1073741819).

Parameters: **Table 4-55** describes the parameters.

Table 4-55 REPLACE parameters

Parameter	Type	Description
str	TEXT	String to be replaced.
from_str	TEXT	Target search.
to_str	TEXT	Replaced to.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT REPLACE('abc,efg,hijk,abcde,dabc,e', 'abc', 'xyz');
          replace
----------------------------
 xyz,efg,hijk,xyzde,dxyz,e
(1 row)

m_db=# SELECT REPLACE('abc,efg,hijk,abcde,dabc,e', NULL, 'xyz');
 replace
---------

(1 row)
```

REVERSE

REVERSE(str)

Description: Returns **str** in reverse order.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT REVERSE('abcd');
 reverse
---------
 dcba
(1 row)
```

RIGHT

RIGHT(str, len)

Description: Returns the rightmost **len** characters of **str**. If **str** or **len** is **NULL**, **NULL** is returned.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT RIGHT('abcdef', 4);
 right
-------
 cdef
(1 row)

m_db=# SELECT RIGHT('abcdef', NULL);
 right
-------

(1 row)
```

RPAD

RPAD(str, len, padstr)

Description: Uses **padstr** to pad **str** on the right until the length is **len**. The length of the return value is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is as follows: Min(max_allowed_packet/Maximum length of a single character in the character set returned by the function, 1073741819).

Parameters: **Table 4-56** describes the parameters.

Table 4-56 RPAD parameters

Parameter	Type	Description
str	TEXT	String to be padded.
len	INT	Length of the string after padding.
padstr	TEXT	String used for padding.

Return value type: TEXT or LONGBLOB

- If the length of **str** is greater than **len**, **len** characters on the left of **str** are returned.
- If **str**, **len**, or **padstr** is **NULL**, **NULL** is returned.
- If the value of **len** is negative, **NULL** is returned.

Examples:

```
m_db=#  SELECT RPAD('hello', 10, 'abc');
    rpad
------------
 helloabcab
(1 row)

m_db=# SELECT RPAD('hello', 3, 'abc');
 rpad
------
 hel
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT RPAD('hello', 10, NULL);
 rpad
------

(1 row)

m_db=# SELECT RPAD('hello', -10, 'abc');
 rpad
------

(1 row)
```

RTRIM

RTRIM(str)

Description: Deletes spaces on the right of **str**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT RTRIM('  hello   ');
  rtrim
---------
   hello
(1 row)
```

SPACE

SPACE(N)

Description: Returns a string consisting of *N* spaces. If *N* is less than or equal to **0**, an empty string is returned. The length of the return value is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is as follows: Min(max_allowed_packet, 1073741819).

Return value type: TEXT

Examples:

```
m_db=# SELECT CONCAT('a', SPACE(6), 'b');
  concat
----------
 a      b
(1 row)

m_db=# SELECT CONCAT('a', SPACE(-1), 'b');
 concat
--------
 ab
(1 row)
```

STRCMP

STRCMP(str1, str2)

Description: Checks whether **str1** is the same as **str2**.

Return type: INT

- If **str1** is equal to **str2**, the STRCMP function returns **0**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- If **str1** is smaller than **str2**, the STRCMP function returns **-1**.
- If **str1** is greater than **str2**, the STRCMP function returns **1**.
- If any parameter is **NULL**, the STRCMP function returns **NULL**.

Examples:

```
m_db=# SELECT STRCMP('abc', 'abc');
 strcmp
--------
      0
(1 row)

m_db=# SELECT STRCMP('abc1', 'abc');
 strcmp
--------
      1
(1 row)

m_db=# SELECT STRCMP('abc', 'abc1');
 strcmp
--------
     -1
(1 row)

m_db=# SELECT STRCMP('abc1', 'abc2');
 strcmp
--------
     -1
(1 row)

m_db=# SELECT STRCMP('abc1', NULL);
 strcmp
--------

(1 row)
```

SUBSTR

The SUBSTR function has the following four prototypes. The meanings of **str**, **pos**, and **len** in each prototype are the same.

- SUBSTR(str, pos)
- SUBSTR(str FROM pos)
- SUBSTR(str, pos, len)
- SUBSTR(str FROM pos FOR len)
- SUBSTR(str FOR len FROM pos)
- SUBSTR(str FOR len)

Description: Returns the sub-string of **str**. The start position is **pos** and the length is **len**. If **len** is not specified, the returned sub-string starts from **pos** and ends at the end of **str**.

- The position of **pos** starts from **1**.
- If the value of a parameter contains **NULL**, **NULL** is returned.
- If **pos** is a negative number, the start position is determined from the tail of **str** to the head in reverse order.
- If the value of **len** is less than or equal to **0**, the position of **pos** is **0**, or the position of **pos** is out of range, an empty string is returned.

Parameters: **Table 4-57** describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-57 SUBSTR parameters

Parameter	Type	Description
str	TEXT	String to be truncated.
pos	INT	Start position.
len	INT	Length of a sub-string.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT SUBSTR('abcdefg', 2, 3);
 substr
--------
 bcd
(1 row)

m_db=# SELECT SUBSTR('abcdefg', NULL);
 substr
--------

(1 row)

m_db=# SELECT SUBSTR('abcdefg', 2);
 substr
--------
 bcdefg
(1 row)

m_db=# SELECT SUBSTR('abcdefg', -2, 3);
 substr
--------
 fg
(1 row)

m_db=# SELECT SUBSTR('abcdefg', -2);
 substr
--------
 fg
(1 row)

m_db=# SELECT SUBSTR('abcdefg', 0, 3);
 substr
--------

(1 row)
```

SUBSTRING

The SUBSTRING function has the following four prototypes. The meanings of **str**, **pos**, and **len** in each prototype are the same.

- SUBSTRING(str, pos)
- SUBSTRING(str FROM pos)
- SUBSTRING(str, pos, len)
- SUBSTRING(str FROM pos FOR len)
- SUBSTRING(str FOR len FROM pos)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- SUBSTRING(str FOR len)

Description: The function and usage of SUBSTRING are the same as those of **SUBSTR**.

Parameters: **Table 4-58** describes the parameters.

Table 4-58 SUBSTRING parameters

Parameter	Type	Description
str	TEXT	String to be truncated.
pos	INT	Start position.
len	INT	Length of a sub-string.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT SUBSTRING('abcdefg', 2, 3);
 substring
-----------
 bcd
(1 row)

m_db=# SELECT SUBSTRING('abcdefg', NULL);
 substring
-----------

(1 row)

m_db=# SELECT SUBSTRING('abcdefg', 2);
 substring
-----------
 bcdefg
(1 row)

m_db=# SELECT SUBSTRING('abcdefg', -2, 3);
 substring
-----------
 fg
(1 row)

m_db=# SELECT SUBSTRING('abcdefg', -2);
 substring
-----------
 fg
(1 row)

m_db=# SELECT SUBSTRING('abcdefg', 0, 3);
 substring
-----------

(1 row)
```

SUBSTRING_INDEX

SUBSTRING_INDEX(str, delim, count)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Uses **delim** and **count** to determine the position of the final delimiter in the string str and returns all characters on the left (or right) of the final delimiter. When **delim** is matched, case sensitivity is required.

- If **count** is a positive number, the system counts from the left of the string until **delim** appears for **count** times. All characters on the left of the final delimiter are returned.
- If **count** is a negative number, the system counts from the right of the string until **delim** appears for **|count|** times. All characters on the right of the final delimiter are returned.
- If any parameter is **NULL**, **NULL** is returned.
- If **str** or **delim** is an empty string, an empty string is returned.
- If **count** is **0**, an empty string is returned.

Parameters: **Table 4-59** describes the parameters.

Table 4-59 SUBSTRING_INDEX parameters

Parameter	Type	Description
str	TEXT	String to be truncated.
delim	TEXT	Delimiter.
count	INT	Number of occurrences of delimiters.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT SUBSTRING_INDEX('abc..def..ghi..jkl....', '..', 3);
 substring_index
-----------------
 abc..def..ghi
(1 row)

m_db=# SELECT SUBSTRING_INDEX('abc..def..ghi..jkl....', '..', -3);
 substring_index
-----------------
 jkl....
(1 row)

m_db=# SELECT SUBSTRING_INDEX('abc..def..ghi..jkl....', '..', NULL);
 substring_index
-----------------

(1 row)

m_db=# SELECT SUBSTRING_INDEX('abc..def..ghi..jkl....', '..' ,0);
 substring_index
-----------------

(1 row)
```

TRIM

TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Deletes all prefix/suffix **remstr** from **str**. If any parameter is **NULL**, **NULL** is returned. Matching **remstr** is case-sensitive.

- remstr: The default value is a space.
- [BOTH | LEADING | TRAILING]:
  - BOTH: Deletes the prefix and suffix.
  - LEADING: Deletes the prefix.
  - TRAILING: Deletes the suffix.
  - The default value is **BOTH**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT TRIM('   abc   ');
 trim
------
 abc
(1 row)

m_db=# SELECT TRIM(TRAILING FROM '   abc   ');
  trim
--------
    abc
(1 row)

m_db=# SELECT TRIM(BOTH 'xyz' FROM 'xyzxyabcxyzxy');
    trim
------------
 xyabcxyzxy
(1 row)
```

UCASE

UCASE(str)

Description: Converts lowercase letters in a string to uppercase letters. The function and usage of UCASE are the same as those of **UPPER**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT UCASE('Hello,world!');
    ucase
--------------
 HELLO,WORLD!
(1 row)
```

UNHEX

UNHEX(str)

Description: Interprets each character in **str** as a hexadecimal number, converts it to the byte represented by the number, and returns a binary string. The reverse operation is **HEX(str)**. The character in **str** must be a valid hexadecimal number. If the parameter contains any invalid number, **NULL** is returned.

Return value type: LONGBLOB

Examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT UNHEX(616263), HEX(UNHEX('616263'));
 unhex |  hex
-------+--------
 abc   | 616263
(1 row)
```

UPPER

UPPER(str)

Description: Converts lowercase letters in a string to uppercase letters. The function and usage of UPPER are the same as those of **UCASE**.

Return value type: TEXT or LONGBLOB

Examples:

```
m_db=# SELECT UPPER('Hello,world!');
    upper
--------------
 HELLO,WORLD!
(1 row)
```

4.5.8 Datetime Functions

ADDDATE

- ADDDATE(date, INTERVAL expr unit)

  Description: Adds an interval on a specified date and returns the calculation result. The function and usage of this form of function is the same as those of **DATE_ADD**.

  Parameters: **Table 4-60** describes the parameters.

  Table 4-60 ADDDATE parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
expr	Integer, floating-point number, string, or expression.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.
unit	Keyword.	Specifies the unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND. For details, see **Time Interval Expressions**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT ADDDATE('2018-05-01', INTERVAL 1 DAY);
  adddate
------------
 2018-05-02
(1 row)
```

- ADDDATE(date, days)

  Description: Adds the number of days specified by **days** to the date specified by **date** and returns the calculation result.

  Parameters: **Table 4-61** describes the parameters.

  Table 4-61 ADDDATE parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
days	Integer, floating-point number, or string.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.

  Return value type: TEXT, DATE, DATETIME, or TIME.

  Examples:

```
m_db=# SELECT ADDDATE('2018-05-01', 1);
  adddate
------------
 2018-05-02
(1 row)
```

ADDTIME

ADDTIME(expr1, expr2)

Description: Returns the sum of **expr1** and **expr2**. The format of the return value is the same as that of **expr1**.

Parameters:

- **expr1**: specifies the time or datetime expression.
- **expr2**: specifies the time expression.

Return value type: TEXT, DATETIME, or TIME. The return value type is related to the **expr1** type. If **expr1** is parsed as **DATETIME**, **DATETIME** is returned. If **expr1** is parsed as **TIME**, **TIME** is returned.

Examples:

```
m_db=# SELECT ADDTIME('2000-03-01 20:59:59', '00:00:01');
      addtime
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
---------------------
 2000-03-01 21:00:00
(1 row)

m_db=# SELECT ADDTIME('20:59:59', '00:00:01');
 addtime
----------
 21:00:00
(1 row)
```

CONVERT_TZ

CONVERT_TZ(dt, from_tz, to_tz)

Description: The CONVERT_TZ function is used to convert the datetime value **dt** from **from_tz** time zone to **to_tz** time zone. If the **dt** parameter is invalid, null is returned. If **dt** is out of the range supported by the TIMESTAMP type, no conversion occurs.

Parameters: **Table 4-62** describes the parameters.

Table 4-62 CONVERT_TZ parameters

Parameter	Type	Description	Value Range
dt	TEXT, DATETIME, DATE, TIME, and other types that can be implicitly converted to DATETIME.	Specifies the datetime.	For details, see **Table 4-76**.
from_tz/ to_tz	A string in the format of ±hh:mm.	Offset compared with the UTC time, for example, '+08:00'.	[-12:59, +13:00]. The plus and minus signs are required.
	Named time zone.	For example, 'MET' and 'UTC'.	For details, see the PG_TIMEZONE_NAMES system view.

Return value type: DATETIME

Examples:

```
m_db=# SELECT CONVERT_TZ('2023-01-01 10:10:10', '+00:00', '+08:00');
    convert_tz
---------------------
 2023-01-01 11:10:10
(1 row)
```

CURDATE

CURDATE()

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Returns the system date when the local function calling starts. The time zone can be changed within the same connection. The returned date is affected by the time zone.

Return value type: DATE

Examples:

```
m_db=# SELECT CURDATE();
  curdate
------------
 2023-12-09
(1 row)
```

CURRENT_DATE

CURRENT_DATE/CURRENT_DATE()

Description: Returns the system date when the local function calling starts. The time zone can be changed within the same connection. The returned date is affected by the time zone. This function is an alias of **CURDATE**.

Return value type: DATE

Examples:

```
m_db=# SELECT CURRENT_DATE;
 current_date
--------------
 2023-12-09
(1 row)

m_db=# SELECT CURRENT_DATE();
 current_date
--------------
 2023-12-09
(1 row)
```

CURRENT_TIME

CURRENT_TIME/CURRENT_TIME([scale])

Description: Returns the start time of the local function calling, excluding the date.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: TIME

Examples:

```
m_db=# SELECT CURRENT_TIME;
 current_time
--------------
 10:14:12
(1 row)

m_db=# SELECT CURRENT_TIME(6);
  current_time
-----------------
 10:14:15.512064
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP/CURRENT_TIMESTAMP([scale])

Description: Returns the datetime when the local function calling starts. This function is an alias of **NOW**.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT CURRENT_TIMESTAMP;
  current_timestamp
---------------------
 2023-12-09 11:20:04
(1 row)

m_db=# SELECT CURRENT_TIMESTAMP(6);
     current_timestamp
----------------------------
 2023-12-09 11:20:07.059582
(1 row)
```

CURTIME

CURTIME([scale])

Description: Returns the start time of the local function calling, excluding the date. This function is an alias of **CURRENT_TIME**.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: TIME

Examples:

```
m_db=# SELECT CURTIME();
 curtime
----------
 11:24:03
(1 row)

m_db=# SELECT CURTIME(6);
     curtime
-----------------
 11:24:05.590022
(1 row)
```

DATE()

DATE(expr)

Description: Returns the date part of **expr**.

For the parameter **expr**, the value can be TEXT, DATETIME, DATE, TIME, or a time or date expression.

Return value type: DATE

Examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT DATE('2023-01-01 10:11:12');
    date
------------
 2023-01-01
(1 row)
```

DATE_ADD

DATE_ADD(date, INTERVAL expr unit)

Description: Adds an interval on a specified date and returns the calculation result.

Parameters: **Table 4-63** describes the parameters.

Table 4-63 DATE_ADD parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
expr	Integer, floating-point number, string, or expression.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.
unit	Keyword.	Specifies the unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND. For details, see **Time Interval Expressions**.

Return value type: TEXT, DATE, DATETIME, or TIME.

Examples:

```
m_db=# SELECT DATE_ADD('2018-05-01', INTERVAL 1 DAY);
  adddate
------------
 2018-05-02
(1 row)
```

DATE_FORMAT

DATE_FORMAT(date, format)

Description: Outputs the datetime in the specified format.

Parameters: **Table 4-64** describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-64 DATE_FORMAT parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Date to be formatted.	See the value ranges of the corresponding types.
format	TEXT.	Formatted string.	For details, see **Table 4-65**.

Table 4-65 Values and meanings of formatted strings

Format Character	Description
%a	Abbreviation of a week (Sun, ..., Sat).
%b	Abbreviation of a month (Jan, ..., Dec).
%c	Month number (0, ..., 12).
%D	Every day in a month with an English prefix (0th, 1st, 2nd, 3rd, ...).
%d	Two-digit representation of every day in a month (00, ..., 31).
%e	Sequence number of every day in a month (0, ..., 31).
%f	Microsecond (000000, ..., 999999).
%H	Hour (00, ..., 23).
%h	Hour (01, ..., 12).
%I	Hour (01, ..., 12).
%i	Minute (00, ..., 59).
%j	Every day in a year (001, ..., 366).
%k	Hour (0, ..., 23).
%l	Hour (1, ..., 12).
%M	Month name (January, ..., December).
%m	Two-digit month (00, ..., 12).
%p	AM or PM.
%r	12-hour time (hh:mm:ss followed by AM or PM).
%S	Second (00, ..., 59).
%s	Second (00, ..., 59).
%T	24-hour time (hh:mm:ss).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Format Character	Description
%U	Week of a year (00, ..., 53). Each week starts from Sunday.
%u	Week of a year (00, ..., 53). Each week starts from Monday.
%V	Week of a year (01, ..., 53). Each week starts from Sunday.
%v	Week of a year (01, ..., 53). Each week starts from Monday.
%W	Name of a week (Sunday, ..., Saturday).
%w	Every day in a week (0 = Sunday, ..., 6 = Saturday).
%X	Week of a year. Each week starts from Sunday. The value is a four-digit number and is used for %V.
%x	Week of a year. Each week starts from Monday. The value is a four-digit number and is used for %v.
%Y	Four-digit year.
%y	Two-digit year.
%%	Alphanumeric character %.
%*x*	Any *x* not listed above.

Return value type: TEXT

Examples:

```
m_db=# SELECT DATE_FORMAT('2023-10-11 12:13:14.151617','%b %c %M %m');
    date_format
--------------------
 Oct 10 October 10
(1 row)
```

DATE_SUB

DATE_SUB(date, INTERVAL expr unit)

Description: Subtracts a period of time from a specified date and returns the calculation result.

Parameters: **Table 4-66** describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-66 DATE_SUB parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
expr	Integer, floating-point number, string, or expression.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.
unit	Keyword.	Specifies the unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND. For details, see **Time Interval Expressions**.

Return value type: TEXT, DATE, DATETIME, or TIME.

Examples:

```
m_db=# SELECT DATE_SUB('2018-05-01', INTERVAL 1 DAY);
  date_sub
------------
 2018-04-30
(1 row)
```

DATEDIFF

DATEDIFF(date1, date2)

Description: Returns the number of days between **date1** and **date2**. Only the date part of the parameter is used in the calculation. The time part is ignored.

Parameters:

- **date1**: time expression, TEXT, DATETIME, DATE, or TIME.
- **date2**: time expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT DATEDIFF('2018-05-01', '2018-04-01');
 datediff
----------
       30
(1 row)
```

DAY

DAY(date)

Description: Extracts the number of days in a datetime expression and returns the result. This function is an alias of **DAYOFMONTH**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The parameter **date** specifies the datetime to be extracted. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT DAY('2018-05-12');
 day
-----
  12
(1 row)
```

DAYNAME

DAYNAME(date)

Description: Returns the name of a day in a week. The language used by the return value is specified by the GUC parameter **lc_time_names**.

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return value type: TEXT

Examples:

```
m_db=# SELECT DAYNAME('2018-05-12');
 dayname
-----------
 Saturday
(1 row)
```

DAYOFMONTH

DAYOFMONTH(date)

Description: Extracts the number of days in a datetime expression and returns the result. This function is an alias of **DAY**.

The parameter **date** specifies the datetime to be extracted. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT DAYOFMONTH('2018-05-12');
 dayofmonth
------------
         12
(1 row)
```

DAYOFWEEK

DAYOFWEEK(date)

Description: Returns the working day index of a date (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return type: BIGINT

Examples:

```
m_db=# SELECT DAYOFWEEK('2023-04-16');
 dayofweek
-----------
         1
(1 row)
```

DAYOFYEAR

DAYOFYEAR(date)

Description: Returns the number of a day in the year.

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: integer; value range: [1,366].

Examples:

```
m_db=# SELECT DAYOFYEAR('2000-12-31');
 dayofyear
-----------
       366
(1 row)
```

EXTRACT

EXTRACT(unit FROM date)

Description: Returns the **unit** value of **date** as an integer. If **unit** specifies multiple parts, all values are concatenated in sequence.

Parameters:

- **unit**: specifies the unit of the time interval. For details, see **Time Interval Expressions**.
- **date**: specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: BIGINT

Examples:

```
m_db=# SELECT EXTRACT(DAY FROM '2000-12-31');
 extract
---------
      31
(1 row)

m_db=# SELECT EXTRACT(WEEK FROM '2000-12-31');
 extract
---------
      53
(1 row)
m_db=# SELECT EXTRACT(YEAR_MONTH FROM '2000-12-31');
 extract
---------
  200012
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

FROM_DAYS

FROM_DAYS(days)

Description: Returns the date corresponding to **days**. The number of days refers to the number of days since 0000-01-01.

The parameter **days** specifies the specified number of days.

Return value type: DATE

Examples:

```
m_db=# SELECT FROM_DAYS(50000);
 from_days
------------
 0136-11-23
(1 row)
```

FROM_UNIXTIME

FROM_UNIXTIME(unix_timestamp[, format])

Description: Converts a Unix timestamp to the datetime format. A Unix timestamp is the number of seconds from 08:00:00 UTC on January 1, 1970 to a specified time.

Parameters:

- unix_timestamp: Unix timestamp.
- format: Returns a datetime string in the specified format.

  For details about the return format, see the **DATE_FORMAT** function.

Return value type: TEXT or DATETIME

Examples:

```
m_db=# SELECT FROM_UNIXTIME(1111885200);
    from_unixtime
---------------------
 2005-03-27 09:00:00
(1 row)

m_db=# SELECT FROM_UNIXTIME(1447430881, '%Y %D %M %h:%i:%s %x');
          from_unixtime
-----------------------------------
 2015 14th November 12:08:01 2015
(1 row)
```

GET_FORMAT

GET_FORMAT({DATE | TIME | DATETIME | TIMESTAMP}, {'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL'})

Description: Returns a string in a specified format, that is, the year, month, day, hour, minute, and second formats and sorting standards of different regions.

Parameters:

- DATE|TIME|DATETIME|TIMESTAMP: time types, which are keywords.
- 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL': five time formats, which are of the TEXT type.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return value type: TEXT

Examples:

```
m_db=# SELECT GET_FORMAT(DATE, 'EUR');
 get_format
------------
 %d.%m.%Y
(1 row)

m_db=# SELECT GET_FORMAT(DATE, 'USA');
 get_format
------------
 %m.%d.%Y
(1 row)
```

HOUR

HOUR(expr)

Description: Returns the hour part of **expr**.

The parameter **time** specifies the time. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT HOUR('10:11:12');
 hour
------
   10
(1 row)
```

LAST_DAY

LAST_DAY(date)

Description: Returns the date of the last day of the month specified by **date**.

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return value type: DATE

Examples:

```
m_db=# SELECT LAST_DAY('2023-01-01');
  last_day
------------
 2023-01-31
(1 row)

m_db=# SELECT LAST_DAY('2023-01-01 10:11:12');
  last_day
------------
 2023-01-31
(1 row)
```

LOCALTIME

LOCALTIME/LOCALTIME([scale])

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Returns the datetime when the local function calling starts. LOCALTIME and LOCALTIME() are synonyms of **NOW**.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT LOCALTIME;
     localtime
---------------------
 2023-12-09 17:58:18
(1 row)

m_db=# SELECT LOCALTIME(6);
        localtime
----------------------------
 2023-12-09 17:58:21.283227
(1 row)
```

LOCALTIMESTAMP

LOCALTIMESTAMP/LOCALTIMESTAMP([scale])

Description: Returns the current datetime. LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms of **NOW**.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT LOCALTIMESTAMP;
  localtimestamp
---------------------
 2023-12-09 18:00:34
(1 row)

m_db=# SELECT LOCALTIMESTAMP(6);
      localtimestamp
----------------------------
 2023-12-09 18:01:03.053427
(1 row)
```

MAKEDATE

MAKEDATE(year, dayofyear)

Description: Returns a date based on the given year and day.

Parameters:

- **year**: BIGINT, indicating the specified year.
- **dayofyear**: BIGINT, indicating the number of days in the year. The value can cross years. If the value is less than or equal to **0**, **NULL** is returned.

Return value type: DATE

Examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT MAKEDATE(2000, 60);
  makedate
------------
 2000-02-29
(1 row)
```

MAKETIME

MAKETIME(hour, minute, second)

Description: Generates a specified time value based on the input parameters **hour**, **minute**, and **second**.

Parameters:

- hour: value corresponding to the hour part.
- minute: value corresponding to the minute part.
- second: value corresponding to the second part, which can contain the decimal part.

Return value type: TIME

Examples:

```
m_db=# SELECT MAKETIME(8, 15, 26.53);
  maketime
-------------
 08:15:26.53
(1 row)

m_db=# SELECT MAKETIME(-838, 15, 26.53);
   maketime
---------------
 -838:15:26.53
(1 row)
```

MICROSECOND

MICROSECOND(expr)

Description: Returns the number of microseconds in **expr**.

For the parameter **expr**, the value can be TEXT, DATETIME, DATE, TIME, or a time or date expression.

Return type: BIGINT

Examples:

```
m_db=# SELECT MICROSECOND('2023-5-5 10:10:10.24485');
 microsecond
-------------
      244850
(1 row)
```

MINUTE

MINUTE(expr)

Description: Returns the number of minutes in **expr**.

For the parameter **expr**, the value can be TEXT, DATETIME, DATE, TIME, or a time or date expression.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return type: INT

Examples:

```
m_db=# SELECT MINUTE('10:11:12');
 minute
--------
     11
(1 row)
```

MONTH

MONTH(date)

Description: Extracts the month part of the datetime and returns the result.

The parameter **date** specifies the datetime to be extracted. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT MONTH('2018-05-12');
 month
-------
     5
(1 row)
```

MONTHNAME

MONTHNAME(date)

Description: Returns the full month name of a date.

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return value type: TEXT

Examples:

```
m_db=# SELECT MONTHNAME('2018-05-12');
 monthname
-----------
 May
(1 row)
```

NOW

NOW([scale])

Description: Returns the datetime when the local function calling starts. The datetime are affected by the time zone setting.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT NOW();
        now
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
---------------------
 2023-12-11 09:46:56
(1 row)

m_db=# SELECT NOW(6);
            now
----------------------------
 2023-12-11 09:46:58.404811
(1 row)
```

PERIOD_ADD

PERIOD_ADD(period, month_number)

Description: Adds a specified number of months to a date in the specified format and returns the date in YYYYMM format.

Parameters:

- period: date in the format of YYYYMM or YYMM.
- month_number: number of months to be added. The value can be a negative number.

Return type: BIGINT.

Examples:

```
m_db=# SELECT PERIOD_ADD(202205, -12);
 period_add
------------
     202105
(1 row)
```

PERIOD_DIFF

PERIOD_DIFF(P1, P2)

Description: Calculates the month difference between two dates.

Parameters: **P1** and **P2** are dates in the YYMM or YYYYMM format.

Return value type: BIGINT (month difference)

Examples:

```
m_db=# SELECT PERIOD_DIFF('202101', '202102');
 period_diff
-------------
          -1
(1 row)
```

QUARTER

QUARTER(date)

Description: Returns the quarter value of a specified date. The value ranges from 1 to 4

The parameter **date** specifies the datetime. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT QUARTER('2018-05-12');
 quarter
---------
       2
(1 row)
```

SEC_TO_TIME

SEC_TO_TIME(seconds)

Description: Converts the number of seconds into a time of the TIME type.

The parameter **seconds** indicates the specified number of seconds. The value range is [-3020399, +3020399].

Return value type: TIME

Examples:

```
m_db=# SELECT SEC_TO_TIME(3020399);
 sec_to_time
-------------
 838:59:59
(1 row)
```

SECOND

SECOND(expr)

Description: Returns the number of seconds in **expr**.

The parameter **expr** specifies the time. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT SECOND('10:11:12');
 second
--------
     12
(1 row)
```

STR_TO_DATE

STR_TO_DATE(str, format)

Description: Converts a specified string to a date or time based on the specified date format.

Parameters:

- str: string to be formatted into a date.
- format: formatted string. For details, see **Table 4-65**.

Return value type: DATETIME, DATE, or TIME

Examples:

```
m_db=# SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
 str_to_date
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-------------
 2013-05-01
(1 row)
```

SUBDATE

- SUBDATE(date, INTERVAL expr unit)

  Description: Subtracts a period of time from a specified date and returns the calculation result. The function and usage of this form of function is the same as those of **DATE_SUB**.

  Parameters: **Table 4-67** describes the parameters.

  Table 4-67 SUBDATE parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
expr	Integer, floating-point number, string, or expression.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.
unit	Keyword.	Specifies the unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND. For details, see **Time Interval Expressions**.

  Return value type: TEXT, DATE, DATETIME, or TIME.

  Examples:

```
m_db=# SELECT SUBDATE('2018-05-01', INTERVAL 1 DAY);
  subdate
------------
 2018-04-30
(1 row)
```

- SUBDATE(date, days)

  Description: Subtracts the number of days specified by **days** from the date specified by **date** and returns the calculation result.

  Parameters: **Table 4-68** describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-68 SUBDATE parameters

Parameter	Type	Description	Value Range
date	Time expression, TEXT, DATETIME, DATE, or TIME.	Specifies the datetime reference.	See the value ranges of the corresponding types.
days	Integer, floating-point number, or string.	Specifies the time interval, which can be a negative value.	See the value ranges of the corresponding types.

Return value type: TEXT, DATE, DATETIME, or TIME.

Examples:

```
m_db=# SELECT SUBDATE('2018-05-01', 1);
  subdate
------------
 2018-04-30
(1 row)
```

SUBTIME

SUBTIME(expr1, expr2)

Description: Returns the value obtained by subtracting **expr2** from **expr1**. The format of the return value is the same as that of **expr1**.

Parameters:

- expr1: specifies the time or datetime expression.
- expr2: specifies the time expression.

Return value type: TEXT, DATETIME, or TIME. The return value type is related to the **expr1** type. If **expr1** is parsed as **DATETIME**, **DATETIME** is returned. If **expr1** is parsed as **TIME**, **TIME** is returned.

Examples:

```
m_db=# SELECT SUBTIME('2000-03-01 20:59:59', '00:00:01');
       subtime
---------------------
 2000-03-01 20:59:58
(1 row)

m_db=# SELECT SUBTIME('20:59:59', '00:00:01');
 subtime
----------
 20:59:58
(1 row)
```

SYSDATE

SYSDATE([precision])

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Returns the system date and time when a function is executed.

The parameter **precision** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT SYSDATE();
       sysdate
---------------------
 2023-12-11 11:28:29
(1 row)

m_db=# SELECT SYSDATE(6);
          sysdate
----------------------------
 2023-12-11 11:28:31.535374
(1 row)
```

TIME

TIME(expr)

Description: Returns the time part of **expr**.

For the parameter **expr**, the value can be TEXT, DATETIME, DATE, TIME, or a time or date expression.

Return value type: TIME

Examples:

```
m_db=# SELECT TIME('2023-01-01 10:11:12');
   time
----------
 10:11:12
(1 row)
```

TIME_FORMAT

TIME_FORMAT(time, format)

Description: Formats the **time** input parameter based on the format specified by **format**.

Parameters:

- time: time expression, TEXT, DATETIME, DATE, or TIME.
- format: formatted string. **Table 4-69** describes the values and meanings.

Table 4-69 Formatted string

Format Character	Description
%f	Microsecond (000000–999999)
%H	Hour (00 to 23)
%h, %I	Hour (00 to 12)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Format Character	Description
%l	Hour (0 to 12)
%k	Hour (0 to 838)
%i	Minute (00 to 59)
%p	AM or PM
%r	Time in 12-hour AM or PM format (hh:mm:ss AM/PM)
%S, %s	Second (00 to 59)
%T	Time in 24-hour format (hh:mm:ss)
%a, %b, %D, %j, %M, %U, %u, %V, %v, %W, %w, %X, %x	Null
%c, %e	0
%d, %m, %y	00
%Y	0000
%*Other characters*|*Other characters*, for example, %A/A	The character itself is returned, for example, A.
%*Single character* + string s	%*Single character* is parsed and then concatenated with s.

Return value type: TEXT

Examples:

```
m_db=# SELECT TIME_FORMAT('25:30:30', '%T|%r|%H|%h|%I|%i|%S|%f|%p|%k');
                time_format
---------------------------------------------------
 25:30:30|01:30:30 AM|25|01|01|30|30|000000|AM|25
(1 row)
```

TIME_TO_SEC

TIME_TO_SEC(time)

Description: Converts the input parameter of the TIME type to the number of seconds.

The parameter **time** is a time type expression, TEXT, DATETIME, DATE, or TIME. The time expression is parsed as TIME.

Return type: INT

Examples:

```
m_db=# SELECT TIME_TO_SEC('10:11:12');
 time_to_sec
-------------
       36672
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

TIMEDIFF

TIMEDIFF(date1, date2)

Description: Returns the interval between two dates in TIME type.

The parameter **date1** or **date2** is a time type expression, TEXT, DATETIME, DATE, or TIME.

Return value type: TIME

Examples:

```
m_db=# SELECT TIMEDIFF(date'2022-12-30',20221229);
 timediff
----------
 24:00:00
(1 row)
```

TIMESTAMP

- TIMESTAMP(expr)

  Description: Returns the datetime value of **expr**.

  Parameter: time type expression, TEXT, DATETIME, DATE, or TIME.

  Return value type: DATETIME

  Examples:

  ```
  m_db=# SELECT TIMESTAMP('2022-12-30');
      timestamp
  ---------------------
   2022-12-30 00:00:00
  (1 row)
  ```

- TIMESTAMP(expr1, expr2)

  Description: Returns the sum of **expr1** and **expr2**.

  Parameter: time type expression, TEXT, DATETIME, DATE, or TIME.

  Return value type: DATETIME

  Examples:

  ```
  m_db=# SELECT TIMESTAMP('2022-12-30', '12:00:00');
      timestamp
  ---------------------
   2022-12-30 12:00:00
  (1 row)
  ```

TIMESTAMPADD

TIMESTAMPADD(unit, interval, datetime_expr)

Description: Returns a new datetime, which is calculated by adding multiple intervals of a unit to **datetime_expr**.

Parameters: **Table 4-70** describes the parameters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-70 TIMESTAMPADD parameters

Parameter	Type	Description	Value Range
unit	Keyword.	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND.
interval	Numeric expression.	Interval.	See the value ranges of the corresponding types.
datetime_expr	Time expression, TEXT, DATETIME, DATE, or TIME.	Reference value of the time and date.	See the value ranges of the corresponding types.

Return value type: TEXT, DATE, DATETIME, or TIME.

Examples:

```
m_db=#  SELECT TIMESTAMPADD(DAY,-2,'2022-07-27');
 timestampadd
--------------
 2022-07-25
(1 row)
```

TIMESTAMPDIFF

TIMESTAMPDIFF(unit, date1, date2)

Description: Returns the interval between two dates in the form of **unit**.

Parameters: **Table 4-71** describes the parameters.

Table 4-71 TIMESTAMPDIFF parameters

Parameter	Type	Description	Value Range
unit	Keyword.	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, or MICROSECOND.
date1	Time expression, TEXT, DATETIME, DATE, or TIME.	Datetime.	See the value ranges of the corresponding types.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Type	Description	Value Range
date2	Time expression, TEXT, DATETIME, DATE, or TIME.	Datetime.	See the value ranges of the corresponding types.

Return type: BIGINT

Examples:

```
m_db=# SELECT TIMESTAMPDIFF(SECOND, '2023-01-01 10:11:12', '2023-01-01 05:00:00');
 timestampdiff
---------------
        -18672
(1 row)
```

TO_DAYS

TO_DAYS(date)

Description: Returns the number of days between a specified date and 0000-01-01.

Parameter: The input parameter is a time expression, TEXT, DATETIME, DATE, or TIME. The time expression is parsed as **DATE**.

Return type: INT

Examples:

```
m_db=# SELECT TO_DAYS('2000-1-1');
 to_days
---------
  730485
(1 row)
```

TO_SECONDS

TO_SECONDS(expr)

Description: Returns the number of seconds since year 0 to **expr**.

Parameter: The input parameter is a time or datetime expression, TEXT, DATETIME, DATE, or TIME type. The time expression is parsed as **DATETIME**.

Return type: BIGINT

Examples:

```
m_db=# SELECT TO_SECONDS('2009-11-29 13:43:32');
 to_seconds
-------------
 63426721412
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

UNIX_TIMESTAMP

UNIX_TIMESTAMP([date])

Description:

- If **date** is not specified, the number of seconds between the current time and 1970-01-01 00:00:00 is returned, which is affected by the setting of the time zone parameter.
- If **date** is specified, the number of seconds between the specified time and 1970-01-01 00:00:00 is returned, which is affected by the setting of the time zone parameter.

Parameter: time type expression, TEXT, DATETIME, DATE, or TIME.

Return value type: DECIMAL or BIGINT. If the input parameter contains a decimal part or the input parameter is an invalid string of the time type, **DECIMAL** is returned. Otherwise, **BIGINT** is returned.

Examples:

```
m_db=# SELECT UNIX_TIMESTAMP('2022-12-22');
 unix_timestamp
----------------
     1671638400
(1 row)
```

UTC_DATE

UTC_DATE/UTC_DATE()

Description: Returns the current UTC date of function execution as a value in YYYY-MM-DD format.

Return value type: DATE

Examples:

```
m_db=# SELECT UTC_DATE;
  utc_date
------------
 2023-12-11
(1 row)

m_db=# SELECT UTC_DATE();
  utc_date
------------
 2023-12-11
(1 row)
```

UTC_TIME

UTC_TIME/UTC_TIME([scale])

Description: Returns the current UTC time of function execution as a value in HH:MM:SS format.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: TIME

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT UTC_TIME;
 utc_time
----------
 07:10:49
(1 row)

m_db=# SELECT UTC_TIME();
 utc_time
----------
 07:10:49
(1 row)

m_db=# SELECT UTC_TIME(6);
    utc_time
-----------------
 07:10:51.445213
(1 row)
```

UTC_TIMESTAMP

UTC_TIMESTAMP/UTC_TIMESTAMP([scale])

Description: Returns the current UTC timestamp of function execution as a value in YYYY-MM-DD HH:MM:SS format.

The parameter **scale** specifies the precision of the microsecond part. The value is an integer ranging from 0 to 6. The default value is **0**.

Return value type: DATETIME

Examples:

```
m_db=# SELECT UTC_TIMESTAMP;
    utc_timestamp
---------------------
 2023-12-11 07:12:02
(1 row)

m_db=# SELECT UTC_TIMESTAMP();
    utc_timestamp
---------------------
 2023-12-11 07:12:04
(1 row)

m_db=# SELECT UTC_TIMESTAMP(3);
      utc_timestamp
-------------------------
 2023-12-11 07:12:06.413
(1 row)
```

WEEK

WEEK(date[, mode])

Description: Returns the number of weeks of a date. The number of weeks is calculated according to ISO 8601:1988. In the ISO definition, the first few days of January may be the 52nd or 53rd week of the previous year, and the last few days of December may be the first week of the following year.

Parameters:

- **date**: specifies the datetime, which is a time type expression, TEXT, DATETIME, DATE, or TIME.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- When mode parameters are omitted, the value of the system variable **default_week_format** is used. The default value is **0**. When mode parameters are specified, you can specify the start of a week and the range of return values. **Table 4-72** describes the meanings of optional mode parameters. The default value is **0**.

  Table 4-72 Values and meanings of mode parameters

mode	First Day of a Week	Return Value Range	First Week in January
0	Sunday	0~53	The first week with Sunday.
1	Monday	0~53	The first week with four days or more.
2	Sunday	1~53	The first week with Sunday.
3	Monday	1~53	The first week with four days or more.
4	Sunday	0~53	The first week with four days or more.
5	Monday	0~53	The first week with Monday.
6	Sunday	1~53	The first week with four days or more.
7	Monday	1~53	The first week with Monday.

Return type: BIGINT

Examples:

```
m_db=# SELECT WEEK('2000-01-01');
 week
------
    0
(1 row)

m_db=# SELECT WEEK('2000-01-01', 2);
 week
------
   52
(1 row)
```

WEEKDAY

WEEKDAY(date)

Description: Returns the working day index of a date, that is, Monday is 0, Tuesday is 1, Wednesday is 2, Thursday is 3, Friday is 4, Saturday is 5, and Sunday is 6.

Parameter: time type expression, TEXT, DATETIME, DATE, or TIME.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Return type: INT

Examples:

```
m_db=# SELECT WEEKDAY('1970-01-01 12:00:00');
 weekday
---------
 3
(1 row)
```

WEEKOFYEAR

WEEKOFYEAR(date)

Description: Returns the calendar week of the datetime. The value range is [1, 53]. Its functions are the same as those of **WEEK(date, 3)**.

Parameter: time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT

Examples:

```
m_db=# SELECT WEEKOFYEAR('1970-05-22');
 weekofyear
------------
         21
(1 row)
```

YEAR

YEAR(date)

Description: Extracts the year part of the datetime and returns the result.

The parameter **date** specifies the datetime to be extracted. The value can be a time type expression, TEXT, DATETIME, DATE, or TIME.

Return type: INT.

Examples:

```
m_db=# SELECT YEAR('2018-05-12');
 year
------
 2018
(1 row)
```

YEARWEEK

YEARWEEK(date[, mode])

Description: Returns the year and week of a date.

Parameters:

- **date**: specifies the datetime, which is a time type expression, TEXT, DATETIME, DATE, or TIME.
- For details about the optional mode parameters, see **Table 4-72**.

Return type: BIGINT

Examples:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=#  SELECT YEARWEEK('2000-01-01', 3);
 yearweek
----------
   199952
(1 row)
```

4.5.9 Type Conversion Functions

CAST

CAST(expr AS type)

Description: Explicitly converts **expr** to **type**, which has the same function as **CONVERT(expr, type)**. **type** supports the following data types:

- BINARY[(N)]

  If **expr** is **NULL**, the return value type is **BINARY(0)**. If **expr** is not **NULL**, the return value type is **VARBINARY**. If the optional length *N* is specified, the part that exceeds *N* is truncated, and the part that is less than *N* is padded with 0x00. If the optional length *N* is not specified, the length of the return value is determined based on the result of the expression. If the length of the return value exceeds the upper limit of **BINARY**, the return value type is **BLOB**. If the length is still insufficient, the return value type is **LONGBLOB**. The value of *N* is affected by the GUC parameter **max_allowed_packet**. The range of the maximum value is Min(max_allowed_packet, 1073741819).

- CHAR[(N)]

  If **expr** is **NULL**, the return value type is CHAR(0). If **expr** is not **NULL**, the return value type is VARCHAR. If the optional length *N* is specified, the part that exceeds *N* is truncated, and the part that is less than *N* is not padded. If the optional length *N* is not specified, the length of the return value is determined based on the result of the expression. If the length of the return value exceeds the upper limit of **VARCHAR**, the return value type is TEXT. If the length is still insufficient, the return value type is LONGTEXT.

- DATE

  The return value type is DATE.

- DATETIME[(M)]

  The return value type is DATETIME. **M** is used to determine the precision of seconds. The value range is [0,6].

- TIME[(N)]

  The return value type is TIME. **M** is used to determine the precision of seconds. The value range is [0,6].

- DECIMAL[M [, D])]

  The return value type is DECIMAL. **M** is used to determine the maximum number of digits (precision). The maximum value is **65**, and the default value is **10**. **D** is used to determine the number of digits after the decimal point. The maximum value is **30**, and the default value is **0**.

- DOUBLE

  The return value type is DOUBLE.

- FLOAT[(p)]

  If the precision p is not specified, the return type is FLOAT. If the precision p is greater than or equal to 0 and is less than 24, the return value type is FLOAT.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If the precision p is greater than or equal to 24, the return value type is DOUBLE. If the precision p is less than 0, an error is returned.

- SIGNED[INTEGER]

  The return value type is signed BIGINT.

- UNSIGNED[INTEGER]

  The return value type is BIGINT without signs.

Examples:

```
m_db=# SELECT CAST('abc' AS BINARY(2));
WARNING:  Truncated incorrect binary(2) value: 'abc'
CONTEXT:  referenced column: cast
 cast
------
 ab
(1 row)

m_db=# SELECT CAST('abc' AS CHAR(10));
 cast
------
 abc
(1 row)

m_db=# SELECT CAST('2023/1/1' AS DATE);
    cast
------------
 2023-01-01
(1 row)

m_db=# SELECT CAST('2023/01/01' AS DATETIME(2));
          cast
------------------------
 2023-01-01 00:00:00.00
(1 row)

m_db=#  SELECT CAST('09:23:14' AS TIME(2));
    cast
-------------
 09:23:14.00
(1 row)

m_db=# SELECT CAST(12.34567 AS DECIMAL(10,4));
  cast
---------
 12.3457
(1 row)

m_db=# SELECT CAST(-12 AS SIGNED);
 cast
------
  -12
(1 row)

m_db=# SELECT CAST(-12 AS UNSIGNED);
         cast
----------------------
 18446744073709551604
(1 row)
```

CONVERT

- CONVERT(expr, type)

  Description: Explicitly converts **expr** to **type**, which has the same function as **CAST(expr, type)**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Examples:

```
m_db=# SELECT CONVERT('abc', BINARY(2));
WARNING:  Truncated incorrect binary(2) value: 'abc'
CONTEXT:  referenced column: convert
 convert
---------
 ab
(1 row)

m_db=# SELECT CONVERT('abc', CHAR(10));
 convert
---------
 abc
(1 row)

m_db=# SELECT CONVERT('2023/1/1', DATE);
  convert
------------
 2023-01-01
(1 row)

m_db=# SELECT CONVERT('2023/01/01', DATETIME(2));
        convert
-----------------------
 2023-01-01 00:00:00.00
(1 row)

m_db=# SELECT CONVERT('09:23:14', TIME(2));
   convert
-------------
 09:23:14.00
(1 row)

m_db=# SELECT CONVERT(12.34567, DECIMAL(10,4));
 convert
---------
 12.3457
(1 row)

m_db=# SELECT CONVERT(-12, SIGNED);
 convert
---------
     -12
(1 row)

m_db=# SELECT CONVERT(-12, UNSIGNED);
       convert
----------------------
 18446744073709551604
(1 row)
```

- CONVERT(expr USING transcoding_name)

  Description: Converts the expression **expr** to the character set specified by **transcoding_name**.

  Examples:

  ```
  m_db=# SELECT CONVERT('abc' USING 'latin1');
   convert
  ---------
   abc
  (1 row)
  ```

4.5.10 Aggregate Functions

An aggregate function evaluates a set of values and returns a single value. Except **COUNT(*)**, aggregate functions ignore the value **NULL**. Aggregate functions are usually used with the **GROUP BY** clause.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

AVG

AVG([ALL | DISTINCT] expr)

Description: Returns the average value of the input **expr**. **NULL** will be ignored.

Parameter type: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, DECIMAL, FLOAT, or DOUBLE.

Return value type:

- The type of the return value depends on the type of the input parameter. If the input parameter is an exact value (**INT** or **DECIMAL**), **DECIMAL** is returned. If the input parameter is an approximate value (**FLOAT** or **DOUBLE**), **DOUBLE** is returned.
- If all values in **expr** are **NULL**, **AVG()** returns **NULL**.
- If **ALL** is specified or **ALL** and **DISTINCT** are not specified, the average value of all non-null values in **expr** is returned.
- If **DISTINCT** is specified, the average value of different values in **expr** is returned.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT age, AVG(salary) FROM m_test GROUP BY (age);
 age |  avg
-----+--------
   1 | 1.6667
   2 |
(2 rows)

m_db=# SELECT age, AVG(DISTINCT salary) FROM m_test GROUP BY (age);
 age |  avg
-----+--------
   1 | 1.5000
   2 |
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

BIT_AND

BIT_AND([ALL] expr)

Description: Returns the bitwise AND of all values in **expr**, ignoring **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter type: **BIT_AND** needs to use **BIGINT** as the parameter. Parameters of other types will be converted to **BIGINT** and may be truncated.

Return value type:

- The return value type is BIGINT.
- If all values in **expr** are **NULL**, **BIT_AND()** returns a neutral value (all bits are set to **1**).
- **ALL** is optional and does not affect the result.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT BIT_AND(ALL salary) FROM m_test GROUP BY age;
       bit_and
----------------------
 18446744073709551615
 0
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

BIT_OR

BIT_OR([ALL] expr)

Description: Returns the bitwise OR of all values in **expr**, ignoring **NULL**.

Parameter type: **BIT_OR** needs to use **BIGINT** as the parameter. Parameters of other types will be converted to **BIGINT** and may be truncated.

Return value type:

- The return value type is BIGINT.
- If all values in **expr** are **NULL**, **BIT_OR()** returns a neutral value (all bits are set to **0**).
- **ALL** is optional and does not affect the result.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT BIT_OR(ALL salary) FROM m_test GROUP BY age;
 bit_or
--------
 0
 3
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

BIT_XOR

BIT_XOR([ALL] expr)

Description: Returns the bitwise XOR of all values in **expr**, ignoring **NULL**.

Parameter type: **BIT_XOR** needs to use **BIGINT** as the parameter. Parameters of other types will be converted to **BIGINT** and may be truncated.

Return value type:

- The return value type is BIGINT.
- If all values in **expr** are **NULL**, **BIT_OR()** returns a neutral value (all bits are set to **0**).
- **ALL** is optional and does not affect the result.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT BIT_XOR(ALL salary) FROM m_test GROUP BY age;
 bit_xor
---------
 0
 1
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

COUNT

- COUNT([ALL] expr)

  Description: Returns the number of non-null values in **expr**.

  Parameter type: The input parameter can be of any type.

  Return value type:

  – The return value type is BIGINT.
  – If all values in **expr** are **NULL**, **COUNT()** returns **0**.
  – **ALL** is optional and does not affect the result.
  – COUNT(*) is special. It returns the number of all values in **expr**, whether **NULL** or not.

  Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT COUNT(ALL salary) FROM m_test GROUP BY age;
 count
-------
     3
     0
(2 rows)

m_db=# SELECT COUNT(*) FROM m_test GROUP BY age;
 count
-------
     4
     2
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

- COUNT(DISTINCT expr1 [, expr2, ..., exprN])

  Description: Returns the number of different non-null values in **expr**.

  Parameter type: The input parameter can be of any type.

  Return value type:

  – The return value type is BIGINT.
  – If all values in **expr** are **NULL**, **0** is returned.

  Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT COUNT(DISTINCT salary) FROM m_test GROUP BY age;
 count
-------
     2
     0
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

GROUP_CONCAT

The syntax of GROUP_CONCAT is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
          [ORDER BY {unsigned_integer | col_name | expr}
              [ASC | DESC] [,col_name ...]]
          [SEPARATOR str_val])
```

Description: Concatenates values in **expr** and returns a string. **NULL** is ignored.

- When **DISTINCT** is specified, duplicate values in **expr** are eliminated.
- ORDER BY: Sorts the values in **expr** by specifying a column.
- ASC | DESC: **ASC** indicates ascending order, and **DESC** indicates descending order. Ascending order is used by default.
- SEPARATOR: separator used for concatenating strings. Comma (,) is used by default.
- If both **DISTINCT** and **ORDER BY** are specified, the **DISTINCT** expression must contain the **ORDER BY** expression. Otherwise, an error is reported.
- GROUP_CONCAT(... **ORDER BY** *Number***)** does not indicate the sequence of the parameter. The number is only a constant expression, which is equivalent to no sorting.
- Parameter **group_concat_max_len** is used to limit the maximum length of **GROUP_CONCAT**. If the length exceeds the maximum, the system truncates the length. Currently, the maximum length that can be returned is 1073741823.

Return value type: TEXT

Examples:

```
m_db=# CREATE TABLE m_test(id INT, name CHAR) DISTRIBUTE BY HASH(id);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES (1, 'a'), (1, 'b'), (2, 'd'), (2, 'c'), (3, 'a'), (3, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 id | name
----+------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 1 | a
 1 | b
 2 | d
 2 | c
 3 | a
 3 |
(6 rows)

m_db=# SELECT GROUP_CONCAT(name SEPARATOR '') FROM m_test;
 group_concat
--------------
 abdca
(1 row)

m_db=# SELECT GROUP_CONCAT(DISTINCT id, name ORDER BY id DESC) FROM m_test;
   group_concat
------------------
 3a,2c,2d,1a,1b
(1 row)

m_db=# DROP TABLE m_test;
DROP TABLE
```

MAX

MAX([ALL | DISTINCT] expr)

Description: Returns the maximum value of **expr**. **NULL** will be ignored.

Parameter type: any number, string, or datetime.

Return value type:

- Return value type: same as the parameter type
- If all values in **expr** are **NULL**, **MAX()** returns **NULL**.
- If **ALL** is specified or **ALL** and **DISTINCT** are not specified, the maximum value of all non-null values in **expr** is returned.
- If **DISTINCT** is specified, the maximum value of different non-null values in **expr** is returned, which is the same as the result when **DISTINCT** is not specified.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT MAX(DISTINCT salary) FROM m_test GROUP BY age;
 max
-----
   2
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

MIN

MIN([ALL | DISTINCT] expr)

Description: Returns the minimum value of **expr**. **NULL** will be ignored.

Parameter type: any number, string, or datetime.

Return value type:

- Return value type: same as the parameter type
- If all values in **expr** are **NULL**, **MIN()** returns **NULL**.
- If **ALL** is specified or **ALL** and **DISTINCT** are not specified, the minimum value of all non-null values in **expr** is returned.
- If **DISTINCT** is specified, the minimum value of different non-null values in **expr** is returned, which is the same as the result when **DISTINCT** is not specified.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT MIN(DISTINCT salary) FROM m_test GROUP BY age;
 min
-----
   1

(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

SUM

SUM([ALL | DISTINCT] expr)

Description: Returns the sum of input **expr**. **NULL** will be ignored.

Parameter type: TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, DECIMAL, FLOAT, or DOUBLE.

Return value type:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The type of the return value depends on the type of the input parameter. If the input parameter is an exact value (**INT** or **DECIMAL**), **DECIMAL** is returned. If the input parameter is an approximate value (**FLOAT** or **DOUBLE**), **DOUBLE** is returned.
- If all values in **expr** are **NULL**, **SUM()** returns **NULL**.
- If **ALL** is specified or **ALL** and **DISTINCT** are not specified, the sum of all non-null values in **expr** is returned.
- If **DISTINCT** is specified, the sum of different non-null values in **expr** is returned.

Examples:

```
m_db=# CREATE TABLE m_test(age INT, salary INT) DISTRIBUTE BY HASH(age);
CREATE TABLE

m_db=# INSERT INTO m_test VALUES(1, NULL), (1, 1), (1, 2), (1, 2), (2, NULL), (2, NULL);
INSERT 0 6

m_db=# SELECT * FROM m_test;
 age | salary
-----+--------
   1 |
   1 |      1
   1 |      2
   1 |      2
   2 |
   2 |
(6 rows)

m_db=# SELECT age, SUM(salary) FROM m_test GROUP BY (age);
 age | sum
-----+-----
   1 |   5
   2 |
(2 rows)

m_db=# SELECT age, SUM(DISTINCT salary) FROM m_test GROUP BY (age);
 age | sum
-----+-----
   1 |   3
   2 |
(2 rows)

m_db=# DROP TABLE m_test;
DROP TABLE
```

4.5.11 Encryption and Decryption Functions

AES_DECRYPT

AES_DECRYPT(crypt_str, key_str[, init_vector][, kdf_name][, salt][, info | iterations])

Description: Based on AES algorithm, the encrypted string **crypt_str** is decrypted using the decryption password **key_str**, initialization vector **init_vector**, and KDF-related options (**kdf_name**, **salt**, and **info**/**iterations**).

Parameters:

- **crypt_str**: string to be decrypted. If **crypt_str** is set to **NULL**, the function returns **NULL**.
- **key_str**: decryption password. If **key_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

bit secure random number as the key string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block_encryption_mode**).

- **init_vector**: An initialization variable is provided for the required block encryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **str** nor **key_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV for each encryption is unique in OFB mode and that the IV for each encryption is unpredictable in CBC or CFB mode.
- **kdf_name**: KDF algorithm to be used. The value can be **PBKDF2** or **HKDF**.
- **salt**: used together with the original password to generate a key.
- **info**/**iterations**: For HKDF, **INFO** is the parameter specified by this column. For PBKDF2, **ITERATIONS** is the number of iterations specified by this column.

Return value type: LONGBLOB

Examples:

```
m_db=# SELECT
AES_DECRYPT(AES_ENCRYPT('huwei123','123456vfhex4dyu','vdaladhjsadadabcdef','hkdf'),'123456vfhex4dyu',
'vdaladhjsadadabcdef','hkdf');
 aes_decrypt
-------------
 huwei123
(1 row)
```

NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
- Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. When using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. After you finish using the logs, delete them as soon as possible.
- Do not call the function when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
- Ensure that the values of **block_encryption_mode**, **key_str**, and **iv** are the same as those during encryption.
- Due to encoding differences, encrypted data cannot be directly copied from the gsql client for decryption. In this scenario, the decryption result may be different from the character string before encryption.
- The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If the setting is SQL_ASCII, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to SQL_ASCII, the encoding of the client must also be set to SQL_ASCII. Otherwise, an error is reported. The database cannot automatically convert or verify non-ASCII characters.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

AES_ENCRYPT

AES_ENCRYPT(str, key_str[, init_vector][, kdf_name][, salt][, info | iterations])

Description: Based on AES algorithm, the string **str** is encrypted using the encryption password **key_str**, initialization vector **init_vector**, and KDF-related options (**kdf_name**, **salt**, and **info**/**iterations**).

Parameters:

- **str**: string to be encrypted. If **str** is set to **NULL**, the function returns **NULL**.
- **key_str**: encryption password. If **key_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-bit secure random number as the key string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block_encryption_mode**).
- **init_vector**: An initialization variable is provided for the required block encryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **str** nor **key_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV for each encryption is unique in OFB mode and that the IV for each encryption is unpredictable in CBC or CFB mode.
- **kdf_name**: KDF algorithm to be used. The value can be **PBKDF2** or **HKDF**.
- **salt**: used together with the original password to generate a key.
- **info**/**iterations**: For HKDF, **INFO** is the parameter specified by this column. For PBKDF2, **ITERATIONS** is the number of iterations specified by this column.

Return value type: LONGBLOB

Examples:

```
m_db=# SELECT AES_ENCRYPT('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456');
   aes_encrypt
--------------------
 Z_Z\x7F\\x7Fy\x1D
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- An encryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
- Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. When using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. After you finish using the logs, delete them as soon as possible.
- Do not call the function when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
- The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If the setting is SQL_ASCII, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to SQL_ASCII, the encoding of the client must also be set to SQL_ASCII. Otherwise, an error is reported. The database cannot automatically convert or verify non-ASCII characters.

SHA

SHA(str)

Description: Calculates the SHA-1 160-bit checksum of string **str**.

Parameter: **str**, which is a string or number.

Return value: string. The value is a string of 40 hexadecimal digits.

Examples:

```
m_db=# SELECT SHA('abc');
                 sha
------------------------------------------
 a9993e364706816aba3e25717850c26c9cd0d89d
(1 row)
```

SHA1

SHA1(str)

Description: SHA1 is the alias of the **SHA** function and has the same function and usage.

Examples:

```
m_db=# SELECT SHA1('abc');
                 sha1
------------------------------------------
 a9993e364706816aba3e25717850c26c9cd0d89d
(1 row)
```

SHA2

SHA2(str, hash_length)

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Description: Calculates the SHA-2 checksum of string **str**.

Parameters:

str: character string or number.

hash_length: corresponds to an SHA2 algorithm. The value can be **0** (SHA-256), **224** (SHA-224), **256** (SHA-256), **384** (SHA-384), or **512** (SHA-512). For other values, **NULL** is returned.

Return value: string. The value is a string of 40 hexadecimal digits.

Examples:

```
m_db=# SELECT SHA2('abc', 224);
                            sha2
------------------------------------------------------------
 23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7
(1 row)
```

4.5.12 Process Control Functions

IF

IF(expr1, expr2, expr3)

Description: If **expr1** is set to **TRUE** (that is, **expr1** is not equal to **0** and **expr1** is not **NULL**), **IF()** returns **expr2**. Otherwise, **IF()** returns **expr3**.

Examples:

```
m_db=# SELECT IF(true, 'ab', 1);
 if
----
 ab
(1 row)

m_db=# SELECT IF(1>2, 1, NULL);
 if
----

(1 row)

m_db=# SELECT IF(NULL, 'TRUE', 'FALSE');
  if
-------
 FALSE
(1 row)
```

IFNULL

IFNULL(expr1, expr2)

Description: If **expr1** is not **NULL**, **IFNULL()** returns **expr1**. If **expr1** is **NULL**, **IFNULL()** returns **expr2**.

Examples:

```
m_db=# select ifnull(12, 123);
 ifnull
--------
     12
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# select ifnull(null, 123);
 ifnull
--------
    123
(1 row)
```

NULLIF

NULLIF(expr1, expr2)

Description: Returns **NULL** if **expr1** is equal to **expr2**; otherwise, returns **expr1**.

Examples:

```
m_db=# select nullif(123, 123);
 nullif
--------

(1 row)

m_db=# select nullif(12, 123);
 nullif
--------
     12
(1 row)
```

4.5.13 Internal Functions

You are advised not to directly use internal functions of an M-compatible database.

attname_eq_mysql	avg_any	avg_collect	avg_final	binary_binary_mysql	binary_blob_mysql	binary_char_mysql
binary_datetime_mysql	binary_float_mysql	binary_in_mysql	binary_int4_mysql	binary_int8_mysql	binary_longblob_mysql	binary_mediumblob_mysql
binary_mysql	binary_numeric_mysql	binary_out_mysql	binary_recv_mysql	binary_send_mysql	binary_str_eq_mysql	binary_str_equal_mysql
binary_str_ge_mysql	binary_str_gt_mysql	binary_str_le_mysql	binary_str_lt_mysql	binary_str_ne_mysql	binary_text_mysql	binary_tinyblob_mysql
binary_typmodin_mysql	binary_typmodout_mysql	binary_uint8_mysql	bit_binary_mysql	bit_blob_mysql	bit_char_mysql	bit_date_mysql
bit_datetime_mysql	bit_double_mysql	bit_equal_mysql	bit_float_mysql	bit_func_binary_mysql	bit_func_char_mysql	bit_func_varbinary_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

bit_func_varchar_mysql	bit_in_mysql	bit_int4_mysql	bit_int8_mysql	bit_mediumint_mysql	bit_mysql	bit_numeric_mysql
bit_out_mysql	bit_recv_mysql	bit_send_mysql	bit_smallint_mysql	bit_text_mysql	bit_time_mysql	bit_timestamp_mysql
bit_tinyint_mysql	bit_typmodin_mysql	bit_typmodout_mysql	bit_uint1_mysql	bit_uint2_mysql	bit_uint3_mysql	bit_uint4_mysql
bit_uint8_mysql	bit_varbinary_mysql	bit_varchar_mysql	bit_year_mysql	blob_binary_mysql	blob_bit_mysql	blob_char_mysql
blob_date_mysql	blob_datetime_mysql	blob_in_mysql	blob_int4_mysql	blob_int8_mysql	blob_numeric_mysql	blob_out_mysql
blob_recv_mysql	blob_send_mysql	blob_text_mysql	blob_uint8_mysql	bool_binary_mysql	bool_bit_mysql	bool_bool_equal_mysql
bool_char_mysql	bool_double_mysql	bool_int1_mysql	bool_int2_mysql	bool_int4_mysql	bool_int8_mysql	bool_mediumint_mysql
bool_numeric_mysql	bool_real_mysql	bool_uint1_mysql	bool_uint2_mysql	bool_uint3_mysql	bool_uint4_mysql	bool_uint8_mysql
booltext_mysql	booltext_mysql	bpchar_binary_mysql	bpchar_bit_mysql	bpchar_bpchar_mysql	bpchar_date_mysql	bpchar_datetime_mysql
bpchar_eq_mysql	bpchar_equal_mysql	bpchar_ge_mysql	bpchar_gt_mysql	bpchar_le_mysql	bpchar_lt_mysql	bpchar_mysql
bpchar_ne_mysql	bpchar_numeric_mysql	bpchar_out_mysql	bpchar_recv_mysql	bpchar_send_mysql	bpchar_time_mysql	bpchar_timestamp_mysql
bpchar_typmodin_mysql	bpchar_year_mysql	cast_to_binary	cast_to_cstring	cast_to_datetime	cast_to_float8	cast_to_int8
cast_to_numeric	date_binary_mysql	date_bit_mysql	date_blob_mysql	date_bpchar_mysql	date_date_mysql	date_datetime_eq_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

date_datetime_equal_mysql	date_datetime_ge_mysql	date_datetime_gt_mysql	date_datetime_le_mysql	date_datetime_lt_mysql	date_datetime_mysql	date_datetime_ne_mysql
date_equal_mysql	date_float4_mysql	date_float8_mysql	date_func_bit_mysql	date_func_int1_mysql	date_func_int2_mysql	date_func_int3_mysql
date_func_int4_mysql	date_func_int8_mysql	date_func_uint1_mysql	date_func_uint2_mysql	date_func_uint3_mysql	date_func_uint4_mysql	date_func_uint8_mysql
date_func_year_mysql	date_in_mysql	date_int1_mysql	date_int2_mysql	date_int3_mysql	date_int4_mysql	date_int8_mysql
date_literal_mysql	date_longblob_mysql	date_longtext_mysql	date_mediumblob_mysql	date_mediumtext_mysql	date_numeric_mysql	date_out_mysql
date_row_bit_mysql	date_row_float4_mysql	date_row_float8_mysql	date_row_int1_mysql	date_row_int2_mysql	date_row_int3_mysql	date_row_int4_mysql
date_row_int8_mysql	date_row_numeric_mysql	date_row_time_mysql	date_row_uint1_mysql	date_row_uint2_mysql	date_row_uint3_mysql	date_row_uint4_mysql
date_row_uint8_mysql	date_row_year_mysql	date_text_mysql	date_time_mysql	date_timestamp_eq_mysql	date_timestamp_equal_mysql	date_timestamp_ge_mysql
date_timestamp_gt_mysql	date_timestamp_le_mysql	date_timestamp_lt_mysql	date_timestamp_mysql	date_timestamp_ne_mysql	date_tinyblob_mysql	date_tinytext_mysql
date_uint1_mysql	date_uint2_mysql	date_uint3_mysql	date_uint4_mysql	date_uint8_mysql	date_varbinary_mysql	date_varchar_mysql
date_year_mysql	datetime_binary_mysql	datetime_bit_mysql	datetime_blob_mysql	datetime_bpchar_mysql	datetime_date_eq_mysql	datetime_date_equal_mysql
datetime_date_ge_mysql	datetime_date_gt_mysql	datetime_date_le_mysql	datetime_date_lt_mysql	datetime_date_mysql	datetime_date_ne_mysql	datetime_float4_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

datetime_float8_mysql	datetime_func_bit_mysql	datetime_func_year_mysql	datetime_in_mysql	datetime_int1_mysql	datetime_int2_mysql	datetime_int3_mysql
datetime_int4_mysql	datetime_int8_mysql	datetime_literal_mysql	datetime_longblob_mysql	datetime_longtext_mysql	datetime_mediumblob_mysql	datetime_mediumtext_mysql
datetime_numeric_mysql	datetime_out_mysql	datetime_recv_mysql	datetime_row_bit_mysql	datetime_row_date_mysql	datetime_row_float4_mysql	datetime_row_float8_mysql
datetime_row_int1_mysql	datetime_row_int2_mysql	datetime_row_int3_mysql	datetime_row_int4_mysql	datetime_row_int8_mysql	datetime_row_numeric_mysql	datetime_row_time_mysql
datetime_row_uint1_mysql	datetime_row_uint2_mysql	datetime_row_uint3_mysql	datetime_row_uint4_mysql	datetime_row_uint8_mysql	datetime_row_year_mysql	datetime_scale_mysql
datetime_send_mysql	datetime_text_mysql	datetime_time_mysql	datetime_timestamp_eq_mysql	datetime_timestamp_equal_mysql	datetime_timestamp_ge_mysql	datetime_timestamp_gt_mysql
datetime_timestamp_le_mysql	datetime_timestamp_lt_mysql	datetime_timestamp_mysql	datetime_timestamp_ne_mysql	datetime_tinyblob_mysql	datetime_tinytext_mysql	datetime_uint1_mysql
datetime_uint2_mysql	datetime_uint3_mysql	datetime_uint4_mysql	datetime_uint8_mysql	datetime_varbinary_mysql	datetime_varchar_mysql	datetime_year_mysql
datetimetypmodin_mysql	datetimetypmodout_mysql	float4_binary_mysql	float4_bit_mysql	float4_blob_mysql	float4_bpchar_mysql	float4_date_mysql
float4_datetime_mysql	float4_float8_mysql	float4_in_mysql	float4_int1_mysql	float4_int2_mysql	float4_int3_mysql	float4_int4_mysql
float4_int8_mysql	float4_mysql	float4_numeric_mysql	float4_out_mysql	float4_recv_mysql	float4_row_blob_mysql	float4_row_text_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

float4_send_mysql	float4_text_mysql	float4_time_mysql	float4_timestamp_mysql	float4_typmod_in_mysql	float4_typmod_out_mysql	float4_uint1_mysql
float4_uint2_mysql	float4_uint3_mysql	float4_uint4_mysql	float4_uint8_mysql	float4_varbinary_mysql	float4_varchar_mysql	float4_year_mysql
float48equal_mysql	float4equal_mysql	float8_binary_mysql	float8_bit_mysql	float8_blob_mysql	float8_bpchar_mysql	float8_date_mysql
float8_datetime_mysql	float8_float4_mysql	float8_in_mysql	float8_int1_mysql	float8_int2_mysql	float8_int3_mysql	float8_int4_mysql
float8_int8_mysql	float8_mysql	float8_numeric_mysql	float8_out_mysql	float8_recv_mysql	float8_send_mysql	float8_text_mysql
float8_time_mysql	float8_timestamp_mysql	float8_typmod_in_mysql	float8_typmod_out_mysql	float8_uint1_mysql	float8_uint2_mysql	float8_uint3_mysql
float8_uint4_mysql	float8_uint8_mysql	float8_varbinary_mysql	float8_varchar_mysql	float8_year_mysql	float84equal_mysql	float8eq_mysql
float8ge_mysql	float8gt_mysql	float8le_mysql	float8lt_mysql	float8ne_mysql	i1tobinary_mysql	i1tobit_mysql
i1toblob_mysql	i1tof4_mysql	i1tof8_mysql	i1toi2_mysql	i1toi3_mysql	i1toi4_mysql	i1toi8_mysql
i1toui1_mysql	i1toui2_mysql	i1toui3_mysql	i1toui4_mysql	i1toui8_mysql	i1tovarbinary_mysql	i1typmodin_mysql
i1typmodout_mysql	i2tobinary_mysql	i2tobit_mysql	i2toblob_mysql	i2tod_mysql	i2tof_mysql	i2toi1_mysql
i2toi3_mysql	i2toi4_mysql	i2toui1_mysql	i2toui2_mysql	i2toui3_mysql	i2toui4_mysql	i2toui8_mysql
i2tovarbinary_mysql	i2typmodin_mysql	i2typmodout_mysql	i3tobinary_mysql	i3tobit_mysql	i3toblob_mysql	i3tobpchar_mysql
i3tof4_mysql	i3tof8_mysql	i3toi1_mysql	i3toi2_mysql	i3toi4_mysql	i3toi8_mysql	i3tonum_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

i3totext_mysql	i3toui3_mysql	i3tovarbinary_mysql	i3tovarchar_mysql	i4tobinary_mysql	i4toblob_mysql	i4tod_mysql
i4tof_mysql	i4toi1_mysql	i4toi3_mysql	i4toui1_mysql	i4toui2_mysql	i4toui3_mysql	i4toui4_mysql
i4toui8_mysql	i4tovarbinary_mysql	i4typmodin_mysql	i4typmodout_mysql	i8tobinary_mysql	i8toblob_mysql	i8tod_mysql
i8tof_mysql	i8toi1_mysql	i8toi3_mysql	i8toui1_mysql	i8toui2_mysql	i8toui3_mysql	i8toui4_mysql
i8toui8_mysql	i8tovarbinary_mysql	i8typmodin_mysql	i8typmodout_mysql	int1_bpchar_mysql	int1_equal_mysql	int1_int16_mysql
int1_numeric_mysql	int1_text_mysql	int1_varchar_mysql	int16_int1_mysql	int16_int3_mysql	int16_uint1_mysql	int16_uint2_mysql
int16_uint3_mysql	int16_uint4_mysql	int16_uint8_mysql	int1in_mysql	int1out_mysql	int1recv_mysql	int1send_mysql
int2_bpchar_mysql	int2_int2_equal_mysql	int2_int3_eq_mysql	int2_int3_equal_mysql	int2_int3_ge_mysql	int2_int3_gt_mysql	int2_int3_le_mysql
int2_int3_lt_mysql	int2_int3_ne_mysql	int2_int4_equal_mysql	int2_int8_equal_mysql	int2_numeric_mysql	int2_text_mysql	int2_uint_eq_mysql
int2_uint_equal_mysql	int2_uint_ge_mysql	int2_uint_gt_mysql	int2_uint_le_mysql	int2_uint_lt_mysql	int2_uint_ne_mysql	int2_varchar_mysql
int28_mysql	int2in_mysql	int2out_mysql	int3_int16_mysql	int3_int2_eq_mysql	int3_int2_equal_mysql	int3_int2_ge_mysql
int3_int2_gt_mysql	int3_int2_le_mysql	int3_int2_lt_mysql	int3_int2_ne_mysql	int3_int4_eq_mysql	int3_int4_equal_mysql	int3_int4_ge_mysql
int3_int4_gt_mysql	int3_int4_le_mysql	int3_int4_lt_mysql	int3_int4_ne_mysql	int3_int8_eq_mysql	int3_int8_equal_mysql	int3_int8_ge_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

int3_int8_gt_mysql	int3_int8_le_mysql	int3_int8_lt_mysql	int3_int8_ne_mysql	int3_uint_eq_mysql	int3_uint_equal_mysql	int3_uint_ge_mysql
int3_uint_gt_mysql	int3_uint_le_mysql	int3_uint_lt_mysql	int3_uint_ne_mysql	int4_bit_mysql	int4_bpchar_mysql	int4_int2_equal_mysql
int4_int3_eq_mysql	int4_int3_equal_mysql	int4_int3_ge_mysql	int4_int3_gt_mysql	int4_int3_le_mysql	int4_int3_lt_mysql	int4_int3_ne_mysql
int4_int4_equal_mysql	int4_int8_equal_mysql	int4_numeric_mysql	int4_text_mysql	int4_time_mysql	int4_uint_eq_mysql	int4_uint_equal_mysql
int4_uint_ge_mysql	int4_uint_gt_mysql	int4_uint_le_mysql	int4_uint_lt_mysql	int4_uint_ne_mysql	int4_varchar_mysql	int4_year_mysql
int48_mysql	int4in_mysql	int4out_mysql	int8_bit_mysql	int8_bpchar_mysql	int8_date_mysql	int8_datetime_mysql
int8_int2_equal_mysql	int8_int3_eq_mysql	int8_int3_equal_mysql	int8_int3_ge_mysql	int8_int3_gt_mysql	int8_int3_le_mysql	int8_int3_lt_mysql
int8_int3_ne_mysql	int8_int4_equal_mysql	int8_numeric_mysql	int8_text_mysql	int8_time_mysql	int8_timestamp_mysql	int8_uint_eq_mysql
int8_uint_equal_mysql	int8_uint_ge_mysql	int8_uint_gt_mysql	int8_uint_le_mysql	int8_uint_lt_mysql	int8_uint_ne_mysql	int8_varchar_mysql
int84_mysql	int8eq_mysql	int8ge_mysql	int8gt_mysql	int8in_mysql	int8le_mysql	int8lt_mysql
int8ne_mysql	int8out_mysql	longblob_binary_mysql	longblob_char_mysql	longblob_date_mysql	longblob_datetime_mysql	longblob_in_mysql
longblob_int4_mysql	longblob_int8_mysql	longblob_numeric_mysql	longblob_out_mysql	longblob_recv_mysql	longblob_send_mysql	longblob_text_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

longblob_uint8_mysql	longtext_binary_mysql	longtext_char_mysql	longtext_date_mysql	longtext_datetime_mysql	longtext_float_mysql	longtext_in_mysql
longtext_int4_mysql	longtext_numeric_mysql	longtext_out_mysql	longtext_recv_mysql	longtext_send_mysql	longtext_uint8_mysql	max_any
mediumblob_binary_mysql	mediumblob_char_mysql	mediumblob_date_mysql	mediumblob_datetime_mysql	mediumblob_in_mysql	mediumblob_int4_mysql	mediumblob_int8_mysql
mediumblob_numeric_mysql	mediumblob_out_mysql	mediumblob_recv_mysql	mediumblob_send_mysql	mediumblob_text_mysql	mediumblob_uint8_mysql	mediumint_eq_mysql
mediumint_equal_mysql	mediumint_ge_mysql	mediumint_gt_mysql	mediumint_le_mysql	mediumint_lt_mysql	mediumint_ne_mysql	mediumintin_mysql
mediumintout_mysql	mediumintrecv_mysql	mediumintsend_mysql	mediumintttypmodin_mysql	mediumintttypmodout_mysql	mediumtext_binary_mysql	mediumtext_char_mysql
mediumtext_date_mysql	mediumtext_datetime_mysql	mediumtext_float_mysql	mediumtext_in_mysql	mediumtext_int4_mysql	mediumtext_numeric_mysql	mediumtext_out_mysql
mediumtext_recv_mysql	mediumtext_send_mysql	mediumtext_uint8_mysql	min_any	numeric_bit_mysql	numeric_blob_mysql	numeric_bpchar_mysql
numeric_date_mysql	numeric_datetime_mysql	numeric_eq_mysql	numeric_float4_mysql	numeric_float8_mysql	numeric_ge_mysql	numeric_gt_mysql
numeric_in_mysql	numeric_int1_mysql	numeric_int2_mysql	numeric_int4_mysql	numeric_int8_mysql	numeric_le_mysql	numeric_lt_mysql
numeric_mediumint_mysql	numeric_ne_mysql	numeric_out_mysql	numeric_recv_mysql	numeric_send_mysql	numeric_text_mysql	numeric_time_mysql
numeric_timestamp_mysql	numeric_typ_mod_in_mysql	numeric_typ_mod_out_mysql	numeric_uint1_mysql	numeric_uint2_mysql	numeric_uint3_mysql	numeric_uint4_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

numeric_uint8_mysql	numeric_year_mysql	str_eq_mysql	str_equal_mysql	str_ge_mysql	str_gt_mysql	str_le_mysql
str_lt_mysql	str_ne_mysql	sum_any	sum_final	text_binary_mysql	text_bit_mysql	text_blob_mysql
text_date_mysql	text_datetime_mysql	text_eq_mysql	text_float4_mysql	text_float8_mysql	text_ge_mysql	text_gt_mysql
text_in_mysql	text_int1_mysql	text_int2_mysql	text_int4_mysql	text_int8_mysql	text_le_mysql	text_longblob_mysql
text_longtext_mysql	text_lt_mysql	text_mediumblob_mysql	text_mediumint_mysql	text_mediumtext_mysql	text_mysql	text_ne_mysql
text_numeric_mysql	text_out_mysql	text_recv_mysql	text_send_mysql	text_text_mysql	text_time_mysql	text_timestamp_mysql
text_tinyblob_mysql	text_tinytext_mysql	text_uint1_mysql	text_uint2_mysql	text_uint3_mysql	text_uint4_mysql	text_uint8_mysql
text_varbinary_mysql	text_varchar_mysql	text_year_mysql	time_binary_mysql	time_bit_mysql	time_blob_mysql	time_bpchar_mysql
time_date_mysql	time_datetime_mysql	time_eq_mysql	time_float4_mysql	time_float8_mysql	time_format_mysql	time_func_bit_mysql
time_func_year_mysql	time_ge_mysql	time_gt_mysql	time_in_mysql	time_int1_mysql	time_int2_mysql	time_int3_mysql
time_int4_mysql	time_int8_mysql	time_le_mysql	time_literal_mysql	time_longblob_mysql	time_longtext_mysql	time_lt_mysql
time_mediumblob_mysql	time_mediumtext_mysql	time_ne_mysql	time_numeric_mysql	time_out_mysql	time_recv_mysql	time_row_bit_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

time_row_date_mysql	time_row_datetime_mysql	time_row_float4_mysql	time_row_float8_mysql	time_row_int1_mysql	time_row_int2_mysql	time_row_int3_mysql
time_row_int4_mysql	time_row_int8_mysql	time_row_numeric_mysql	time_row_timestamp_mysql	time_row_uint1_mysql	time_row_uint2_mysql	time_row_uint3_mysql
time_row_uint4_mysql	time_row_uint8_mysql	time_row_year_mysql	time_scale_mysql	time_text_mysql	time_timestamp_mysql	time_tinyblob_mysql
time_tinytext_mysql	time_typmodout_mysql	time_uint1_mysql	time_uint2_mysql	time_uint3_mysql	time_uint4_mysql	time_uint8_mysql
time_varbinary_mysql	time_varchar_mysql	time_year_mysql	timestamp_binary_mysql	timestamp_bit_mysql	timestamp_blob_mysql	timestamp_bpchar_mysql
timestamp_date_eq_mysql	timestamp_date_equal_mysql	timestamp_date_ge_mysql	timestamp_date_gt_mysql	timestamp_date_le_mysql	timestamp_date_lt_mysql	timestamp_date_ne_mysql
timestamp_datetime_eq_mysql	timestamp_datetime_equal_mysql	timestamp_datetime_ge_mysql	timestamp_datetime_gt_mysql	timestamp_datetime_le_mysql	timestamp_datetime_lt_mysql	timestamp_datetime_mysql
timestamp_datetime_ne_mysql	timestamp_equal_mysql	timestamp_float4_mysql	timestamp_float8_mysql	timestamp_int1_mysql	timestamp_int2_mysql	timestamp_int3_mysql
timestamp_int4_mysql	timestamp_int8_mysql	timestamp_longblob_mysql	timestamp_longtext_mysql	timestamp_mediumblob_mysql	timestamp_mediumtext_mysql	timestamp_numeric_mysql
timestamp_out_mysql	timestamp_row_bit_mysql	timestamp_row_date_mysql	timestamp_row_float4_mysql	timestamp_row_float8_mysql	timestamp_row_int1_mysql	timestamp_row_int2_mysql
timestamp_row_int3_mysql	timestamp_row_int4_mysql	timestamp_row_int8_mysql	timestamp_row_numeric_mysql	timestamp_row_time_mysql	timestamp_row_uint1_mysql	timestamp_row_uint2_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

timestamp_row_uint3_mysql	timestamp_row_uint4_mysql	timestamp_row_uint8_mysql	timestamp_row_year_mysql	timestamp_scale_mysql	timestamp_text_mysql	timestamp_time_mysql
timestamp_tinyblob_mysql	timestamp_tinytext_mysql	timestamp_typmodout_mysql	timestamp_uint1_mysql	timestamp_uint2_mysql	timestamp_uint3_mysql	timestamp_uint4_mysql
timestamp_uint8_mysql	timestamp_varbinary_mysql	timestamp_varchar_mysql	timestamp_year_mysql	timestamp1_mysql	timestamp2_mysql	timestamptz_datetime_mysql
tinyblob_binary_mysql	tinyblob_char_mysql	tinyblob_date_mysql	tinyblob_datetime_mysql	tinyblob_in_mysql	tinyblob_int4_mysql	tinyblob_int8_mysql
tinyblob_numeric_mysql	tinyblob_out_mysql	tinyblob_recv_mysql	tinyblob_send_mysql	tinyblob_text_mysql	tinyblob_uint8_mysql	tinytext_binary_mysql
tinytext_char_mysql	tinytext_date_mysql	tinytext_datetime_mysql	tinytext_float_mysql	tinytext_in_mysql	tinytext_int4_mysql	tinytext_numeric_mysql
tinytext_out_mysql	tinytext_recv_mysql	tinytext_send_mysql	tinytext_uint8_mysql	ui1tof4_mysql	ui1tof8_mysql	ui1toi1_mysql
ui1toi2_mysql	ui1toi3_mysql	ui1toi4_mysql	ui1toi8_mysql	ui1toui2_mysql	ui1toui3_mysql	ui1toui4_mysql
ui1toui8_mysql	ui2tof4_mysql	ui2tof8_mysql	ui2toi1_mysql	ui2toi2_mysql	ui2toi3_mysql	ui2toi4_mysql
ui2toi8_mysql	ui2toui1_mysql	ui2toui3_mysql	ui2toui4_mysql	ui2toui8_mysql	ui3tof4_mysql	ui3tof8_mysql
ui3toi1_mysql	ui3toi2_mysql	ui3toi3_mysql	ui3toi4_mysql	ui3toi8_mysql	ui3toui1_mysql	ui3toui2_mysql
ui3toui4_mysql	ui3toui8_mysql	ui4tof4_mysql	ui4tof8_mysql	ui4toi1_mysql	ui4toi2_mysql	ui4toi3_mysql
ui4toi4_mysql	ui4toi8_mysql	ui4toui1_mysql	ui4toui2_mysql	ui4toui3_mysql	ui4toui8_mysql	ui8tof4_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

ui8tof8_mysql	ui8toi1_mysql	ui8toi2_mysql	ui8toi3_mysql	ui8toi4_mysql	ui8toi8_mysql	ui8toui1_mysql
ui8toui2_mysql	ui8toui3_mysql	ui8toui4_mysql	uint_int2_eq_mysql	uint_int2_equal_mysql	uint_int2_ge_mysql	uint_int2_gt_mysql
uint_int2_le_mysql	uint_int2_lt_mysql	uint_int2_ne_mysql	uint_int3_eq_mysql	uint_int3_equal_mysql	uint_int3_ge_mysql	uint_int3_gt_mysql
uint_int3_le_mysql	uint_int3_lt_mysql	uint_int3_ne_mysql	uint_int4_eq_mysql	uint_int4_equal_mysql	uint_int4_ge_mysql	uint_int4_gt_mysql
uint_int4_le_mysql	uint_int4_lt_mysql	uint_int4_ne_mysql	uint_int8_eq_mysql	uint_int8_equal_mysql	uint_int8_ge_mysql	uint_int8_gt_mysql
uint_int8_le_mysql	uint_int8_lt_mysql	uint_int8_ne_mysql	uint_uint_eq_mysql	uint_uint_equal_mysql	uint_uint_ge_mysql	uint_uint_gt_mysql
uint_uint_le_mysql	uint_uint_lt_mysql	uint_uint_ne_mysql	uint1_in_mysql	uint1_int16_mysql	uint1_numeric_mysql	uint1_out_mysql
uint1_recv_mysql	uint1_send_mysql	uint1_typmodin_mysql	uint1_typmodout_mysql	uint2_in_mysql	uint2_int16_mysql	uint2_numeric_mysql
uint2_out_mysql	uint2_recv_mysql	uint2_send_mysql	uint2_typmodin_mysql	uint2_typmodout_mysql	uint3_in_mysql	uint3_int16_mysql
uint3_numeric_mysql	uint3_out_mysql	uint3_recv_mysql	uint3_send_mysql	uint3_typmodin_mysql	uint3_typmodout_mysql	uint4_in_mysql
uint4_int16_mysql	uint4_numeric_mysql	uint4_out_mysql	uint4_recv_mysql	uint4_send_mysql	uint4_typmodin_mysql	uint4_typmodout_mysql
uint8_bit_mysql	uint8_in_mysql	uint8_int16_mysql	uint8_numeric_mysql	uint8_out_mysql	uint8_recv_mysql	uint8_send_mysql

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

uint8_typmodin_mysql	uint8_typmodout_mysql	varbinary_char_mysql	varbinary_datetime_mysql	varbinary_float_mysql	varbinary_in_mysql	varbinary_int_mysql
varbinary_mysql	varbinary_out_mysql	varbinary_recv_mysql	varbinary_send_mysql	varbinary_text_mysql	varbinary_typmodin_mysql	varbinary_typmodout_mysql
varbinary_uint8_mysql	varchar_binary_mysql	varchar_date_mysql	varchar_datetime_mysql	varchar_in_mysql	varchar_mysql	varchar_numeric_mysql
varchar_out_mysql	varchar_recv_mysql	varchar_send_mysql	varchar_time_mysql	varchar_timestamp_mysql	varchar_typmodin_mysql	varchar_uint8_mysql
varchar_year_mysql	year_binary_mysql	year_bit_mysql	year_blob_mysql	year_bpchar_mysql	year_date_mysql	year_datetime_mysql
year_eq_mysql	year_equal_mysql	year_float4_mysql	year_float8_mysql	year_ge_mysql	year_gt_mysql	year_in_mysql
year_int1_mysql	year_int2_mysql	year_int3_mysql	year_int4_mysql	year_int8_mysql	year_le_mysql	year_lt_mysql
year_ne_mysql	year_numeric_mysql	year_out_mysql	year_recv_mysql	year_send_mysql	year_text_mysql	year_time_mysql
year_timestamp_mysql	year_uint1_mysql	year_uint2_mysql	year_uint3_mysql	year_uint4_mysql	year_uint8_mysql	year_varbinary_mysql
year_varchar_mysql	yeartypmodin_mysql	yeartypmodout_mysql	biteq	bitge	bitgt	bitle
bitlt	bitne	booleq	boolge	boolgt	boolle	boollt
boolne	date_eq	date_ge	date_ge_timestamptz	date_gt	date_gt_timestamptz	date_le
date_le_timestamptz	date_lt	date_lt_timestamptz	date_ne	date_ne_timestamptz	eq_binary	eq_datetime

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

equal_binary	equal_datetime	equal_float	equal_int	equal_numeric	equal_string	equal_time
float48eq	float48ge	float48gt	float48le	float48lt	float48ne	float4eq
float4ge	float4gt	float4le	float4lt	float4ne	float84eq	float84ge
float84gt	float84le	float84lt	float84ne	ge_binary	ge_datetime	gt_binary
gt_datetime	int1eq	int1ge	int1gt	int1le	int1lt	int1ne
date_eq_timestamptz	int24eq	int24ge	int24gt	int24le	int24lt	int24ne
int28eq	int28ge	int28gt	int28le	int28lt	int28ne	int2eq
int2ge	int2gt	int2le	int2lt	int2ne	int2seteq	int2setge
int2setgt	int2setle	int2setlt	int2setne	int42eq	int42ge	int42gt
int42le	int42lt	int42ne	int48eq	int48ge	int48gt	int48le
int48lt	int48ne	int4eq	int4ge	int4gt	int4le	int4lt
int4ne	int4seteq	int4setge	int4setgt	int4setle	int4setlt	int4setne
int82eq	int82ge	int82gt	int82le	int82lt	int82ne	int84eq
int84ge	int84gt	int84le	int84lt	int84ne	int8seteq	int8setge
int8setgt	int8setle	int8setlt	int8setne	le_binary	le_datetime	lt_binary
lt_datetime	ne_binary	ne_datetime	textseteq	textsetge	textsetgt	textsetle
textsetlt	textsetne	timestamp_eq	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_timestamptz	timestamp_le	timestamp_le_timestamptz	timestamp_lt	timestamp_lt_timestamptz	timestamp_ne	timestamp_ne_timestamptz

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

plus_any	minus_any	mul_any	div_any	mod_any	neg_any	bit_or_any
bit_and_any	bit_xor_any	bit_neg_any	bit_shift_left_any	bit_shift_right_any	like_any	nlike_any
int_div_any	logical_oper_bool	bool_xor	-	-	-	-

4.5.14 Other Functions

ATTNAME_EQ_MYSQL

attname_eq_mysql(name1, name2)

Description: Returns **true** if **name1** is equal to **name2** (case-insensitive). Otherwise, returns **false**.

Return type: Boolean

Example:

```
m_db=# SELECT attname_eq_mysql('abc', 'ABC');
 attname_eq_mysql
------------------
 t
(1 row)
```

DATABASE

DATEBASE()

Description: Returns the name of the current database (schema).

Return value type: TEXT

Examples:

```
m_db=# SELECT DATABASE();
 database
----------
 public
(1 row)
```

SCHEMA

SCHEMA()

Description: Returns the name of the current schema (database).

Return value type: TEXT

Examples:

```
m_db=# SELECT SCHEMA();
 schema
--------
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 public
(1 row)
```

UUID

UUID()

Description: Returns a universally unique identifier (UUID) defined in RFC 4122, ISO/IEF 9834-8:2005, and related standards. The identifier is a string consisting of lowercase hexadecimal digits. The string consists of a group of 8-bit digits, three groups of 4-bit digits, and a group of 12-bit digits. A total of 32 digits represent 128 bits.

Return value type: VARCHAR

Examples:

```
m_db=# SELECT UUID();
                 uuid
--------------------------------------
 b28d6aaa-7a3a-c03c-f812-fa163e6cc336
(1 row)
```

UUID_SHORT

UUID_SHORT()

Description: Returns a short UUID under certain conditions. This identifier is a 64-bit unsigned integer.

The returned value is unique when the following conditions are met:

- The number of service nodes in the current cluster cannot exceed 256.
- You cannot set the system time of the server host between node restarts.
- The average number of UUID_SHORT() calls per second between node restarts is less than 16 million.

Return value type: UINT64

Examples:

```
m_db=# SELECT UUID_SHORT();
     uuid_short
----------------------
 13863614461119561729
(1 row)
```

4.6 Data Types

4.6.1 Overview

M-compatible databases support the following data types:

- Boolean type: BOOLEAN or BOOL. For details, see **Boolean Types**.
- Binary types: BINARY, VARBINARY, TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. For details, see **Table 4-74**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Character types: CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. For details, see **Table 4-75**.
- Time types: DATE, TIME, TIMESTAMP, DATETIME, and YEAR. For details, see **Table 4-76**.
- Bit string type: BIT. For details, see **Table 4-77**.
- Numeric types: TINYINT, SMALLINT, MEDIUMINT, INT/INTEGER, BIGINT, NUMERIC or DECIMAL, FLOAT, and DOUBLE. For details, see **Numeric Types**.

M-compatible databases support conversion between data types. For details, see **Type Conversion**.

M-compatible databases support constraints on data types. For details, see "MySQL Compatibility Description > M-compatible Mode" in *Compatibility Description*.

4.6.2 Boolean Types

BOOLEAN data type is used to store Boolean values **true** and **false**. The BOOLEAN or BOOL type is a synonym of the TINYINT type. Values other than **1** and **0** can be inserted into the BOOL or BOOLEAN column. **Table 4-73** lists the Boolean types supported by M-compatible databases.

Table 4-73 Boolean Types

Name	Description	Storage Space	Value
BOOLEAN BOOL	Boolean. The value ranges from –128 to 127.	1 byte	• **TRUE**, **1**, **true**, and all non-zero values. • **FALSE**, **0**, and **false**. • **null**.

NOTE

Boolean values **true** and **false** are recommended in SQL.

Examples

Boolean values are displayed using the numbers 1 and 0.

```
-- Create a table.
m_db=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
);

-- Insert data.
m_db=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

m_db=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

m_db=# INSERT INTO bool_type_t1 VALUES (123, 'sic est');

-- View data.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 1       | sic est
 0       | non est
 123     | sic est
(3 rows)

m_db=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
WARNING:  Truncated incorrect double value: 't'
 bt_col1 | bt_col2
---------+---------
 0       | non est

m_db=# SELECT * FROM bool_type_t1 WHERE bt_col1 = '123';
 bt_col1 | bt_col2
---------+---------
 123     | sic est
(1 row)

-- Drop the table.
m_db=# DROP TABLE bool_type_t1;
```

4.6.3 Binary Types

Table 4-74 lists the M-compatible binary types.

Table 4-74 Binary types

Name	Description	Storage Space
BINARY[(n)]	Fixed-length binary string. Empty characters are filled in with blank spaces. **n** indicates the byte length. If it is not specified, the default precision **1** is used.	Up to 255 bytes
VARBINARY(n)	Variable-length binary string. **n** indicates the byte length.	Up to 65532 bytes
BLOB[(n)]	Binary large object (BLOB). **n** indicates the optional byte length of the type. After **n** is specified, a column is created as the BLOB type that has the minimum length but is sufficient to hold **n** bytes.	Up to 65535 bytes
TINYBLOB	Binary large object (BLOB).	Up to 255 bytes
MEDIUMBLOB	Binary large object (BLOB).	Up to 16777215 bytes
LONGBLOB	Binary large object (BLOB).	Up to 1 GB minus 1 byte

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If the value of **sql_mode** contains **strict_trans_tables**, an error is reported if the input value is invalid or exceeds the range.
- If the value of **sql_mode** does not contain **strict_trans_tables** and the input value is invalid or exceeds the range, the truncated value is returned.
- The '||' connector is not supported.

Examples

Binary types are displayed using strings.

```
-- Create a table.
m_db=# CREATE TABLE blob_type_t1
(
    BT_COL1 BINARY(10),
    BT_COL2 VARBINARY(10),
    BT_COL3 BLOB,
    BT_COL4 BLOB(10)
) DISTRIBUTE BY REPLICATION;

-- Insert data.
m_db=# INSERT INTO blob_type_t1 VALUES ('blob_test1', 'blob_test2','blob_test3','blob_test4');

-- Query data in the table.
m_db=# SELECT *  FROM blob_type_t1;
  bt_col1   |  bt_col2   |  bt_col3   |  bt_col4
------------+------------+------------+------------
 blob_test1 | blob_test2 | blob_test3 | blob_test4
(1 row)

-- Drop the table.
m_db=# DROP TABLE blob_type_t1;

-- Example: BINARY type
-- Create a table.
m_db=# CREATE TABLE blob_type_t2
(
    BT_COL1 BINARY(5)
) DISTRIBUTE BY REPLICATION;

-- With the strict mode enabled, an error is reported if the length of an inserted string exceeds the length specified for the type.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO blob_type_t2 VALUES ('too long');
ERROR:  value too long for type binary(5)
CONTEXT:  referenced column: bt_col1

-- With the strict mode disabled, the value is truncated if the length of an inserted string exceeds the length specified for the type.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO blob_type_t2 VALUES ('too long');
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM blob_type_t2;
 bt_col1
---------
 too l
(1 row)

-- Drop the table.
m_db=# DROP TABLE blob_type_t2;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.6.4 Character Types

Table 4-75 lists the character types supported by M-compatible databases.

Table 4-75 Character Types

Name	Description	Storage Space
CHAR[(n)]	Fixed-length character string. Empty characters are filled in with blank spaces. **n** indicates the string length. If it is not specified, the default precision **1** is used.	Up to 255 characters
VARCHAR(n)	Variable-length character string. **n** indicates the string length.	Up to 65532 bytes
TEXT[(n)]	Variable-length character string. **n** indicates the optional string length of the type. After **n** is specified, a column is created as the TEXT type that has the minimum length but is sufficient to hold the byte length corresponding to **n** characters.	Up to 65535 bytes
TINYTEXT	Variable-length character string.	Up to 255 bytes
MEDIUMTEXT	Variable-length character string.	Up to 16777215 bytes
LONGTEXT	Variable-length character string.	Up to 1 GB minus 1 byte

NOTE

- If the value of **sql_mode** contains **strict_trans_tables** and the input value is invalid or exceeds the storage space range, an ERROR message is reported.
- If the value of **sql_mode** does not contain **strict_trans_tables** and the input value is invalid or exceeds the storage space range, the truncated value is returned.
- If the value of **sql_mode** contains **pad_char_to_full_length** and table columns and temporary variables are of the CHAR type, the character string with trailing spaces is output. If the value of **sql_mode** does not contain **pad_char_to_full_length** and table columns and temporary variables are of the CHAR type, the character string with no trailing spaces is output.
- The '||' connector is not supported.

Examples

Character types are displayed using character strings.

```
-- Create a table with data of the CHAR type.
m_db=# CREATE TABLE text_type_t1
(
    BT_COL1 CHAR(10)
);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Insert data.
m_db=# INSERT INTO text_type_t1 VALUES ('text_test1');

-- Query data in the table.
m_db=# SELECT *  FROM text_type_t1;
  BT_COL1
------------
 text_test1
(1 row)

-- Drop the table.
m_db=# DROP TABLE text_type_t1;

-- Create a table with data of the VARCHAR type.
m_db=# CREATE TABLE text_type_t2
(
    BT_COL2 VARCHAR(10)
);

-- Insert data.
m_db=# INSERT INTO text_type_t2 VALUES ('text_test2');

-- Query data in the table.
m_db=# SELECT *  FROM text_type_t2;
  BT_COL2
------------
 text_test2
(1 row)

-- Drop the table.
m_db=# DROP TABLE text_type_t2;

-- Create a table with data of the TEXT type.
m_db=# CREATE TABLE text_type_t3
(
    BT_COL3 TEXT
);

-- Insert data.
m_db=# INSERT INTO text_type_t3 VALUES ('text_test3');

-- Query data in the table.
m_db=# SELECT *  FROM text_type_t3;
  BT_COL3
------------
 text_test3
(1 row)

-- Drop the table.
m_db=# DROP TABLE text_type_t3;

-- Example: CHAR type.
-- Create a table.
m_db=# CREATE TABLE char_type_t1
(
    BT_COL1 CHAR(5)
);

-- With the strict mode enabled, an error is reported if the length of an inserted string exceeds the length
specified for the type.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO char_type_t1 VALUES ('too long');
ERROR:  value too long for type character(5)
CONTEXT:  referenced column: BT_COL1

-- With the strict mode disabled, a warning message is reported and the value is truncated if the length of
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
an inserted string exceeds the length specified for the type.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO char_type_t1 VALUES ('too long');
WARNING:  value too long for type character(5)
CONTEXT:  referenced column: BT_COL1
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM char_type_t1;
m_db=# SELECT * FROM char_type_t1;
 BT_COL1
---------
 too l
(1 row)

-- Drop the table.
m_db=# DROP TABLE char_type_t1;
```

4.6.5 Date and Time Types

Table 4-76 lists the date and time types that can be used in M-compatible databases.

Table 4-76 Date and Time Types

Name	Description	Storage Space
DATE	Used to store year, month, and day information, that is, date information. • Input format: Strings 'YYYY-MM-DD' with separators and 'YYYYMMDD' without separators as well as numbers YYYYMMDD are supported. A year can be specified with only two digits. In this case, 70–99 corresponds to 1970–1999 and 00–69 corresponds to 2000–2069 by default. – Any punctuation character can be used as the separator. • Output format: Only YYYY-MM-DD is supported. • Value range: 1000-01-01 to 9999-12-31. – If the value of **sql_mode** contains **allow_invalid_dates**, the month in the range from 1 to 12 and the day in the range from 1 to 31 can be arbitrarily combined. That is, 02-31 can also be stored. – If the value of **sql_mode** does not contain **no_zero_in_date**, the month and day can be 0. That is, 1999-00-00 is legal. – If the value of **sql_mode** does not contain **no_zero_date**, the year, month and day can be 0. That is, 0000-00-00 is legal.	4 bytes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space
TIME[(p)]	Used to store hour, minute, second, and millisecond information, that is, time information. ● Input format: Strings 'hh:mm:ss' with separators and 'hhmmss' without separators as well as numbers hhmmss are supported. Strings 'D hh:mm:ss' containing the day information is supported. Milliseconds are supported. The input value is rounded off to the specified precision. – Only a single colon (:) can be used as the separator. ● Output format: hh:mm:ss or hhh:mm:ss is supported. ● Value range: –838:59:59.000000 to 838:59:59.000000. ● Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If **p** is set to a value greater than 6, the value **6** is used.	8 bytes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space
DATETIME[(p)]	Used to store year, month, and day information, that is, date information. ● Input format: Strings 'YYYY-MM-DD' with separators and 'YYYYMMDD' without separators as well as numbers YYYYMMDD are supported. A year can be specified with only two digits. In this case, 70–99 corresponds to 1970–1999 and 00–69 corresponds to 2000–2069 by default. Milliseconds are supported. The input value is rounded off to the specified precision. – Any punctuation character can be used as the separator in the date or time part while only the decimal point can be used as the separator before milliseconds. – Spaces or a single uppercase letter T is allowed between the date part (year, month, and day) and the time part (hour, minute, and second). Spaces apply only to strings with separators. ● Output format: Only YYYY-MM-DD hh:mm:ss is supported. ● Value range: 1000-01-01 00:00:00 to 9999-12-31 23:59:59. – If the value of **sql_mode** contains **allow_invalid_dates**, the month in the range from 1 to 12 and the day in the range from 1 to 31 can be arbitrarily combined. That is, 02-31 can also be stored. – If the value of **sql_mode** does not contain **no_zero_in_date**, the month and day can be 0. That is, 1999-00-00 10:00:00 is legal. – If the value of **sql_mode** does not contain **no_zero_date**, the year, month and day can be 0. That is, 0000-00-00 10:00:00 is legal. ● Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If **p** is set to a value greater than 6, the value **6** is used.	8 bytes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space
TIMESTAMP[(p)]	Used to store data that contains both date and time information. Values are affected by the database time zone. ● Input format: Strings 'YYYY-MM-DD hh:mm:ss' with separators and 'YYYYMMDDhhmmss' without separators as well as numbers YYYYMMDDhhmmss are supported. A year can be specified with only two digits. In this case, 70–99 corresponds to 1970–1999 and 00–69 corresponds to 2000–2069 by default. The input value is rounded off to the specified precision. – Any punctuation character can be used as the separator in the date or time part while only the decimal point can be used as the separator before milliseconds. – Spaces or a single uppercase letter T is allowed between the date part (year, month, and day) and the time part (hour, minute, and second). Spaces apply only to strings with separators. ● Output format: Only YYYY-MM-DD hh:mm:ss is supported. ● Value range: 1970-01-01 00:00:01' UTC to 2038-01-19 03:14:07' UTC. ● Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If **p** is set to a value greater than 6, the value **6** is used.	8 bytes
YEAR	Used to store year information. ● Input format: YYYY (four digits) and YY (two digits at most) are supported. For a two-digit year, 70–99 corresponds to 1970–1999 and 00–69 corresponds to 2000–2069 by default. ● Output format: Only YYYY is supported. ● Value range: 1901 to 2155.	4 bytes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- M compatibility does not support the following ODBC syntax literals:

  { d 'str' }

  { t 'str' }

  { ts 'str' }
- M compatibility supports the following standard SQL literals:

  DATE'str'

  TIME 'str'

  TIMESTAMP'str'
- The data of the time type automatically ignores all zeros at the end of the data when it is displayed.
- In strings with separators, the backslash (\) is regarded as a common punctuation character instead of an escape character in the M-compatible database.
- The default value of **p** is **0**.
- If the value of **sql_mode** contains **strict_trans_tables** and the input value is invalid or out of the value range, an error is reported. If **strict_trans_tables** is not contained and the input value is invalid or out of the value range, **0** is stored.

Example:

```
-- Create a table.
m_db=# CREATE TABLE temporal_date(a DATE);
m_db=# CREATE TABLE temporal_time(b TIME);
m_db=# CREATE TABLE temporal_datetime(c DATETIME(5));
m_db=# CREATE TABLE temporal_timestamp(d TIMESTAMP);
m_db=# CREATE TABLE temporal_year(e YEAR);
-- Insert data.
m_db=# INSERT INTO temporal_date VALUES ('2020-02-02');
m_db=# INSERT INTO temporal_date VALUES (date'2020-02-02');
m_db=# INSERT INTO temporal_date VALUES (20200202);
m_db=# INSERT INTO temporal_time VALUES ('20 10:00:00');
m_db=# INSERT INTO temporal_time VALUES ('800:00:00');
m_db=# INSERT INTO temporal_time VALUES (time'200:00:00');
m_db=# INSERT INTO temporal_datetime VALUES ('2020-02-02T04:04:04');
m_db=# INSERT INTO temporal_datetime VALUES (timestamp'2020-02-02 10:00:00');
m_db=# INSERT INTO temporal_datetime VALUES (20201010010101);
m_db=# INSERT INTO temporal_timestamp VALUES ('2020-02-02 10:00:00');
m_db=# INSERT INTO temporal_timestamp VALUES (20200220101010);
m_db=# INSERT INTO temporal_year VALUES (2020);
m_db=# INSERT INTO temporal_year VALUES (20);
-- View data.
m_db=# SELECT * FROM temporal_date;
     a
------------
 2020-02-02
 2020-02-02
 2020-02-02
(3 rows)

m_db=# SELECT * FROM temporal_time;
     b
-----------
 490:00:00
 800:00:00
 200:00:00
(3 rows)

m_db=# SELECT * FROM temporal_datetime;
            c
---------------------------
 2020-02-02 04:04:04.00000
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
 2020-02-02 10:00:00.00000
 2020-10-10 01:01:01.00000
(3 rows)

m_db=# SELECT * FROM temporal_timestamp;
          d
---------------------
 2020-02-02 10:00:00
 2020-02-20 10:10:10
(2 rows)

m_db=# SELECT * FROM temporal_year;
  e
------
 2020
 2020
(2 rows)

-- Drop the table.
m_db=# DROP TABLE temporal_date;
m_db=# DROP TABLE temporal_time;
m_db=# DROP TABLE temporal_datetime;
m_db=# DROP TABLE temporal_timestamp;
m_db=# DROP TABLE temporal_year;
```

4.6.6 Bit String Types

Bit strings are strings consisting of 1 and 0. They can be used to store bit masks. **Table 4-77** lists the bit string types supported by M-compatible databases.

Table 4-77 Bit String Types

Name	Description	Storage Space
BIT[(n)]	Bit strings are strings consisting of 1 and 0. They can be used to store bit masks. **n** indicates the bit string length. The value range is [1,64]. If it is not specified, the default precision **1** is used.	Up to 64 bytes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If one explicitly casts a bit-string value to **BIT(n)**, it will be truncated or zero-padded on the right to be exactly *n* bits, without raising an error.
- If the bit string value is output, the bit string value of the original data is output as a character string. For details, see **Example**.
- The BIT type supports bit string input and common character string input. For details, see **Example**.
- If the value of **sql_mode** contains **strict_trans_tables** and the input value is invalid or out of the value range, an error is reported. If **strict_trans_tables** is not contained and the input value is invalid or out of the value range, the maximum value within the specified length range is stored.
- When setBytes or setBinaryStream is used to insert data into a BIT column in the JDBC&&PBE scenario, if the byte stream is not 4-byte-aligned, the data fails to be inserted and an error is reported because the data is incomplete in the M-compatible database. Replace the setBytes method with the setString method.

Example

Bit string types are displayed using character strings.

```
-- Create a table.
m_db=# CREATE TABLE bit_type_t1
(
    BIT_COL BIT(64)
) DISTRIBUTE BY REPLICATION;

-- Insert data. The operation is successful in centralized mode, and an error is reported in distributed mode.
m_db=# INSERT INTO bit_type_t1 VALUES (b'101');

-- Query data in the table.
m_db=# SELECT *  FROM bit_type_t1;
 bit_col
---------
 101
(1 row)

-- Drop the table.
m_db=# DROP TABLE bit_type_t1;

-- Create a table.
m_db=# CREATE TABLE bit_type_t2
(
    BIT_COL BIT(3)
) DISTRIBUTE BY REPLICATION;

-- With the strict mode enabled, an error is reported if the length of an inserted string exceeds the length specified for the type.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO bit_type_t2 VALUES (b'10101');
ERROR:  bit string too long for type bit varying(3)
CONTEXT:  referenced column: bit_col

-- With the strict mode disabled, the maximum value within the specified length range is inserted if an inserted string exceeds the length specified for the type.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO bit_type_t2 VALUES (b'10101');
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM bit_type_t2;
 bit_col
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
---------
 111
(1 row)

-- Drop the table.
m_db=# DROP TABLE bit_type_t2;
```

4.6.7 Numeric Types

Numeric types in M-compatible databases include signed integer types, unsigned integer types, fixed-point types, and floating-point types.

4.6.7.1 Signed Integer Types

Table 4-78 lists the signed integer types supported by M-compatible databases.

Table 4-78 Signed Integer Types

Name	Description	Storage Space	Range
TINYINT [SIGNED] [ZEROFILL]	Tiny integer. The type alias is INT1.	1 byte	[-128, 127]
SMALLINT [SIGNED] [ZEROFILL]	Small integer. The type alias is INT2.	2 bytes	[-32768, 32767]
MEDIUMINT [SIGNED] [ZEROFILL]	Medium-sized integer.	4 bytes	[-8388608, 8388607]
INT/INTEGER [SIGNED] [ZEROFILL]	Common integer. The type alias is INT4.	4 bytes	[-2147483648, 2147483647]
BIGINT [SIGNED] [ZEROFILL]	Large integer. The type alias is INT8.	8 bytes	[-2^63, 2^63-1]

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- If the value of **sql_mode** contains **strict_trans_tables** and non-numeric strings are input in the signed integer column, an ERROR message is reported.
- If the value of **sql_mode** does not contain **strict_trans_tables** and non-numeric strings are input in the signed integer column, the truncated value is returned.
- To create a table column, adding SIGNED syntax after the integer type is equivalent to directly using the signed integer type.
- The ZEROFILL attribute can be set for the signed integer type.
  - When the ZEROFILL attribute is set, zeros are added before the value based on the display width. If the display width is not set, the default display width of the data type is used. For example, for a column declared as **INT(4) ZEROFILL**, a value of **5** is retrieved as 0005.
  - When the ZEROFILL attribute is set, the UNSIGNED attribute is automatically added. Even if the ZEROFILL attribute is set to SIGNED, the UNSIGNED attribute is converted to the UNSIGNED attribute.

Examples

```
-- Create a table whose column is of signed integer type.
m_db=# CREATE TABLE test_int
(
    a TINYINT,
    b SMALLINT,
    c MEDIUMINT,
    d INTEGER,
    e BIGINT
);

-- Insert data.
m_db=# INSERT INTO test_int VALUES (100, 200, 300, 400, 500);

-- Query data in the table.
m_db=# SELECT * FROM test_int;
  a  |  b  |  c  |  d  |  e
-----+-----+-----+-----+-----
 100 | 200 | 300 | 400 | 500
(1 row)

-- With the strict mode enabled, an error is reported if a non-numeric string is inserted.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '200$30', '1@3');
ERROR:  invalid input syntax for integer: "1@2@3"
LINE 1: INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '20...
                                     ^
CONTEXT:  referenced column: a

-- With the strict mode disabled, a warning message is reported if a non-numeric string is inserted, and the
string is automatically truncated.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '200$30', '1@3');
WARNING:  invalid input syntax for integer: "1@2@3"
LINE 1: INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '20...
                                     ^
CONTEXT:  referenced column: a
WARNING:  invalid input syntax for integer: "200&20&2"
LINE 1: INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '20...
                                           ^
CONTEXT:  referenced column: b
WARNING:  invalid input syntax for integer: "4%4"
LINE 1: INSERT INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '20...
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
                                                ^
CONTEXT:  referenced column: c
WARNING:  invalid input syntax for integer: "200$30"
LINE 1: ...INTO test_int VALUES ('1@2@3', '200&20&2', '4%4', '200$30', ...
                                                             ^
CONTEXT:  referenced column: d
WARNING:  invalid input syntax for integer: "1@3"
LINE 1: ...st_int VALUES ('1@2@3', '200&20&2', '4%4', '200$30', '1@3');
                                                               ^
CONTEXT:  referenced column: e
INSERT 0 1

m_db=# SELECT * FROM test_int;
  a  |  b  |  c  |  d  |  e
-----+-----+-----+-----+-----
 100 | 200 | 300 | 400 | 500
 1   | 200 | 4   | 200 |   1
(2 rows)

-- Set the zerofill attribute.
m_db=# dROP TABLE IF EXISTS t1;
DROP TABLE
m_db=# CREATE TABLE t1 (a int zerofill);
CREATE TABLE
m_db=# INSERT INTO t1 VALUES(1);
INSERT 0 1
m_db=# SELECT * FROM t1;
     a
------------
 0000000001
(1 row)

-- Drop the table.
m_db=# DROP TABLE test_int;
```

4.6.7.2 Unsigned Integer Types

Table 4-79 lists the unsigned integer types supported by M-compatible databases.

Table 4-79 Unsigned Integer Types

Name	Description	Storage Space	Range
TINYINT UNSIGNED [ZEROFILL]	Unsigned tiny integer.	1 byte	[0, 255]
SMALLINT UNSIGNED [ZEROFILL]	Unsigned small integer.	2 bytes	[0, 65,535]
MEDIUMINT UNSIGNED [ZEROFILL]	Unsigned medium-sized integer.	4 bytes	[0, 16,777,215]
INT/INTEGER UNSIGNED [ZEROFILL]	Unsigned common integer.	4 bytes	[0, 4,294,967,295]

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space	Range
BIGINT UNSIGNED [ZEROFILL]	Unsigned large integer.	8 bytes	[0, 18,446,744,073,709,551,615]

NOTE

- If the value of **sql_mode** contains **strict_trans_tables** and a negative value is input in the unsigned integer column, an ERROR message is reported.
- If the value of **sql_mode** contains **strict_trans_tables** and non-numeric strings are input in the unsigned integer column, an ERROR message is reported.
- If the value of **sql_mode** does not contain **strict_trans_tables** and a negative value is input in the unsigned integer column, a WARNING message is reported and **0** is returned.
- If the value of **sql_mode** does not contain **strict_trans_tables** and non-numeric strings are input in the unsigned integer column, the truncated value is returned.
- The ZEROFILL attribute can be set for the unsigned integer type.
  - When the ZEROFILL attribute is set, zeros are added before the value based on the display width. If the display width is not set, the default display width of the data type is used. For example, for a column declared as **INT(4) UNSIGNED ZEROFILL**, a value of **5** is retrieved as 0005.
  - When the ZEROFILL attribute is set, the UNSIGNED attribute is automatically added.

Examples

```
-- Create a table whose column is of unsigned integer type.
m_db=# CREATE TABLE test_uint
(
    a TINYINT UNSIGNED,
    b SMALLINT UNSIGNED,
    c MEDIUMINT UNSIGNED,
    d INTEGER UNSIGNED,
    e BIGINT UNSIGNED
);

-- Insert data.
m_db=# INSERT INTO test_uint VALUES (100, 200, 300, 400, 500);

-- Query data in the table.
m_db=# SELECT * FROM test_uint;
  a  |  b  |  c  |  d  |  e
-----+-----+-----+-----+-----
 100 | 200 | 300 | 400 | 500
(1 row)

-- With the strict mode enabled, an error is reported if a negative value is inserted.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO test_uint VALUES (-1, -1, -1, -1, -1);
ERROR:  unsigned tinyint out of range
CONTEXT:  referenced column: a

-- With the strict mode disabled, a warning message is reported and 0 is returned if a negative value is
inserted.
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO test_uint VALUES (-1, -1, -1, -1, -1);
WARNING:  unsigned tinyint out of range
CONTEXT:  referenced column: a
INSERT 0 1
m_db=# select * from test_uint;
 a | b | c | d | e
---+---+---+---+---
 0 | 0 | 0 | 0 | 0
(1 row)

-- Set the zerofill attribute.
m_db=# DROP TABLE IF EXISTS t1;
DROP TABLE
m_db=# CREATE TABLE t1 (a int UNSIGNED zerofill);
CREATE TABLE
m_db=# INSERT INTO t1 VALUES(1);
INSERT 0 1
m_db=# SELECT * FROM t1;
     a
------------
 0000000001
(1 row)


-- Drop the table.
m_db=# DROP TABLE test_uint;
```

4.6.7.3 Fixed-Point Types

The fixed-point types are used to store exact numeric values. **Table 4-80** lists the fixed-point types supported by M-compatible databases.

Table 4-80 Fixed-Point Types

Name	Description	Storage Space	Value Range (Decimal Digits)
NUMERIC[(p[,s])] [ZEROFILL]	• The value range of the precision **p** is [1,65], and the value range of the scale **s** is [0,30] (s ≤ p). • If the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 81 digits

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space	Value Range (Decimal Digits)
DECIMAL[(p[,s])] [ZEROFILL]	This type maps to NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 81 digits
DEC[(p[,s])] [ZEROFILL]	This type maps to NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 81 digits
FIXED[(p[,s])] [ZEROFILL]	This type maps to NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 81 digits

NOTE

- The precision **p** indicates the maximum number of digits, and the scale **s** indicates the number of decimal digits.
- If the input value exceeds the range implied by the scale, a NOTICE message is reported and the truncated value is returned.
- If the value of **sql_mode** contains **strict_trans_tables**, an ERROR message is reported if the input value is invalid or exceeds the range implied by the precision.
- If the value of **sql_mode** does not contain **strict_trans_tables** and the input value is invalid or exceeds the range implied by the precision, the truncated value or the maximum value within the range implied by the precision is returned.
- The ZEROFILL attribute of the fixed-point number type can be set.
  - When the ZEROFILL attribute is set, zeros are added to the integer and decimal places of a value based on the precision and scale. If the precision and scale are not set, the default precision and scale of the data type are used as the specific precision and scale. For example, for a column declared as **NUMERIC(10,5) ZEROFILL**, a value of **2.1** is retrieved as 00002.10000.
  - When the ZEROFILL attribute is set, the UNSIGNED attribute is automatically added.

Examples

```
-- Create a table whose columns are of fixed-point type.
m_db=# CREATE TABLE num_tab1
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(
    a NUMERIC(10,5),
    b DECIMAL(10,5),
    c DEC(10,5),
    d FIXED(10,5)
);

-- Insert data within the range implied by the specified precision and scale.
m_db=# INSERT INTO num_tab1 VALUES(1234.56, 1234.56, 1234.56, 1234.56);

-- If the inserted data exceeds the range implied by the specified scale, a notice message is reported, and the value truncated based on the specified scale is returned.
m_db=# INSERT INTO num_tab1 VALUES(12.3456789, 12.3456789, 12.3456789, 12.3456789);
NOTICE:  Truncated incorrect DECIMAL value: '12.3456789'
CONTEXT:  referenced column: a
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM num_tab1;
     a      |     b      |     c      |     d
------------+------------+------------+------------
   12.34568 |   12.34568 |   12.34568 |   12.34568
 1234.56000 | 1234.56000 | 1234.56000 | 1234.56000
(2 rows)

-- With the strict mode enabled, an error is reported if the inserted data exceeds the range implied by the specified precision.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO num_tab1 VALUES(12345678999, 12345678999, 12345678999, 12345678999);
ERROR:  numeric field overflow
DETAIL:  A field with precision 10, scale 5 must round to an absolute value less than 10^5.
CONTEXT:  referenced column: a

-- With the strict mode disabled, if the inserted data exceeds the range implied by the specified precision, a warning message is reported and the maximum value within the range implied by the specified precision is returned.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO num_tab1 VALUES(12345678999, 12345678999, 12345678999, 12345678999);
WARNING:  numeric field overflow
DETAIL:  A field with precision 10, scale 5 must round to an absolute value less than 10^5.
CONTEXT:  referenced column: a
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM num_tab1;
      a       |      b      |      c      |      d
--------------+-------------+-------------+-------------
  1234.56000 |  1234.56000 |  1234.56000 |  1234.56000
    12.34568 |    12.34568 |    12.34568 |    12.34568
  99999.99999 | 99999.99999 | 99999.99999 | 99999.99999
(3 rows)

-- Set the zerofill attribute.
m_db=# DROP TABLE IF EXISTS t1;
DROP TABLE
m_db=# CREATE TABLE t1 (a numeric zerofill);
CREATE TABLE
m_db=# INSERT INTO t1 VALUES(1);
INSERT 0 1
m_db=# SELECT * FROM t1;
     a
------------
 0000000001
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- Drop the table.
m_db=# DROP TABLE num_tab2;
```

4.6.7.4 Floating-Point Types

The floating-point types are used to store approximate numeric values. **Table 4-81** lists the floating-point types supported by M-compatible databases.

Table 4-81 Floating-Point Types

Name	Description	Storage Space	Range
FLOAT4[(p, s)] [ZEROFILL]	Single-precision floating-point number, which has errors. The value range of the precision **p** is [1,255], and the value range of the scale **s** is [0,min(*p*, 30)].	4 bytes	–3.402E+38 to 3.402E+38, 6-bit decimal digits
FLOAT8[(p,s)] [ZEROFILL]	Double-precision floating-point number, which has errors. The value range of the precision **p** is [1,255], and the value range of the scale **s** is [0,min(*p*, 30)].	8 bytes	–1.79E+308 to 1.79E+308, 15-bit decimal digits
FLOAT[(p)] [ZEROFILL]	The value range of **p** is [0,53]. • When **p** is set to [0, 25), FLOAT(p) is the same as FLOAT4. • When **p** is set to [25, 54), FLOAT(p) is the same as FLOAT8.	4 bytes when **p** is set to [0, 25); 8 bytes when **p** is set to [25, 54)	When **p** is set to [0, 25), the value range is the same as that of FLOAT4. When **p** is set to [25, 54), the value range is the same as that of FLOAT8.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Description	Storage Space	Range
REAL[(p, s)] [ZEROFILL]	The behavior is the same as that of DOUBLE[(p, s)]. Data is stored by DOUBLE[(p, s)]. When **sql_mode** is set to **REAL_AS_FLOAT**, data is stored by FLOAT[(p, s)]. The behavior is the same as that of FLOAT8[(p, s)].	8 bytes 4 bytes when **sql_mode** is set to **REAL_AS_FLOAT**	–1.79E+308 to 1.79E+308, 15-bit decimal digits –3.402E+38 to 3.402E+38, 6-bit decimal digits when **sql_mode** is set to **REAL_AS_FLOAT**
FLOAT[(p, s)] [ZEROFILL]	This type maps to FLOAT4[(p,s)]. The value range of the precision **p** is [1,255], and the value range of the scale **s** is [0,min(*p*, 30)].	4 bytes	–3.402E+38 to 3.402E+38, 6-bit decimal digits
DOUBLE[(p,s)] [ZEROFILL]	This type maps to FLOAT8[(p,s)]. The value range of the precision **p** is [1,255], and the value range of the scale **s** is [0,min(*p*, 30)].	8 bytes	–1.79E+308 to 1.79E+308, 15-bit decimal digits
DOUBLE PRECISION[(p,s)] [ZEROFILL]	This type maps to FLOAT8[(p,s)]. The value range of the precision **p** is [1,255], and the value range of the scale **s** is [0,min(*p*, 30)].	8 bytes	–1.79E+308 to 1.79E+308, 15-bit decimal digits

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

NOTE

- The precision **p** indicates the maximum number of digits, and the scale **s** indicates the number of decimal digits.
- If the input value exceeds the scale, the truncated value is returned.
- If the precision is not set, the column range is the same as that specified in **Table 4-81**. If the precision is set, the column range is subject to the configured precision.
- If the value of **sql_mode** contains **strict_trans_tables**, an ERROR message is reported if the input value is invalid or exceeds the range implied by the precision.
- If the value of **sql_mode** does not contain **strict_trans_tables** and the input value is invalid or exceeds the range implied by the precision, the truncated value or the maximum value within the range implied by the precision is returned.
- If the value of **sql_mode** contains **real_as_float**, columns of the REAL type are stored as the FLOAT type, and the behavior is the same as that of the FLOAT type.
- If the value of **sql_mode** does not contain **real_as_float**, columns of the REAL type are stored as the DOUBLE type, and the behavior is the same as that of the DOUBLE type.

Examples

```
-- Create a table whose columns are of floating-point type.
m_db=# CREATE TABLE float_tab1
(
    a FLOAT(6,2),
    b DOUBLE(6,2)
) DISTRIBUTE BY REPLICATION;

-- Insert data within the range implied by the specified precision and scale.
m_db=# INSERT INTO float_tab1 VALUES(1234.56, 1234.56);

-- If the inserted data exceeds the range implied by the specified scale, the value truncated based on the specified scale is returned.
m_db=# INSERT INTO float_tab1 VALUES(12.3456789, 12.3456789);

-- With the strict mode enabled, an error is reported if the inserted data exceeds the range implied by the specified precision.
m_db=# SET SQL_MODE =
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,
no_auto_create_user,no_engine_substitution';
SET
m_db=# INSERT INTO float_tab1 VALUES(12345678999, 12345678999);
ERROR:  float field overflow
DETAIL:  A field with precision 6, scale 2 must round to an absolute value less than 9.9999900e+03.
CONTEXT:  referenced column: a


-- With the strict mode disabled, if the inserted data exceeds the range implied by the specified precision, the maximum value within the range implied by the specified precision is returned.
m_db=# SET SQL_MODE = '';
SET
m_db=# INSERT INTO float_tab1 VALUES(12345678999, 12345678999);
INSERT 0 1

-- Query data in the table.
m_db=# SELECT * FROM float_tab1;
    a    |    b
---------+---------
 1234.56 | 1234.56
   12.35 |   12.35
 9999.99 | 9999.99
(3 rows)

-- Drop the table.
m_db=# DROP TABLE float_tab1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
-- When SQL_MODE does not contain REAL_AS_FLOAT, create a table whose columns are of REAL type.
m_db=# SET SQL_MODE='';
SET
m_db=# CREATE TABLE real_tab1(a REAL) DISTRIBUTE BY REPLICATION;

-- Insert data.
m_db=# INSERT INTO real_tab1 VALUES(12345.678999999999);

-- Query data in the table. The result is the same as that of the DOUBLE type and is accurate to 12 digits.
m_db=# SELECT * FROM real_tab1;
    a
---------
 12345.678999999998
 (1 row)

-- Drop the table.
m_db=# DROP TABLE real_tab1;

-- When SQL_MODE contains REAL_AS_FLOAT, create a table whose columns are of REAL type.
m_db=# SET SQL_MODE='REAL_AS_FLOAT';
SET
m_db=# CREATE TABLE real_tab2(a REAL) DISTRIBUTE BY REPLICATION;

-- Insert data.
m_db=# INSERT INTO real_tab2 VALUES(12345.678999999999);

-- Query data in the table. The result is the same as that of the FLOAT type and is accurate to six digits.
m_db=# SELECT * FROM real_tab2;
    a
---------
 12345.7
 (1 row)

-- Set the zerofill attribute.
m_db=# DROP TABLE IF EXISTS t1;
DROP TABLE
m_db=# CREATE TABLE t1 (a float zerofill) DISTRIBUTE BY REPLICATION;
CREATE TABLE
m_db=# INSERT INTO t1 VALUES(1);
INSERT 0 1
m_db=# SELECT * FROM t1;
      a
--------------
 000000000001
(1 row)

-- Drop the table.
m_db=# DROP TABLE real_tab2;
```

4.7 Type Conversion

Context

In SQL, every data item has an associated data type which determines its behavior and allowed usage. There are four fundamental SQL structures requiring distinct type conversion rules in M-compatible databases:

- Function calls

  Much of the SQL type system is built around a rich set of functions. Functions can have one or more arguments. Since SQL permits function overloading, the function name alone does not uniquely identify the function to be called. The parser must select the right function based on the data types of the supplied arguments.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Operators

  SQL allows expressions with prefix and postfix unary (one-argument) operators, as well as binary (two-argument) operators. Like functions, operators can be overloaded, so the same problem of selecting the right operator exists.

- Value storage

  The INSERT and UPDATE statements place the results of expressions into a table. The expressions in the statements must be matched up with, and perhaps converted to, the types of the target columns.

- UNION, CASE, and related constructs

  Since all query results from a unionized SELECT statement must appear in a single set of columns, the types of the results of each SELECT clause must be matched up and converted to a uniform set. Similarly, the result expressions of a CASE construct must be converted to a common type so that the **CASE** expression as a whole has a known output type.

Based on type conversion rules of M-compatible databases, the four SQL structures are classified into three scenarios: implicit type conversion, explicit type conversion, UNION, CASE, and related constructs. These three scenarios have different conversion behaviors and conversion relationships in M-compatible databases.

The return type and conversion behavior of an expression are determined during semantic analysis. In M-compatible databases, the basic data types include INTEGER, DECIMAL, STRING, BITSTRING, FLOAT, and TEMPORAL. To solve the type selection problem of expressions, implicit conversion is available between basic types in M-compatible databases. With the types and available implicit conversions, it is possible to ensure that ambiguous expressions can be resolved in a useful way.

4.7.1 Implicit Type Conversion

Implicit type conversion mainly applies to function calls, operators, and value storage scenarios. This section describes type resolution in the value storage scenario. For details about other scenarios, see **Functions and Operators**.

Value Storage Type Resolution

1. Search for an exact match with the target column.
2. Try to convert the expression to the target type. If a registered cast function exists between the two types, call the cast function directly. If the expression is an unknown-type literal, the content of the literal string will be passed to the input conversion routine for the target type.
3. Check whether there is a sizing cast for the target type. A sizing cast is a cast from that type to itself. If the target type has sizing cast logic, convert it to the expression before storing into the destination column. The cast function receives a parameter of the INTEGER type and the value of **atttypmod** of the target field (the value is the length specified by the function, and the meaning of **atttypmod** varies according to the data types). In addition, the function may receive a Boolean data record as the third parameter, indicates the conversion mode (explicit conversion or implicit conversion). The cast

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

function is responsible for setting length-dependent statements such as sizing check or truncation.

Example:

```
m_db=# CREATE TABLE char_storage (
    VS_COL1 CHAR(20)
);
m_db=# INSERT INTO char_storage VALUES('abcdef');
m_db=# SELECT VS_COL1, length(VS_COL1) FROM char_storage;
 VS_COL1 | length
---------+--------
 abcdef  |      6
(1 row)

m_db=# DROP TABLE char_storage;
```

Convert the FLOAT storage type to the BINARY storage type. Insert the value obtained from the FLOAT column to the table whose target column is **BINARY(5)**.

```
m_db=# CREATE TABLE varchar_storage (
    VS_COL1 BINARY(5)
) DISTRIBUTE BY REPLICATION;
m_db=# CREATE TABLE float_storage (
    VS_COL1 FLOAT
) DISTRIBUTE BY REPLICATION;
m_db=# INSERT INTO float_storage VALUES(1234567);
m_db=# INSERT INTO varchar_storage SELECT * FROM float_storage;
m_db=# DROP TABLE varchar_storage, float_storage;
```

4.7.2 Explicit Type Conversion

Explicit type conversion mainly applies to the scenario where type conversion functions are called. For details about type conversion functions, see **Type Conversion Functions**.

The explicit type conversion syntax is as follows:

```
SELECT CAST(expression AS data_type);
```

expression indicates the expression to be converted, and **data_type** indicates the target data type. **Table 4-82** lists the target types that are supported.

Table 4-82 Explicit cast conversion

Target Type	Data Type of Input Values (Type of Return Parameters)
SIGNED [INTEGER]	BIGINT
UNSIGNED [INTEGER]	BIGINT UNSIGNED
DATE	DATE
TIME	TIME
DATETIME	DATETIME
BINARY	TEXT
CHAR	TEXT
DECIMAL	NUMERIC

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Target Type	Data Type of Input Values (Type of Return Parameters)
FLOAT	FLOAT4
DOUBLE	FLOAT8

Example:

```
-- Convert cast usage of SIGNED.
m_db=# SELECT CAST(9223372036854775808 AS SIGNED);
        cast
----------------------
 -9223372036854775808
(1 row)

-- Boundary value processing.
m_db=# SELECT CAST(9223372036854775808.0 AS SIGNED);
WARNING:  Truncated incorrect DECIMAL value: '9223372036854775808.0'
CONTEXT:  referenced column: cast
        cast
---------------------
 9223372036854775807
(1 row)

-- Convert cast usage of CHAR (modifiers-limited boundaries).
m_db=# SELECT CAST(123 AS CHAR(3));
 cast
------
 123
(1 row)
```

NOTE

Double-colon syntaxes, such as "SELECT 1::INT", can also be used for explicit conversion. The logic of double-colon explicit conversion is the same as the default logic of implicit conversion, and new data types in an M-compatible database support the double-colon syntaxes. You are advised not to use the conversion mode directly because the result may not meet the expectation.

4.7.3 UNION, CASE, and Related Constructs

SQL UNION/CASE constructs must match up possibly dissimilar types to become a single result set. The resolution algorithm is applied separately to each output column of a union query. The EXCEPT constructs resolve different types by using the same algorithm as UNION.

UNION/CASE converts expressions of the two types to a unified target type and converts data to the target type through implicit type conversion to implement aggregation.

Type Resolution for UNION and Related Constructs

UNION is classified into UNION and UNION ALL. UNION aggregates data and deduplicates data. UNION ALL aggregates data but does not deduplicate data.

Example:

```
-- Create a table.
m_db=# CREATE TABLE test_union1 (union_col1 INT);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# CREATE TABLE test_union2 (union_col2 CHAR(5));

-- Insert data.
m_db=# INSERT INTO test_union1 VALUES (123);
m_db=# INSERT INTO test_union2 VALUES ('123');

-- Usage of UNION
m_db=# SELECT union_col1 FROM test_union1 UNION SELECT union_col2 FROM test_union2;
 union_col1
------------
 123
(1 row)

-- Usage of UNION ALL
m_db=# SELECT union_col1 FROM test_union1 UNION ALL SELECT union_col2 FROM test_union2;
 union_col1
------------
 123
 123
(2 rows)

-- UNION application of a non-column value
m_db=# SELECT '234' FROM test_union1 UNION SELECT '234';
 ?column?
----------
 234
(1 row)

-- Drop the table.
m_db=# DROP TABLE test_union1;
m_db=# DROP TABLE test_union2;
```

Type Resolution for CASE and Related Constructs

The CASE syntax has two expression modes, which have the same functions and meanings.

```
CASE WHEN col = A THEN ... ELSE ...
CASE col WHEN A THEN ... ELSE ...
```

Example:

```
-- Create a table.
m_db=# CREATE TABLE test_case1 (case_col1 INT);

-- Insert data.
m_db=# INSERT INTO test_case1 VALUES (1), (23), (45), (1);

-- Usage of case
m_db=# SELECT CASE WHEN case_col1 = 1 THEN 'a' ELSE 'b' END FROM test_case1;
 case
------
 a
 b
 b
 a
(4 rows)

m_db=# SELECT CASE case_col1 WHEN 1 THEN 'a' ELSE 'b' END FROM test_case1;
 case
------
 a
 b
 b
 a
(4 rows)

-- Drop the table.
m_db=# DROP TABLE test_case1;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4.8 Expressions

4.8.1 Simple Expressions

Logical Expressions

Logical Operators lists the operators and calculation rules of logical expressions.

Comparison Expressions

Comparison Functions and Comparison Operators lists the common comparison operators.

In addition to comparison operators, you can also use the following sentence structures:

BETWEEN operator

```
a BETWEEN x AND y
```

Generally, it is equivalent to a ≥ x AND a ≤ y.

```
a NOT BETWEEN x AND y
```

Generally, it is equivalent to a < x OR a > y.

NOTE

- During conversion, the BETWEEN operator first determines the comparison type for a and x, and then determines the final comparison type based on the comparison type for a and x as well as y. Therefore, the result is not equivalent to that of a >= x AND a <= y in complex scenarios.
- In a BETWEEN x AND y, x and y cannot use expressions, for example, 1 < 1.

Example

```
m_db=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

m_db=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

m_db=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

m_db=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT '2' between 'a' AND 3.0 AS RESULT;
WARNING:  Truncated incorrect double value: 'a'
CONTEXT:  referenced column: RESULT
 result
----------
 t
(1 row)

m_db=# SELECT '2' >= 'a' AND '2' <= 3.0 AS RESULT;
 result
----------
 f
(1 row)
```

4.8.2 Condition Expressions

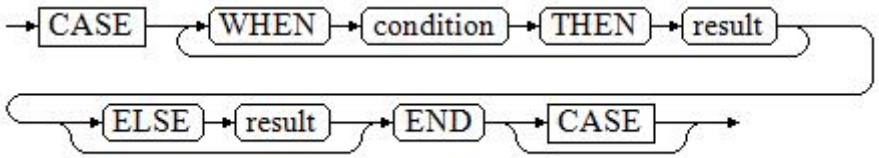
Data that meets the requirements specified by condition expressions is filtered during SQL statement execution.

Condition expressions include the following types:

- CASE

  A CASE expression is a conditional expression. **Figure 4-1** shows the syntax diagram.

  Figure 4-1 case::=

  A CASE clause can be used in a valid expression. **condition** is an expression that returns a value of Boolean type.

  - If the result is true, the result of the CASE expression is the required result.
  - If the result is false, the WHEN or ELSE clauses in the statement are processed in the same way.
  - If every WHEN condition is not true, the result of the expression is the result of the ELSE clause. If the ELSE clause is omitted and no match is found, the result is **NULL**.
  - Operations on XML data are supported.

  Example:

  ```
  m_db=# CREATE SCHEMA tpcds;

  m_db=# CREATE TABLE tpcds.case_when_t1(CW_COL1 INT);

  m_db=# INSERT INTO tpcds.case_when_t1 VALUES (1), (2), (3);

  m_db=# SELECT * FROM tpcds.case_when_t1;
  cw_col1
  ---------
   1
   2
   3
  (3 rows)
  ```

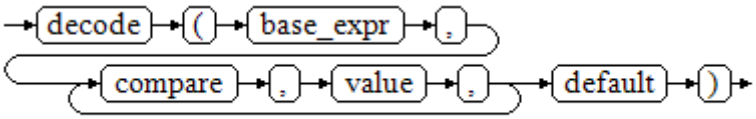
 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two'
ELSE 'other' END FROM tpcds.case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
       2 | two
       3 | other
(3 rows)

m_db=# DROP TABLE tpcds.case_when_t1;

m_db=# DROP SCHEMA tpcds;
```

- DECODE

  Figure 4-2 shows the syntax of DECODE.

  Figure 4-2 decode::=

  Compare each compare(n) with base_expr. **value(n)** is returned if a compare(n) matches the base_expr expression. If they do not match, **default** is returned.

  Operations on XML data are supported.

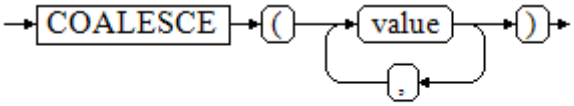
  For details about the information and examples, see "SQL Reference > Functions and Operators > Condition Expression Functions" in *Developer Guide*.

  ```
  m_db=# SELECT DECODE('A','A',1,'B',2,0);
   case
  ------
     1
  (1 row)
  ```

- COALESCE

  Figure 4-3 shows the syntax of COALESCE.

  Figure 4-3 coalesce::=

  COALESCE returns the first NOT NULL value. If all the parameters are **NULL**, **NULL** is returned. It is often used to replace **NULL** with the default value. Like a CASE expression, COALESCE only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-null parameter are not evaluated.

  Operations on XML data are supported.

  Example

  ```
  m_db=# CREATE SCHEMA tpcds;
  m_db=# CREATE TABLE tpcds.c_tabl(description varchar(10), short_description varchar(10),
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
final_value varchar(10));

m_db=# INSERT INTO tpcds.c_tabl VALUES('abc', 'efg', '123');
m_db=# INSERT INTO tpcds.c_tabl VALUES(NULL, 'efg', '123');

m_db=# INSERT INTO tpcds.c_tabl VALUES(NULL, NULL, '123');

m_db=# SELECT description, short_description, final_value, COALESCE(description, short_description,
final_value) FROM tpcds.c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | final_value| coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

m_db=# DROP TABLE tpcds.c_tabl;
m_db=# DROP SCHEMA tpcds;
```

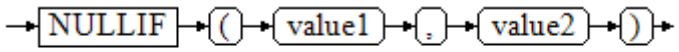
If **description** is not **NULL**, the value of **description** is returned. Otherwise, parameter **short_description** is calculated. If **short_description** is not **NULL**, the value of **short_description** is returned. Otherwise, parameter **final_value** is calculated. If **final_value** is not **NULL**, the value of **final_value** is returned. Otherwise, **none** is returned.

```
m_db=# SELECT COALESCE(NULL,'Hello World');
   coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

  Figure 4-4 shows the syntax of NULLIF.

  Figure 4-4 nullif::=

  Only if the value of **value1** is equal to that of **value2** can NULLIF return **NULL**. Otherwise, the value of **value1** is returned.

  Operations on XML data are supported.

  Example

```
m_db=# CREATE SCHEMA tpcds;
m_db=# CREATE TABLE tpcds.null_if_t1 (
    NI_VALUE1 VARCHAR(10),
    NI_VALUE2 VARCHAR(10)
);

m_db=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'abc');
m_db=# INSERT INTO tpcds.null_if_t1 VALUES('abc', 'efg');

m_db=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM tpcds.null_if_t1
ORDER BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
m_db=# DROP TABLE tpcds.null_if_t1;
m_db=# DROP SCHEMA tpcds;
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

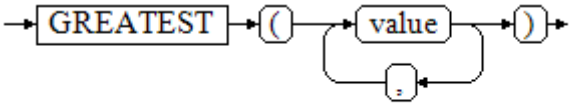
If the value of **value1** is equal to that of **value2**, **NULL** is returned. Otherwise, the value of **value1** is returned.

```
m_db=# SELECT NULLIF('Hello','Hello World');
 nullif
--------
 Hello
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

  Figure 4-5 shows the syntax of GREATEST.

  Figure 4-5 greatest::=

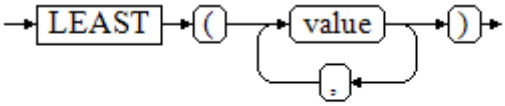
  Select the maximum value from any numerical expression list. Operations on XML data are supported.

```
m_db=# SELECT greatest(9000,155555,2.01);
 greatest
----------
   155555
(1 row)
```

- LEAST (minimum value)

  Figure 4-6 shows the syntax of LEAST.

  Figure 4-6 least::=

  Select the minimum value from any numerical expression list.

  Operations on XML data are supported.

```
m_db=# SELECT least(9000,2);
 least
-------
     2
(1 row)
```

  For details about the information and examples, see "SQL Reference > Functions and Operators > Condition Expression Functions" in *Developer Guide*.

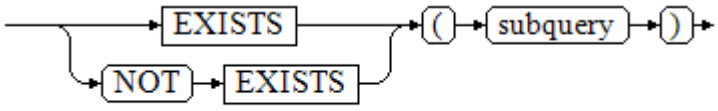
4.8.3 Subquery Expressions

EXISTS/NOT EXISTS

Figure 4-7 shows the syntax of EXISTS/NOT EXISTS.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 4-7 EXISTS/NOT EXISTS::=



- The parameter of an EXISTS expression is an arbitrary SELECT statement, or subquery.
- The subquery is evaluated to determine whether the result returns any rows. If the result returns at least one row, the result of EXISTS is **true**. If the subquery returns no rows, the result of EXISTS is **false**.
- The subquery will generally only be executed long enough to determine whether at least one row is returned, not all the way to completion.

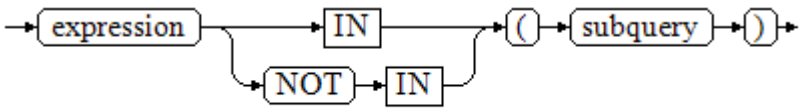
Example:

```
m_db=# CREATE SCHEMA tpcds;
m_db=# CREATE TABLE tpcds.store_returns
(
  sr_reason_sk     integer,
  sr_reason_id      character(16),
  sr_reason_desc    character(100),
  sr_customer_sk    integer
);
m_db=# CREATE TABLE tpcds.date_dim
(
  d_dom            integer,
  d_reason_desc     character(100)
);
m_db=# INSERT INTO tpcds.store_returns VALUES(1, 'test_001', 'test_desc_001', 5), (12, 'test_002',
'test_desc_002', 8), (23, 'test_003', 'test_desc_003', 9), (14, 'test_004', 'test_desc_004', 10), (45, 'test_005',
'test_desc_005', 12), (55, 'test_006', 'test_desc_006', 1), (43, 'test_007', 'test_desc_007', 12);
m_db=# INSERT INTO tpcds.date_dim VALUES(1, 'test_dim_desc_001'), (6, 'test_dim_desc_002'), (9,
'test_dim_desc_003'), (19, 'test_dim_desc_004'), (12, 'test_dim_desc_005'), (23, 'test_dim_desc_006');

m_db=# SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE EXISTS (SELECT d_dom
FROM tpcds.date_dim WHERE d_dom = store_returns.sr_reason_sk AND sr_customer_sk <10);
 sr_reason_sk | sr_customer_sk
--------------+----------------
            1 |              5
           12 |              8
           23 |              9
(3 rows)
```

IN/NOT IN

Figure 4-8 shows the syntax of IN/NOT IN.

Figure 4-8 IN/NOT IN::=



- The right-hand side of IN/NOT IN is a parenthesized subquery, which returns exactly one column.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The expression on the left-hand side of IN/NOT IN is calculated and compared to each row of the subquery result.
- The result of IN is **true** if any equal subquery row is found in the IN/NOT IN statement. The result is **false** if no equal row is found (including the case where the subquery returns no rows).
- This is in accordance with SQL normal rules for Boolean combinations of null values. If the columns corresponding to two rows equal and are not null, the two rows are equal to each other. If any columns corresponding to the two rows do not equal and are not null, the two rows are not equal to each other. Otherwise, the result is **NULL**. If the result of each row does not equal and at least one row yields **NULL**, the result of IN will be **NULL**.

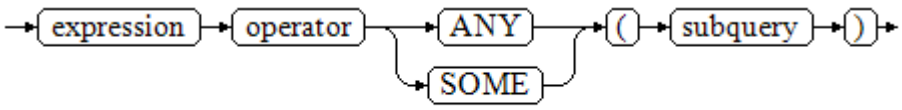
Example:

```
m_db=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk IN (SELECT
d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
          55 |              1
          23 |              9
(2 rows)
```

ANY/SOME

Figure 4-9 shows the syntax of ANY/SOME.

Figure 4-9 any/some::=



- The right-hand side of ANY/SOME is a parenthesized subquery, which returns exactly one column.
- The expression on the left-hand side of ANY/SOME is evaluated and compared to each row of the subquery result using the given operator, which yields a Boolean result. The result of ANY is **true** if any true result is obtained. The result is **false** if no true result is found (including the case where the subquery returns no rows). SOME is a synonym of ANY. IN can be equivalently replaced with ANY.

Example:

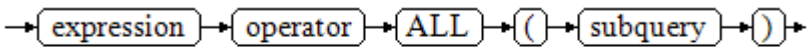
```
m_db=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk < ANY
(SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
--------------+----------------
           1 |              5
          12 |              8
          55 |              1
(3 rows)
```

ALL

Figure 4-10 shows the syntax of ALL.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 4-10 all::=



- The right-hand side of ALL is a parenthesized subquery, which returns exactly one column.
- The expression on the left-hand side of ALL is evaluated and compared to each row of the subquery result using the given operator, which yields a Boolean result. The result of ALL is **true** if all values are **true** (including the case where the subquery returns no rows). The result is **false** if any false result is found.

Example:

```
m_db=# SELECT sr_reason_sk,sr_customer_sk  FROM tpcds.store_returns WHERE sr_customer_sk <
all(SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
 sr_reason_sk | sr_customer_sk
--------------+----------------
(0 rows)

m_db=# DROP TABLE tpcds.store_returns;
m_db=# DROP TABLE tpcds.date_dim;
m_db=# DROP SCHEMA tpcds;
```

4.8.4 Row Expressions

The row comparison allows operators including =, <>, <, <=, >, and >=. The <=> operator or a similar operator is not supported.

The syntax is as follows:

```
row_constructor operator row_constructor
```

NOTE

- Both sides of the operator are row constructors. The values of both rows must have the same number of fields and they are compared with each other.
- The supported data types are consistent with those of comparison operators.

<, <=, >, and >=

For operators <, <=, >, and >=, the columns in rows are compared from left to right until a pair of columns that are not equal or are empty are detected. If the pair of fields contains at least one null value, the comparison result is null. Otherwise, the comparison result of this pair of fields is the final result. If no unequal or empty column is found, the values in the two rows are equal. The final result is determined based on the operator meaning.

Example:

```
m_db=# SELECT (1,2,NULL) < (1,3,0) AS RESULT;
  result
----------
 t
(1 row)

m_db=# SELECT (4,5,6) > (3,2,1) AS RESULT;
 result
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

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

m_db=# SELECT (4,1,1) > (3,2,1) AS RESULT;
 result
--------
 t
(1 row)

m_db=# SELECT ('TEST','DATA') > ('DATA','DATA') AS RESULT;
 result
--------
 t
(1 row)

m_db=# SELECT (4,1,1) > (3,2,NULL) AS RESULT;
 result
--------
 t
(1 row)

m_db=# SELECT (NULL,1,1) > (3,2,1) AS RESULT;
 result
--------

(1 row)

m_db=# SELECT (NULL,5,6) > (NULL,5,6) AS RESULT;
 result
--------

(1 row)

m_db=# SELECT (4,5,6) > (4,5,6) AS RESULT;
 result
--------
 f
(1 row)

m_db=# SELECT (2,2,5) >= (2,2,3) AS RESULT;
 result
--------
 t
(1 row)

m_db=# SELECT (2,2,1) <= (2,2,3) AS RESULT;
 result
--------
 t
(1 row)
```

= and <>

The use of operators = and <> is slightly different from other operators. If all columns in the two rows are not empty and meet the operator condition, the two rows meet the operator condition. If any column in the two rows is not empty and does not meet the operator condition, the two rows do not meet the operator condition. If any column in the two rows is empty, the comparison result is null.

Example:

```
m_db=# SELECT (1,2,3) = (1,2,3) AS RESULT;
 result
--------
 t
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# SELECT (1,2,3) <> (2,2,3) AS RESULT;
 result
--------
 t
(1 row)

m_db=# SELECT (2,2,3) <> (2,2,NULL) AS RESULT;
 result
--------

(1 row)

m_db=#  SELECT (NULL,5,6) <> (NULL,5,6) AS RESULT;
 result
--------

(1 row)
```

4.8.5 Time Interval Expressions

Syntax:

```
INTERVAL EXPR UNIT
```

Description: This is the syntax of the interval_expr expression, representing a time interval. INTERVAL is the keyword. EXPR is any expression and will be interpreted as a value of the time type. UNIT is a specific keyword and represents the unit of a value, such as, HOUR, DAY, or WEEK. The keyword INTERVAL and specifier are case-insensitive.

Table 4-83 describes the value range of UNIT in a time interval expression.

NOTE

- Generally, interval_expr is used with another value of the time type for computing. It can be used as the input parameter of the DATE_ADD(), DATE_SUB(), ADDDATE(), SUBDATE(), and TIMESTAMPADD() functions. It can also be used for computing together with another value of the time type using the plus and minus operators + and -.
- For the subtraction operation, interval_expr is permitted only on the right side because it makes no sense to subtract a time value from an interval.
- The time units listed in **Table 4-83** can be classified into single-time units and multi-time units.
  - Single-time units range from MICROSECOND to YEAR. With a single-time unit, EXPR is interpreted as a time value having one time unit. For example, '1'YEAR represents one year.
  - Multi-time units range from SECOND_MICROSECOND to YEAR_MONTH. With a multi-time unit, EXPR is interpreted to as a time value having multiple time units. For example, with **DAY_MINUTE** specified, EXPR is interpreted to have three parts including days, hours, and minutes.
- Consecutive numeric characters in EXPR will be interpreted as valid numbers and can be separated by any character. For example, '1:1:1' DAY_MINUTE represents 1 day 1 hour 1 minute. If interpreted valid digits do not map to all the time units contained in the multi-time unit, 0s are added to the leftmost parts. For example, '1:1' DAY_MINUTE represents 0 day 1 hour 1 minute.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 4-83 Value range of UNIT in a time interval expression

UNIT Value Range	Expected EXPR Format
MICROSECOND	MICROSECOND
SECOND	SECOND
MINUTE	MINUTE
HOUR	HOUR
DAY	DAY
WEEK	WEEK
MONTH	MONTH
QUARTER	QUARTER
YEAR	YEAR
SECOND_MICROSECOND	SECOND_MICROSECOND
MINUTE_MICROSECOND	MINUTE_MICROSECOND
MINUTE_SECOND	MINUTE_SECOND
HOUR_MICROSECOND	HOUR_MICROSECOND
HOUR_SECOND	HOUR_SECOND
HOUR_MINUTE	HOUR_MINUTE
DAY_MICROSECOND	DAY_MICROSECOND
DAY_SECOND	DAY_SECOND
DAY_MINUTE	DAY_MINUTE
DAY_HOUR	DAY_HOUR
YEAR_MONTH	YEAR_MONTH

Example:

```
m_db=# SELECT date_add('1999-12-31',interval '1' day);
  date_add
------------
 2000-01-01
(1 row)

m_db=# SELECT adddate('1998-12-03', interval '1' YEAR);
  adddate
------------
 1999-12-03
(1 row)

m_db=# SELECT date_sub('1999-12-31',interval '1' day);
  date_sub
------------
 1999-12-30
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
(1 row)

m_db=# SELECT subdate('1998-12-03', interval '1' YEAR);
  subdate
------------
 1997-12-03
(1 row)

m_db=# SELECT timestampadd(day, '1', '1999-12-31');
 timestampadd
--------------
 2000-01-01
(1 row)

m_db=# SELECT '1998-12-03' + INTERVAL '1' YEAR;
  ?column?
------------
 1999-12-03
(1 row)

m_db=# SELECT '1998-12-03' - INTERVAL '1' YEAR;
  ?column?
------------
 1997-12-03
(1 row)

m_db=# SELECT INTERVAL '1' YEAR + '1998-12-03';
  ?column?
------------
 1999-12-03
(1 row)

m_db=# SELECT interval '1' YEAR - '1998-12-03';
ERROR:  syntax error at or near "-"
LINE 1: select interval '1' YEAR - '1998-12-03';
                                 ^
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5 System Catalogs and System Views

When querying M-compatible system catalogs and system views, you need to enter their names in lowercase.

5.1 System Catalogs

M-compatible databases inherit the original GaussDB system catalogs. For details, see "System Catalogs and System Views > System Catalogs" in *Developer Guide*.

5.2 System Views

The M-compatible database inherits some system views of GaussDB. **Table 5-1** shows the inherited system views. For details about the system view definitions and view information, see section "System Catalogs and System Views" in *Developer Guide*.

NOTE

- M-compatible databases do not support auditing. The gs_auditing, gs_auditing_access, and gs_auditing_privilege views are empty.
- M-compatible databases do not support masking. The gs_masking view is empty.

Table 5-1 Views of GaussDB inherited by the M-compatible database

View Category	View Name
DATABASE LINKS	gs_db_links
Partitioned tables	gs_stat_all_partitions
	gs_stat_xact_all_partitions
	gs_statio_all_partitions
Auditing	gs_auditing
	gs_auditing_access

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Category	View Name
	gs_auditing_privilege
User and permission management	gs_db_privileges
	gs_labels
Dynamic data masking	gs_masking
Communications	gs_comm_listen_address_ext_info
	gs_get_listen_address_ext_info
	gs_static_threadpool_ctrl_status
Segment-page storage	global_gs_seg_datafiles
	global_gs_seg_extents
	global_gs_seg_segments
	global_gs_seg_spc_extents
	global_gs_seg_spc_segments
	gs_seg_datafile_layout
	gs_seg_datafiles
	gs_seg_extents
	gs_seg_segments
	gs_seg_segment_layout
	gs_seg_spc_segments
	gs_seg_spc_extents
	gs_seg_spc_remain_extents
	gs_seg_spc_remain_segments
SPM	gs_spm_sql_baseline
	gs_spm_sql_param
	gs_spm_sql_evolution
	gs_spm_sys_baseline
Workload management	gs_wlm_cgroup_info
	gs_wlm_rebuild_user_resource_pool
	gs_wlm_user_info
OLTP table compression	gs_adm_ilmevaluationdetails

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Category	View Name
	gs_adm_ilmdatamovementpolicies
	gs_adm_ilmobjects
	gs_adm_ilmparameters
	gs_adm_ilmpolicies
	gs_adm_ilmresults
	gs_adm_ilmtasks
	gs_my_ilmevaluationdetails
	gs_my_ilmdatamovementpolicies
	gs_my_ilmobjects
	gs_my_ilmpolicies
	gs_my_ilmresults
	gs_my_ilmtasks
Materialized view	gs_matviews
Other system views	global_bad_block_info
	global_clear_bad_block_info
	global_stat_hotkeys_info
	global_wal_sender_status
	gs_all_control_group_info
	gs_all_prepared_statements
	gs_bucket_locks
	gs_get_control_group_info
	gs_glc_memory_detail
	gs_global_archive_status
	gs_gsc_memory_detail
	gs_libcomm_fd_info
	gs_libcomm_memory_info
	gs_lsc_memory_detail
	gs_my_plan_trace
	gs_session_all_settings

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Category	View Name
	gs_sql_count
	gs_stat_db_cu
	gs_stat_session_cu
	gs_total_nodegroup_memory_detail
	gs_workload_rule_stat
	sys_dummy
Multi-tenant	mtd_user (You are advised not to use and query this view in distributed mode.)
Deprecated system views	gs_cluster_resource_info
	gs_session_cpu_statistics
	gs_session_memory_statistics
	gs_wlm_operator_history
	gs_wlm_operator_statistics
	gs_wlm_resource_pool
	gs_wlm_session_history
	gs_wlm_session_info
	gs_wlm_session_info_all
	gs_wlm_session_statistics
	gs_wlm_user_session_info
	gs_wlm_workload_records

The middle view is used by the CN to collect views in a distributed system. For details about the middle view, see **Table 5-2**. You are advised not to use it directly.

Table 5-2 Middle views

gs_local_adm_ilmdatamovementpoli-cies	gs_local_my_ilmdatamovementpolicies
gs_local_adm_ilmevaluationdetails	gs_local_my_ilmevaluationdetails
gs_local_adm_ilmobjects	gs_local_my_ilmobjects
gs_local_adm_ilmpolicies	gs_local_my_ilmpolicies
gs_local_adm_ilmresults	gs_local_my_ilmresults

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

gs_local_adm_ilmtasks	gs_local_my_ilmtasks

The M-compatible database also inherits the original GaussDB system views, such as dv_session_longops and dv_sessions. In addition to the original GaussDB system views, view aliases prefixed with gs_ are created in the corresponding system views, as described in **Table 5-3**. To ensure compatibility, preferentially use views whose alias prefixed with gs_.

Table 5-3 Aliases of the system view in the M-compatible database

View Name	View Alias
dv_session_longops	gs_dv_session_longops
dv_sessions	gs_dv_sessions
get_global_prepared_xacts	gs_get_global_prepared_xacts
mpp_tables	gs_mpp_tables
pg_available_extension_versions	gs_available_extension_versions
pg_available_extensions	gs_available_extensions
pg_comm_delay	gs_comm_delay
pg_comm_recv_stream	gs_comm_recv_stream
pg_comm_send_stream	gs_comm_send_stream
pg_comm_status	gs_comm_status
pg_control_group_config	gs_control_group_config
pg_cursors	gs_cursors
pg_ext_stats	gs_ext_stats
pg_get_invalid_backends	gs_get_invalid_backends
pg_get_senders_catchup_time	gs_get_senders_catchup_time
pg_group	gs_group
pg_gtt_attached_pids	gs_gtt_attached_pids
pg_gtt_relstats	gs_gtt_relstats
pg_gtt_stats	gs_gtt_stats
pg_indexes	gs_indexes
pg_locks	gs_locks
pg_node_env	gs_node_env
pg_os_threads	gs_os_threads

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Name	View Alias
pg_pooler_status	gs_pooler_status
pg_prepared_statements	gs_prepared_statements
pg_prepared_xacts	gs_prepared_xacts
pg_replication_slots	gs_replication_slots
pg_rlspolicies	gs_rlspolicies
pg_roles	gs_roles
pg_rules	gs_rules
pg_running_xacts	gs_running_xacts
pg_seclabels	gs_seclabels
pg_session_iostat	gs_session_iostat
pg_session_wlmstat	gs_session_wlmstat
pg_settings	gs_settings
pg_shadow	gs_shadow
pg_shared_memory_detail	gs_shared_memory_detail
pg_stat_activity	gs_stat_activity
pg_stat_activity_ng	gs_stat_activity_ng
pg_stat_all_indexes	gs_stat_all_indexes
pg_stat_all_tables	gs_stat_all_tables
pg_stat_bad_block	gs_stat_bad_block
pg_stat_bgwriter	gs_stat_bgwriter
pg_stat_database	gs_stat_database
pg_stat_database_conflicts	gs_stat_database_conflicts
pg_stat_replication	gs_stat_replication
pg_stat_sys_indexes	gs_stat_sys_indexes
pg_stat_sys_tables	gs_stat_sys_tables
pg_stat_user_functions	gs_stat_user_functions
pg_stat_user_indexes	gs_stat_user_indexes
pg_stat_user_tables	gs_stat_user_tables
pg_stat_xact_all_tables	gs_stat_xact_all_tables
pg_stat_xact_sys_tables	gs_stat_xact_sys_tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Name	View Alias
pg_stat_xact_user_functions	gs_stat_xact_user_functions
pg_stat_xact_user_tables	gs_stat_xact_user_tables
pg_statio_all_indexes	gs_statio_all_indexes
pg_statio_all_sequences	gs_statio_all_sequences
pg_statio_all_tables	gs_statio_all_tables
pg_statio_sys_indexes	gs_statio_sys_indexes
pg_statio_sys_sequences	gs_statio_sys_sequences
pg_statio_sys_tables	gs_statio_sys_tables
pg_statio_user_indexes	gs_statio_user_indexes
pg_statio_user_sequences	gs_statio_user_sequences
pg_statio_user_tables	gs_statio_user_tables
pg_stats	gs_stats
pg_tables	gs_tables
pg_tde_info	gs_tde_info
pg_thread_wait_status	gs_thread_wait_status
pg_timezone_abbrevs	gs_timezone_abbrevs
pg_timezone_names	gs_timezone_names
pg_total_memory_detail	gs_total_memory_detail
pg_total_user_resource_info	gs_total_user_resource_info
pg_total_user_resource_info_oid	gs_total_user_resource_info_oid
pg_user	gs_user
pg_user_mappings	gs_user_mappings
pg_variable_info	gs_variable_info
pg_views	gs_views
pg_wlm_statistics	gs_wlm_statistics
pgxc_comm_delay	gs_pgxc_comm_delay
pgxc_comm_recv_stream	gs_pgxc_comm_recv_stream
pgxc_comm_send_stream	gs_pgxc_comm_send_stream
pgxc_comm_status	gs_pgxc_comm_status
pgxc_get_stat_all_tables	gs_pgxc_get_stat_all_tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Name	View Alias
pgxc_get_table_skewness	gs_pgxc_get_table_skewness
pgxc_node_env	gs_pgxc_node_env
pgxc_os_threads	gs_pgxc_os_threads
pgxc_prepared_xacts	gs_pgxc_prepared_xacts
pgxc_running_xacts	gs_pgxc_running_xacts
pgxc_sql_count	gs_pgxc_sql_count
pgxc_stat_activity	gs_pgxc_stat_activity
pgxc_stat_activity_with_conninfo_new	gs_pgxc_stat_activity_with_conninfo_new
pgxc_stat_bad_block	gs_pgxc_stat_bad_block
pgxc_thread_wait_status	gs_pgxc_thread_wait_status
pgxc_total_memory_detail	gs_pgxc_total_memory_detail
pgxc_variable_info	gs_pgxc_variable_info
pgxc_wlm_operator_history	gs_pgxc_wlm_operator_history
pgxc_wlm_operator_info	gs_pgxc_wlm_operator_info
pgxc_wlm_operator_statistics	gs_pgxc_wlm_operator_statistics
pgxc_wlm_rebuild_user_respool	gs_pgxc_wlm_rebuild_user_respool
pgxc_wlm_session_history	gs_pgxc_wlm_session_history
pgxc_wlm_session_info	gs_pgxc_wlm_session_info
pgxc_wlm_session_statistics	gs_pgxc_wlm_session_statistics
pgxc_wlm_workload_records	gs_pgxc_wlm_workload_records
pv_file_stat	gs_pv_file_stat
pv_instance_time	gs_pv_instance_time
pv_os_run_info	gs_pv_os_run_info
pv_redo_stat	gs_pv_redo_stat
pv_session_memory	gs_pv_session_memory
pv_session_memory_context	gs_pv_session_memory_context
pv_session_memory_detail	gs_pv_session_memory_detail
pv_session_stat	gs_pv_session_stat
pv_session_time	gs_pv_session_time

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

View Name	View Alias
pv_thread_memory_context	gs_pv_thread_memory_context
pv_total_memory_detail	gs_pv_total_memory_detail

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6 Schemas

The M-compatible database inherits some of the original DBE_PERF Schema views of GaussDB. In addition, the M-compatible database creates view aliases prefixed with **gs**_ under the corresponding views. **Table 6-1** describes the mapping between view aliases and views. To ensure compatibility, preferentially use views whose alias prefixed with **gs**_.

Table 6-1 Aliases of the system views in the M-compatible database

Schema	View Name	View Alias
DBE_PERF	ai_watchdog_detection_warnings	gs_ai_watchdog_detection_warnings
DBE_PERF	ai_watchdog_ftask_status	gs_ai_watchdog_ftask_status
DBE_PERF	ai_watchdog_monitor_status	gs_ai_watchdog_monitor_status
DBE_PERF	ai_watchdog_parameters	gs_ai_watchdog_parameters
DBE_PERF	bgwriter_stat	gs_bgwriter_stat
DBE_PERF	class_vital_info	gs_class_vital_info
DBE_PERF	comm_delay	gs_comm_delay
DBE_PERF	comm_recv_stream	gs_comm_recv_stream
DBE_PERF	comm_send_stream	gs_comm_send_stream
DBE_PERF	comm_status	gs_comm_status
DBE_PERF	config_settings	gs_config_settings
DBE_PERF	file_iostat	gs_file_iostat
DBE_PERF	file_redo_iostat	gs_file_redo_iostat
DBE_PERF	global_active_session	gs_global_active_session

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	global_adio_completer_status	gs_global_adio_completer_status
DBE_PERF	global_aio_slot_usage_status	gs_global_aio_slot_usage_status
DBE_PERF	global_bgwriter_stat	gs_global_bgwriter_stat
DBE_PERF	global_candidate_status	gs_global_candidate_status
DBE_PERF	global_ckpt_status	gs_global_ckpt_status
DBE_PERF	global_comm_check_connection_status	gs_global_comm_check_connection_status
DBE_PERF	global_comm_delay	gs_global_comm_delay
DBE_PERF	global_comm_recv_stream	gs_global_comm_recv_stream
DBE_PERF	global_comm_send_stream	gs_global_comm_send_stream
DBE_PERF	global_comm_status	gs_global_comm_status
DBE_PERF	global_config_settings	gs_global_config_settings
DBE_PERF	global_double_write_status	gs_global_double_write_status
DBE_PERF	global_file_iostat	gs_global_file_iostat
DBE_PERF	global_file_redo_iostat	gs_global_file_redo_iostat
DBE_PERF	global_get_bgwriter_status	gs_global_get_bgwriter_status
DBE_PERF	global_instance_time	gs_global_instance_time
DBE_PERF	global_io_wait_info	gs_global_io_wait_info
DBE_PERF	global_locks	gs_global_locks
DBE_PERF	global_memory_node_detail	gs_global_memory_node_detail
DBE_PERF	global_operator_history	gs_global_operator_history
DBE_PERF	global_operator_history_table	gs_global_operator_history_table
DBE_PERF	global_operator_runtime	gs_global_operator_runtime
DBE_PERF	global_os_runtime	gs_global_os_runtime
DBE_PERF	global_os_threads	gs_global_os_threads
DBE_PERF	global_pagewriter_status	gs_global_pagewriter_status

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	global_parallel_decode_status	gs_global_parallel_decode_status
DBE_PERF	global_parallel_decode_thread_info	gs_global_parallel_decode_thread_info
DBE_PERF	global_plancache_status	gs_global_plancache_status
DBE_PERF	global_pooler_status	gs_global_pooler_status
DBE_PERF	global_prepare_statement_status	gs_global_prepare_statement_status
DBE_PERF	global_record_reset_time	gs_global_record_reset_time
DBE_PERF	global_recovery_status	gs_global_recovery_status
DBE_PERF	global_redo_status	gs_global_redo_status
DBE_PERF	global_rel_iostat	gs_global_rel_iostat
DBE_PERF	global_replication_slots	gs_global_replication_slots
DBE_PERF	global_replication_stat	gs_global_replication_stat
DBE_PERF	global_rto_status	gs_global_rto_status
DBE_PERF	global_session_memory	gs_global_session_memory
DBE_PERF	global_session_memory_detail	gs_global_session_memory_detail
DBE_PERF	global_session_stat	gs_global_session_stat
DBE_PERF	global_session_stat_activity	gs_global_session_stat_activity
DBE_PERF	global_session_time	gs_global_session_time
DBE_PERF	global_shared_memory_detail	gs_global_shared_memory_detail
DBE_PERF	global_single_flush_dw_status	gs_global_single_flush_dw_status
DBE_PERF	global_slow_query_history	gs_global_slow_query_history
DBE_PERF	global_slow_query_info	gs_global_slow_query_info
DBE_PERF	global_stat_all_indexes	gs_global_stat_all_indexes
DBE_PERF	global_stat_all_tables	gs_global_stat_all_tables
DBE_PERF	global_stat_bad_block	gs_global_stat_bad_block
DBE_PERF	global_stat_database	gs_global_stat_database

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	global_stat_database_conflicts	gs_global_stat_database_conflicts
DBE_PERF	global_stat_db_cu	gs_global_stat_db_cu
DBE_PERF	global_stat_session_cu	gs_global_stat_session_cu
DBE_PERF	global_stat_sys_indexes	gs_global_stat_sys_indexes
DBE_PERF	global_stat_sys_tables	gs_global_stat_sys_tables
DBE_PERF	global_stat_user_functions	gs_global_stat_user_functions
DBE_PERF	global_stat_user_indexes	gs_global_stat_user_indexes
DBE_PERF	global_stat_user_tables	gs_global_stat_user_tables
DBE_PERF	global_stat_xact_all_tables	gs_global_stat_xact_all_tables
DBE_PERF	global_stat_xact_sys_tables	gs_global_stat_xact_sys_tables
DBE_PERF	global_stat_xact_user_functions	gs_global_stat_xact_user_functions
DBE_PERF	global_stat_xact_user_tables	gs_global_stat_xact_user_tables
DBE_PERF	global_statement_complex_history	gs_global_statement_complex_history
DBE_PERF	global_statement_complex_history_table	gs_global_statement_complex_history_table
DBE_PERF	global_statement_complex_runtime	gs_global_statement_complex_runtime
DBE_PERF	global_statement_count	gs_global_statement_count
DBE_PERF	global_statio_all_indexes	gs_global_statio_all_indexes
DBE_PERF	global_statio_all_sequences	gs_global_statio_all_sequences
DBE_PERF	global_statio_all_tables	gs_global_statio_all_tables
DBE_PERF	global_statio_sys_indexes	gs_global_statio_sys_indexes
DBE_PERF	global_statio_sys_sequences	gs_global_statio_sys_sequences
DBE_PERF	global_statio_sys_tables	gs_global_statio_sys_tables
DBE_PERF	global_statio_user_indexes	gs_global_statio_user_indexes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	global_statio_user_sequences	gs_global_statio_user_sequences
DBE_PERF	global_statio_user_tables	gs_global_statio_user_tables
DBE_PERF	global_streaming_hadr_rto_and_rpo_stat	gs_global_streaming_hadr_rto_and_rpo_stat
DBE_PERF	global_thread_wait_status	gs_global_thread_wait_status
DBE_PERF	global_threadpool_status	gs_global_threadpool_status
DBE_PERF	global_transactions_prepared_xacts	gs_global_transactions_prepared_xacts
DBE_PERF	global_transactions_running_xacts	gs_global_transactions_running_xacts
DBE_PERF	global_user_transaction	gs_global_user_transaction
DBE_PERF	global_wait_events	gs_global_wait_events
DBE_PERF	global_wlm_workload_runtime	gs_global_wlm_workload_runtime
DBE_PERF	global_workload_transaction	gs_global_workload_transaction
DBE_PERF	instance_time	gs_instance_time
DBE_PERF	local_active_session	gs_local_active_session
DBE_PERF	local_io_wait_info	gs_local_io_wait_info
DBE_PERF	local_plancache_clean	gs_local_plancache_clean
DBE_PERF	local_plancache_status	gs_local_plancache_status
DBE_PERF	local_prepare_statement_status	gs_local_prepare_statement_status
DBE_PERF	local_rel_iostat	gs_local_rel_iostat
DBE_PERF	local_threadpool_status	gs_local_threadpool_status
DBE_PERF	locks	gs_locks
DBE_PERF	memory_node_detail	gs_memory_node_detail
DBE_PERF	memory_node_ng_detail	gs_memory_node_ng_detail
DBE_PERF	operator_history	gs_operator_history
DBE_PERF	operator_history_table	gs_operator_history_table
DBE_PERF	operator_runtime	gs_operator_runtime
DBE_PERF	os_runtime	gs_os_runtime

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	os_threads	gs_os_threads
DBE_PERF	parallel_decode_status	gs_parallel_decode_status
DBE_PERF	parallel_decode_thread_info	gs_parallel_decode_thread_info
DBE_PERF	perf_query	gs_perf_query
DBE_PERF	pooler_status	gs_pooler_status
DBE_PERF	replication_slots	gs_replication_slots
DBE_PERF	replication_stat	gs_replication_stat
DBE_PERF	session_cpu_runtime	gs_session_cpu_runtime
DBE_PERF	session_memory_detail	gs_session_memory_detail
DBE_PERF	session_memory_runtime	gs_session_memory_runtime
DBE_PERF	session_stat	gs_session_stat
DBE_PERF	session_stat_activity	gs_session_stat_activity
DBE_PERF	session_time	gs_session_time
DBE_PERF	session_memory	gs_session_memory
DBE_PERF	shared_memory_detail	gs_shared_memory_detail
DBE_PERF	slow_query_history	gs_slow_query_history
DBE_PERF	slow_query_info	gs_slow_query_info
DBE_PERF	stat_all_indexes	gs_stat_all_indexes
DBE_PERF	stat_all_tables	gs_stat_all_tables
DBE_PERF	stat_bad_block	gs_stat_bad_block
DBE_PERF	stat_database	gs_stat_database
DBE_PERF	stat_database_conflicts	gs_stat_database_conflicts
DBE_PERF	stat_sys_indexes	gs_stat_sys_indexes
DBE_PERF	stat_sys_tables	gs_stat_sys_tables
DBE_PERF	stat_user_functions	gs_stat_user_functions
DBE_PERF	stat_user_indexes	gs_stat_user_indexes
DBE_PERF	stat_user_tables	gs_stat_user_tables
DBE_PERF	stat_xact_all_tables	gs_stat_xact_all_tables
DBE_PERF	stat_xact_sys_tables	gs_stat_xact_sys_tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	stat_xact_user_functions	gs_stat_xact_user_functions
DBE_PERF	stat_xact_user_tables	gs_stat_xact_user_tables
DBE_PERF	statement	gs_statement
DBE_PERF	statement_complex_history	gs_statement_complex_history
DBE_PERF	statement_complex_history_table	gs_statement_complex_history_table
DBE_PERF	statement_complex_runtime	gs_statement_complex_runtime
DBE_PERF	statement_count	gs_statement_count
DBE_PERF	statement_history	gs_statement_history
DBE_PERF	statement_iostat_complex_runtime	gs_statement_iostat_complex_runtime
DBE_PERF	statement_responsetime_percentile	gs_statement_responsetime_percentile
DBE_PERF	statement_user_complex_history	gs_statement_user_complex_history
DBE_PERF	statement_wlmstat_complex_runtime	gs_statement_wlmstat_complex_runtime
DBE_PERF	statio_all_indexes	gs_statio_all_indexes
DBE_PERF	statio_all_sequences	gs_statio_all_sequences
DBE_PERF	statio_all_tables	gs_statio_all_tables
DBE_PERF	statio_sys_indexes	gs_statio_sys_indexes
DBE_PERF	statio_sys_sequences	gs_statio_sys_sequences
DBE_PERF	statio_sys_tables	gs_statio_sys_tables
DBE_PERF	statio_user_indexes	gs_statio_user_indexes
DBE_PERF	statio_user_sequences	gs_statio_user_sequences
DBE_PERF	statio_user_tables	gs_statio_user_tables
DBE_PERF	summary_file_iostat	gs_summary_file_iostat
DBE_PERF	summary_file_redo_iostat	gs_summary_file_redo_iostat
DBE_PERF	summary_rel_iostat	gs_summary_rel_iostat
DBE_PERF	summary_stat_all_indexes	gs_summary_stat_all_indexes
DBE_PERF	summary_stat_all_tables	gs_summary_stat_all_tables

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	summary_stat_bad_block	gs_summary_stat_bad_block
DBE_PERF	summary_stat_database	gs_summary_stat_database
DBE_PERF	summary_stat_database_conflicts	gs_summary_stat_database_conflicts
DBE_PERF	summary_stat_sys_indexes	gs_summary_stat_sys_indexes
DBE_PERF	summary_stat_sys_tables	gs_summary_stat_sys_tables
DBE_PERF	summary_stat_user_functions	gs_summary_stat_user_functions
DBE_PERF	summary_stat_user_indexes	gs_summary_stat_user_indexes
DBE_PERF	summary_stat_user_tables	gs_summary_stat_user_tables
DBE_PERF	summary_stat_xact_all_tables	gs_summary_stat_xact_all_tables
DBE_PERF	summary_stat_xact_sys_tables	gs_summary_stat_xact_sys_tables
DBE_PERF	summary_stat_xact_user_functions	gs_summary_stat_xact_user_functions
DBE_PERF	summary_stat_xact_user_tables	gs_summary_stat_xact_user_tables
DBE_PERF	summary_statement	gs_summary_statement
DBE_PERF	summary_statement_count	gs_summary_statement_count
DBE_PERF	summary_statio_all_indexes	gs_summary_statio_all_indexes
DBE_PERF	summary_statio_all_sequences	gs_summary_statio_all_sequences
DBE_PERF	summary_statio_all_tables	gs_summary_statio_all_tables
DBE_PERF	summary_statio_sys_indexes	gs_summary_statio_sys_indexes
DBE_PERF	summary_statio_sys_sequences	gs_summary_statio_sys_sequences
DBE_PERF	summary_statio_sys_tables	gs_summary_statio_sys_tables
DBE_PERF	summary_statio_user_indexes	gs_summary_statio_user_indexes

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	summary_statio_user_sequences	gs_summary_statio_user_sequences
DBE_PERF	summary_statio_user_tables	gs_summary_statio_user_tables
DBE_PERF	summary_transactions_prepared_xacts	gs_summary_transactions_prepared_xacts
DBE_PERF	summary_transactions_running_xacts	gs_summary_transactions_running_xacts
DBE_PERF	summary_user_login	gs_summary_user_login
DBE_PERF	summary_workload_sql_count	gs_summary_workload_sql_count
DBE_PERF	summary_workload_sql_elapse_time	gs_summary_workload_sql_elapse_time
DBE_PERF	summary_workload_transaction	gs_summary_workload_transaction
DBE_PERF	thread_wait_status	gs_thread_wait_status
DBE_PERF	track_memory_context_detail	gs_track_memory_context_detail
DBE_PERF	transactions_prepared_xacts	gs_transactions_prepared_xacts
DBE_PERF	transactions_running_xacts	gs_transactions_running_xacts
DBE_PERF	user_login	gs_user_login
DBE_PERF	user_transaction	gs_user_transaction
DBE_PERF	wait_event_info	gs_wait_event_info
DBE_PERF	wait_events	gs_wait_events
DBE_PERF	wlm_cgroup_config	gs_wlm_cgroup_config
DBE_PERF	wlm_cluster_resource_runtime	gs_wlm_cluster_resource_runtime
DBE_PERF	wlm_controlgroup_config	gs_wlm_controlgroup_config
dbe_perf	wlm_controlgroup_ng_config	gs_wlm_controlgroup_ng_config
DBE_PERF	wlm_resourcepool_runtime	gs_wlm_resourcepool_runtime
DBE_PERF	wlm_user_resource_config	gs_wlm_user_resource_config

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Schema	View Name	View Alias
DBE_PERF	wlm_user_resource_runtime	gs_wlm_user_resource_runtime
DBE_PERF	wlm_workload_history_info	gs_wlm_workload_history_info
DBE_PERF	wlm_workload_runtime	gs_wlm_workload_runtime
DBE_PERF	workload_sql_count	gs_workload_sql_count
DBE_PERF	workload_sql_elapse_time	gs_workload_sql_elapse_time
DBE_PERF	workload_transaction	gs_workload_transaction

6.1 Information Schemas

The M-compatible database inherits some of the original views under the GaussDB Information Schema. **Table 6-2** shows the inherited views. For details about the view definitions and view information, see section "Schema" in *Developer Guide*.

Table 6-2 Views of GaussDB inherited by the M-compatible database

_gs_foreign_data_wrappers	gs_foreign_table_options
_gs_foreign_servers	gs_foreign_tables
_gs_foreign_table_columns	gs_information_schema_catalog_name
_gs_foreign_tables	gs_key_column_usage
_gs_user_mappings	gs_parameters
gs_administrable_role_authorizations	gs_referential_constraints
gs_applicable_roles	gs_role_column_grants
gs_attributes	gs_role_routine_grants
gs_character_sets	gs_role_table_grants
gs_check_constraint_routine_usage	gs_role_udt_grants
gs_check_constraints	gs_role_usage_grants
gs_collation_character_set_applicability	gs_routine_privileges
gs_collations	gs_routines
gs_column_domain_usage	gs_schemata
gs_column_options	gs_sequences

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

gs_column_privileges	gs_table_constraints
gs_column_udt_usage	gs_table_privileges
gs_columns	gs_tables
gs_constraint_column_usage	gs_triggered_update_columns
gs_constraint_table_usage	gs_triggers
gs_data_type_privileges	gs_udt_privileges
gs_domain_constraints	gs_usage_privileges
gs_domain_udt_usage	gs_user_defined_types
gs_domains	gs_user_mapping_options
gs_element_types	gs_user_mappings
gs_enabled_roles	gs_view_column_usage
gs_foreign_data_wrapper_options	gs_view_routine_usage
gs_foreign_data_wrappers	gs_view_table_usage
gs_foreign_server_options	gs_views
gs_foreign_servers	-

6.1.1 character_sets

The character_sets view provides information about available character sets. For details, see **Table 6-3**. This view is read-only. All users have the read permission on this view.

Table 6-3 information_schema.character_sets columns

Name	Type	Description
CHARACTER_SET_NAME	varchar(32)	Name of a character set.
DEFAULT_COLLATE_NAME	varchar(32)	Default collation for the character set.
DESCRIPTION	varchar(60)	Description of the default collation used by the character set.
MAXLEN	bigint	Maximum number of bytes required to store a character.

The control behavior of the GUC parameter **m_format_dev_version** is forward compatible.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

If **m_format_dev_version** is set to **s1**, you can view the information about the character sets of the current database in the view.

```
m_db=# SET m_format_dev_version=s1;
SET
m_db=# SELECT * FROM information_schema.character_sets;
 CHARACTER_SET_NAME | DEFAULT_COLLATE_NAME |          DESCRIPTION          | MAXLEN
--------------------+----------------------+-------------------------------+--------
 UTF8               | utf8mb4_general_ci   | utf8mb4_general_ci collation |      4
(1 row)
```

6.1.2 collations

The collations view provides information about collations for each character set. For details, see **Table 6-4**. This view is read-only. All users have the read permission on this view.

Table 6-4 information_schema.collations columns

Name	Type	Description
COLLATION_NAME	varchar(32)	Name of the collation.
CHARACTER_SET_NAME	varchar(32)	Name of the character set with which the collation is associated.
ID	bigint	Collation id. For details, see pg_collation.id.
IS_DEFAULT	varchar(3)	Specifies whether the collation is the default for its character set.
IS_COMPILED	varchar(3)	Specifies whether the character set is compiled into the server.
SORTLEN	bigint	Related to the amount of memory required to sort strings expressed in the character set. This column is not supported in the current version, and the value is **null**.

The control behavior of the GUC parameter **m_format_dev_version** is forward compatible.

If the value of **m_format_dev_version** is set to **s1**, you can view the information about collation of the character sets of only the current database in the view.

```
m_db=# SET m_format_dev_version=s1;
SET
m_db=#
m_db=# SELECT * FROM information_schema.collations;
   COLLATION_NAME   | CHARACTER_SET_NAME | ID  | IS_DEFAULT | IS_COMPILED | SORTLEN
--------------------+--------------------+-----+------------+-------------+---------
 utf8mb4_0900_ai_ci | UTF8               | 255 |            | Yes         |
 utf8_bin           | UTF8               |  83 |            | Yes         |
 utf8_unicode_ci    | UTF8               | 192 |            | Yes         |
 utf8_general_ci    | UTF8               |  33 |            | Yes         |
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
utf8mb4_bin        | UTF8              |  46 |           | Yes        |
utf8mb4_unicode_ci | UTF8              | 224 |           | Yes        |
utf8mb4_general_ci | UTF8              |  45 | Yes       | Yes        |
POSIX              | UTF8              |     |           | Yes        |
C                  | UTF8              |     |           | Yes        |
default            | UTF8              |     |           | Yes        |
(10 rows)
```

6.1.3 collation_character_set_applicability

The collation_character_set_applicability view indicates a matching relationship between character sets and collations. For details, see **Table 6-5**. This view is read-only. All users have the read permission on this view.

Table 6-5 information_schema.collation_character_set_applicability columns

Name	Type	Description
COLLATION_NAME	varchar(32)	Name of the collation.
CHARACTER_SET_NAME	varchar(32)	Name of the character set with which the collation is associated.

The control behavior of the GUC parameter **m_format_dev_version** is forward compatible.

If the value of **m_format_dev_version** is set to **s1**, you can view the information about collation of the character sets of only the current database in the view.

```
m_db=# SET m_format_dev_version=s1;
SET
m_db=# SELECT * FROM information_schema.collation_character_set_applicability;
   COLLATION_NAME    | CHARACTER_SET_NAME
---------------------+--------------------
 utf8mb4_0900_ai_ci | UTF8
 utf8_bin           | UTF8
 utf8_unicode_ci    | UTF8
 utf8_general_ci    | UTF8
 utf8mb4_bin        | UTF8
 utf8mb4_unicode_ci | UTF8
 utf8mb4_general_ci | UTF8
 POSIX              | UTF8
 C                  | UTF8
 default            | UTF8
(10 rows)
```

NOTE

The utf8 and utf8mb4 are the same character set in the M-compatible database. Therefore, collations of utf8 and utf8mb4 are equivalent to each other.

6.1.4 columns

The columns view provides information about columns in tables. The related **st_geometry_columns** table provides information about table columns that store spatial data. For details, see **Table 6-6**. This view is read-only. All users have the read permission on this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-6 information_schema.columns columns

Name	Type	Description
TABLE_CATALOG	varchar(512)	Name of the catalog (database) to which the table containing the column belongs. The value is the name of the current database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the table containing the column belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table containing the column. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
COLUMN_NAME	varchar(64)	Name of the column.
ORDINAL_POSITION	bigint unsigned	Position of the column within the table.
COLUMN_DEFAULT	longtext	Default value of the column.
IS_NULLABLE	varchar(3)	Nullability of the column.
DATA_TYPE	varchar(64)	Data type of the column.
CHARACTER_MAXIMUM_LENGTH	bigint unsigned	Maximum length of string columns, in characters.
CHARACTER_OCTET_LENGTH	bigint unsigned	Maximum length of string columns, in bytes.
NUMERIC_PRECISION	bigint unsigned	Precision of numeric columns.
NUMERIC_SCALE	bigint unsigned	Scale of numeric columns.
DATETIME_PRECISION	bigint unsigned	Precision of fractional seconds in temporal columns.
CHARACTER_SET_NAME	varchar(32)	Character set name for string columns.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
COLLATION_NAME	varchar(32)	Collation rule name for string columns.
COLUMN_TYPE	longtext	Data type of the column. The value of **column_type** contains the type name and possibly other information, such as the precision or length.
COLUMN_KEY	varchar(3)	Specifies whether the column is indexed. ● **pri**: The column is a primary key or is one of the columns in a multiple-column primary key. ● **uni**: The column is the first column of a unique index. ● **mul**: The column is the first column of a non-unique index. ● Empty: The column is not indexed or is not **pri**, **uni**, or **mul**.
EXTRA	varchar(30)	Any other available information about the specified column. The value is not NULL in the following cases: ● **auto_increment**: columns with the AUTO_INCREMENT attribute. ● **on update CURRENT_TIMESTAMP**: columns with the on update CURRENT_TIMESTAMP attribute for time columns. ● **STORED GENERATED**: columns with the STORED GENERATED attribute for generated columns. ● **VIRTUAL GENERATED**: columns with the VIRTUAL GENERATED attribute for generated columns.
PRIVILEGES	varchar(80)	Permission the current user has on the column.
COLUMN_COMMENT	varchar(1024)	Any comment included in the column definition.
GENERATION_EXPRESSION	longtext	For a generated column, this column displays the expression used to compute the column value. For a non-generated column, this column is empty.

NOTE

This view can be queried only on CNs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6.1.5 column_privileges

The column_privileges view provides information about column permissions. The values can be obtained from the m_schema.columns_priv view. For details, see **Table 6-7**. This view is read-only. All users have the read permission on this view.

Table 6-7 information_schema.column_privileges columns

Name	Type	Description
GRANTEE	varchar(81)	Name of the account to which the permission is granted.
TABLE_CATALOG	varchar(512)	Name of the catalog (database) to which the table containing the column belongs.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the table containing the column belongs.
TABLE_NAME	varchar(64)	Name of the table containing the column.
COLUMN_NAME	varchar(64)	Name of the column.
PRIVILEGE_TYPE	varchar(64)	Permission granted. The value is any permission that can be granted at the column level.
IS_GRANTABLE	varchar(3)	If the user has the GRANT OPTION permission, the value is **yes**; otherwise, the value is **no**.

6.1.6 engines

The engines view provides information about storage engines. It is used to check whether a storage engine is supported or check the default engine. For details, see **Table 6-8**. This view is read-only. All users have the read permission on this view.

Table 6-8 information_schema.engines columns

Name	Type	Description
ENGINE	varchar(64)	Engine name. Return values: **Ustore** and **Astore**.
SUPPORT	varchar(8)	Level of the storage engine supported by the server. Return values: **YES** and **YES**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
COMMENT	varchar(80)	Brief description of the engine. Return values: **In-place update store** and **Append update store**.
TRANSACTIONS	varchar(3)	Specifies whether the engine supports transactions. Return values: **YES** and **YES**.
XA	varchar(3)	Specifies whether the engine supports XA transactions. Return values: **YES** and **YES**.
SAVEPOINTS	varchar(3)	Specifies whether the engine supports savepoints. Return values: **YES** and **YES**.

6.1.7 events

The events view displays the information about scheduled tasks created by users in the database. For details, see **Table 6-9**. This view is read-only. All users have the read permission on this view, but they can read only the records (schemas) that they have the access permission.

Table 6-9 information_schema.events columns

Name	Type	Description
EVENT_CATALOG	varchar(64)	Name of the database in which the job will be executed. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
EVENT_SCHEMA	varchar(64)	Name of the schema used for job execution. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
EVENT_NAME	varchar(64)	Name of a scheduled task or program.
DEFINER	varchar(93)	Username of the creator.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
TIME_ZONE	varchar(64)	Time zone for job scheduling, and the value is **system**.
EVENT_BODY	varchar(8)	This column is not supported in the current version, and the value is **null**.
EVENT_DEFINITION	longtext	Job content, which is the program content in the scheduled task.
EVENT_TYPE	varchar(9)	Type of the job. This column is not supported in the current version, and the value is **null**.
EXECUTE_AT	datetime	Start time of a one-time job.
INTERVAL_VALUE	varchar(256)	Execution interval of a periodic job.
INTERVAL_FIELD	varchar(18)	This column is not supported in the current version, and the value is **null**.
SQL_MODE	varchar(8192)	SQL mode that takes effect when a job is created or modified. This column is not supported in the current version, and the value is **null**.
STARTS	datetime	Start time of a periodic job.
ENDS	datetime	End time of a periodic job.
STATUS	varchar(18)	Status of the job. The value is **r**, **s**, **f**, or **d**. The default value is **s**. The options are described as follows: ● r=running ● s=successfully finished ● f=job failed ● d=disabled
ON_COMPLETION	varchar(12)	This column is not supported in the current version, and the value is **null**.
CREATED	datetime	Time when a job is created. This column is not supported in the current version, and the value is **null**.
LAST_ALTERED	datetime	Time when a job is last modified. This column is not supported in the current version, and the value is **null**.
LAST_EXECUTED	datetime	Time when a job was executed last time.
EVENT_COMMENT	varchar(64)	This column is not supported in the current version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
ORIGINATOR	bigint	This column is not supported in the current version, and the value is **null**.
CHARACTER_SET_CLIENT	varchar(32)	This column is not supported in the current version, and the value is **null**.
COLLATION_CONNECTION	varchar(32)	This column is not supported in the current version, and the value is **null**.
DATABASE_COLLATION	varchar(32)	Sequence used by the database.

6.1.8 files

The files view provides information about the files that store tablespace data. For details, see **Table 6-10**. This view is read-only. All users have the read permission on this view.

Table 6-10 information_schema.files columns

Name	Type	Description
FILE_ID	bigint	Identifier of a file. This column is not supported in the current version, and the value is **null**.
FILE_NAME	varchar(4000)	Name of the file. This column is not supported in the current version, and the value is **null**.
FILE_TYPE	varchar(20)	Type of the file. This column is not supported in the current version, and the value is **null**.
TABLESPACE_NAME	varchar(64)	Name of the tablespace associated with the file. This column is not supported in the current version, and the value is **null**.
TABLE_CATALOG	varchar(64)	This column is not supported in the current version, and the value is **null**.
TABLE_SCHEMA	varchar(64)	This column is not supported in the current version, and the value is **null**.
TABLE_NAME	varchar(64)	This column is not supported in the current version, and the value is **null**.
LOGFILE_GROUP_NAME	varchar(64)	This column is not supported in the current version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
LOGFILE_GROUP_NUMBER	bigint	This column is not supported in the current version, and the value is **null**.
ENGINE	varchar(64)	Storage engine. This column is not supported in the current version, and the value is **null**.
FULLTEXT_KEYS	varchar(64)	This column is not supported in the current version, and the value is **null**.
DELETED_ROWS	bigint	This column is not supported in the current version, and the value is **null**.
UPDATE_COUNT	bigint	This column is not supported in the current version, and the value is **null**.
FREE_EXTENTS	bigint	This column is not supported in the current version, and the value is **null**.
TOTAL_EXTENTS	bigint	This column is not supported in the current version, and the value is **null**.
EXTENT_SIZE	bigint	This column is not supported in the current version, and the value is **null**.
INITIAL_SIZE	bigint unsigned	This column is not supported in the current version, and the value is **null**.
MAXIMUM_SIZE	bigint unsigned	This column is not supported in the current version, and the value is **null**.
AUTOEXTEND_SIZE	bigint unsigned	This column is not supported in the current version, and the value is **null**.
CREATION_TIME	datetime	This column is not supported in the current version, and the value is **null**.
LAST_UPDATE_TIME	datetime	This column is not supported in the current version, and the value is **null**.
LAST_ACCESS_TIME	datetime	This column is not supported in the current version, and the value is **null**.
RECOVER_TIME	bigint	This column is not supported in the current version, and the value is **null**.
TRANSACTION_COUNTER	bigint	This column is not supported in the current version, and the value is **null**.
VERSION	bigint unsigned	This column is not supported in the current version, and the value is **null**.
ROW_FORMAT	varchar(10)	This column is not supported in the current version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
TABLE_ROWS	bigint unsigned	This column is not supported in the current version, and the value is **null**.
AVG_ROW_LENGTH	bigint unsigned	This column is not supported in the current version, and the value is **null**.
DATA_LENGTH	bigint unsigned	This column is not supported in the current version, and the value is **null**.
MAX_DATA_LENGTH	bigint unsigned	This column is not supported in the current version, and the value is **null**.
INDEX_LENGTH	bigint unsigned	This column is not supported in the current version, and the value is **null**.
DATA_FREE	bigint unsigned	This column is not supported in the current version, and the value is **null**.
CREATE_TIME	datetime	This column is not supported in the current version, and the value is **null**.
UPDATE_TIME	datetime	This column is not supported in the current version, and the value is **null**.
CHECK_TIME	datetime	This column is not supported in the current version, and the value is **null**.
CHECKSUM	bigint unsigned	This column is not supported in the current version, and the value is **null**.
STATUS	varchar(20)	This column is not supported in the current version, and the value is **null**.
EXTRA	varchar(255)	This column is not supported in the current version, and the value is **null**.

6.1.9 global_status

The global_status view provides the system status information about the currently connected database node. For details, see **Table 6-11**. This view is read-only. All users have the read permission on this view. Currently, only **Threads_connected** and **Uptime** are supported, which are used to display the number of global database connections and running time, respectively.

Table 6-11 information_schema.global_status columns

Name	Type	Description
VARIABLE_NAME	varchar(64)	Variable name of the server system status information.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
VARIABLE_VALUE	varchar(1024)	Variable value of the server system status information.

6.1.10 global_variables

The global_variables view provides information about global variables. For details, see **Table 6-12**. This view is read-only. All users have the read permission on this view.

Table 6-12 information_schema.global_variables columns

Name	Type	Description
VARIABLE_NAME	varchar(64)	Name of a variable.
VARIABLE_VALUE	varchar(1024)	Value of the variable.

6.1.11 key_column_usage

The key_column_usage view describes which key columns have constraints. This view provides no information about functional key parts because they are expressions. The view provides information only about columns. For details, see **Table 6-13**. This view is read-only. All users have the read permission on this view.

NOTICE

Constraint: This view displays only one data record for expression constraints (including composite expression constraints) and does not display column names.

Table 6-13 information_schema.key_column_usage columns

Name	Type	Description
CONSTRAINT_CATALOG	varchar(512)	Name of the catalog to which the constraint belongs. The value is the name of the current database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
CONSTRAINT_SCHEMA	varchar(64)	Name of the schema to which the constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_NAME	varchar(64)	Name of the constraint.
TABLE_CATALOG	varchar(512)	Name of the catalog to which the table belongs. The value is the name of the current database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema (database) to which the table belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table that has the constraint. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
COLUMN_NAME	varchar(64)	Name of the column that has the constraint. If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
ORDINAL_POSITION	bigint	Column's position within the constraint. Column positions are numbered from 1.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
POSITION_IN_UNIQUE_CONSTRAINT	bigint	Null for UNIQUE and PRIMARY KEY constraints. For FOREIGN KEY constraints, this column indicates the ordinal position in key of the table that is being referenced.
REFERENCED_TABLE_SCHEMA	varchar(64)	Name of the schema referenced by the constraint. This is not supported in this version, and the value is **null**. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
REFERENCED_TABLE_NAME	varchar(64)	Name of the table referenced by the constraint. This is not supported in this version, and the value is **null**. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
REFERENCED_COLUMN_NAME	varchar(64)	Name of the column referenced by the constraint. This is not supported in this version, and the value is **null**.

6.1.12 optimizer_trace

The optimizer_trace view provides information produced by the optimizer tracing capability for traced statements. For details, see **Table 6-14**. This view is read-only. All users have the read permission on this view.

Table 6-14 information_schema.optimizer_trace columns

Name	Type	Description
QUERY	longtext	This column is not supported in the current version, and the value is **null**.
TRACE	longtext	This column is not supported in the current version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
MISSING_BYTES_BEYOND_MAX_MEM_SIZE	integer	This column is not supported in the current version, and the value is **null**.
INSUFFICIENT_PRIVILEGES	tinyint	This column is not supported in the current version, and the value is **null**.

6.1.13 parameters

The parameters view provides parameters about stored procedures and storage functions, as well as information about the return values of storage functions. For details, see **Table 6-15**. This view does not include built-in or loadable functions. This view is read-only. All users have the read permission on this view.

Table 6-15 information_schema.parameters columns

Name	Type	Description
SPECIFIC_CATALOG	varchar(512)	Name of the catalog to which the routine containing the parameter belongs. The value is the name of the current database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
SPECIFIC_SCHEMA	varchar(64)	Name of the schema to which the routine containing the parameter belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
SPECIFIC_NAME	varchar(64)	Name of the routine containing the parameter.
ORDINAL_POSITION	integer	For successive parameters of a stored procedure or function, the values of **ordinal_position** are **1**, **2**, **3**, and so forth.
PARAMETER_MODE	varchar(5)	Mode of the parameter. The value is **in**, **out**, or **inout**. The value is **null** for a stored function return value.
PARAMETER_NAME	varchar(64)	Name of the parameter. The value is **null** for a stored function return value.
DATA_TYPE	varchar(64)	Data type of the parameter.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
CHARACTER_MAXIMUM_LENGTH	integer	Maximum length of string parameters, in characters. This is not supported in this version, and the value is **null**.
CHARACTER_OCTET_LENGTH	integer	Maximum length of string parameters, in bytes. This is not supported in this version, and the value is **null**.
NUMERIC_PRECISION	bigint	Precision of numeric parameters. This is not supported in this version, and the value is **null**.
NUMERIC_SCALE	integer	Scale of numeric parameters. This is not supported in this version, and the value is **null**.
DATETIME_PRECISION	bigint	Precision of fractional seconds in temporal parameters. This is not supported in this version, and the value is **null**.
CHARACTER_SET_NAME	varchar(64)	Character set name of character string parameters. This is not supported in this version, and the value is **null**.
COLLATION_NAME	varchar(64)	Collation name of character string parameters. This is not supported in this version, and the value is **null**.
DTD_IDENTIFIER	longtext	Data type of the parameter.
ROUTINE_TYPE	varchar(9)	• **procedure**: stored procedure • **function**: stored function

6.1.14 partitions

The partitions view displays the information about partitioned tables and table partitions in the database. The information about level-1 partitions and level-2 partitions is recorded separately. For details, see **Table 6-16**. This view is read-only. All users have the read permission on this view. A user can view only the records that the user has the access permission. Some information in the view is obtained based on statistics. Run **ANALYZE** before checking the information. (If data is updated in the database, you are advised to delay running **ANALYZE**.) Level-2 partitions are not supported in distributed mode. Therefore, the values of columns related to level-2 partitions in this view are always **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-16 information_schema.partitions columns

Name	Type	Description
TABLE_CATALOG	varchar(512)	Name of the catalog to which the table belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
PARTITION_NAME	varchar(64)	Name of the partition.
SUBPARTITION_NAME	varchar(64)	Name of the level-2 partition. If a partition is not a level-2 partition, the value is **null**.
PARTITION_ORDINAL_POSITION	bigint unsigned	Position of the partition in the table.
SUBPARTITION_ORDINAL_POSITION	bigint unsigned	Position of the level-2 partition in the partition. If a partition is not a level-2 partition, the value is **null**.
PARTITION_METHOD	varchar(18)	Partitioning policy. If a partition is not a level-1 partition, the value is **null**. • **r**: range partition. • **l**: list partition. • **h**: hash partition.
SUBPARTITION_METHOD	varchar(12)	Level-2 partitioning policy. If a partition is not a level-2 partition, the value is **null**. • **r**: range partition. • **l**: list partition. • **h**: hash partition.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
PARTITION_EXPRESSION	longtext	Not supported. The value is **NULL**.
SUBPARTITION_EXPRESSION	longtext	Not supported. The value is **NULL**.
PARTITION_DESCRIPTION	longtext	• For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
TABLE_ROWS	bigint unsigned	Number of table rows in the partition.
AVG_ROW_LENGTH	bigint unsigned	Not supported. The value is **NULL**.
DATA_LENGTH	bigint unsigned	Not supported. The value is **NULL**.
MAX_DATA_LENGTH	bigint unsigned	Not supported. The value is **NULL**.
INDEX_LENGTH	bigint unsigned	Not supported. The value is **NULL**.
DATA_FREE	bigint unsigned	Not supported. The value is **NULL**.
CREATE_TIME	datetime	Not supported. The value is **NULL**.
UPDATE_TIME	datetime	Not supported. The value is **NULL**.
CHECK_TIME	datetime	Not supported. The value is **NULL**.
CHECKSUM	bigint unsigned	Not supported. The value is **NULL**.
PARTITION_COMMENT	varchar(80)	Not supported. The value is **NULL**.
NODEGROUP	varchar(12)	Not supported. The value is **NULL**.
TABLESPACE_NAME	varchar(64)	Name of the tablespace to which the partition belongs.

6.1.15 plugins

The plug-ins view provides information about server plug-ins. For details, see **Table 6-17**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-17 information_schema.plugins columns

Name	Type	Description
PLUGIN_NAME	varchar(64)	Name used to reference a plug-in in statements, such as install plug-in and uninstall plug-in.
PLUGIN_VERSION	varchar(20)	Version from the plug-in's general type descriptor.
PLUGIN_STATUS	varchar(10)	Plug-in status. • **active**: installed • **inactive**: not installed
PLUGIN_TYPE	varchar(80)	This column is not supported in the current version, and the value is **null**.
PLUGIN_TYPE_VERSION	varchar(20)	This column is not supported in the current version, and the value is **null**.
PLUGIN_LIBRARY	varchar(64)	This column is not supported in the current version, and the value is **null**.
PLUGIN_LIBRARY_VERSION	varchar(20)	This column is not supported in the current version, and the value is **null**.
PLUGIN_AUTHOR	varchar(64)	This column is not supported in the current version, and the value is **null**.
PLUGIN_DESCRIPTION	longtext	Description of the plug-in.
PLUGIN_LICENSE	varchar(80)	This column is not supported in the current version, and the value is **null**.
LOAD_OPTION	varchar(64)	This column is not supported in the current version, and the value is **null**.

6.1.16 processlist

The processlist view displays information about the operations currently being performed by the set of threads executing within the server. For details, see **Table 6-18**. This view is read-only. All users have the read permission on this view.

Table 6-18 information_schema.processlist columns

Name	Type	Description
ID	bigint	Identifier of a thread. If the GUC parameter **enable_thread_pool** is set to **on** and the backend is idle, the value is **0**.
USER	varchar(32)	User who delivered the statement.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
HOST	varchar(64)	Host name and port number of the client that issues the statement. This parameter is displayed only when you log in through a remote client. In other cases, the value is **NULL**.
DB	varchar(64)	Name of the current M-compatible database.
COMMAND	varchar(16)	Type of the command the thread is executing on behalf of the client. The value is **sleep** if the thread is idle. This is not supported in this version, and the value is **null**.
TIME	integer	Duration that the thread has been in its current state, in seconds.
STATE	varchar(64)	Action, event, or state that indicates what the thread is doing. This column is not supported in the current version, and the value is **NULL**.
INFO	longtext	Statement the thread is executing, or **NULL** if it is executing no statement.

6.1.17 profiling

The profiling view provides statement profiling information. For details, see **Table 6-19**. This view is read-only. All users have the read permission on this view. In the current version, this view is not implemented, and all column values in the view are **NULL**.

Table 6-19 information_schema.profiling columns

Name	Type	Description
QUERY_ID	integer	Statement identifier. This column is not supported in the current version, and the value is **NULL**.
SEQ	integer	Sequence number indicating the display order for rows with the same **query_id** value. This column is not supported in the current version, and the value is **NULL**.
STATE	varchar(30)	Status of each analysis step. This column is not supported in the current version, and the value is **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
DURATION	decimal(9,6)	Duration of statement execution in the given state, in seconds. This column is not supported in the current version, and the value is **NULL**.
CPU_USER	decimal(9,6)	CPU usage of a user, in seconds. This column is not supported in the current version, and the value is **NULL**.
CPU_SYSTEM	decimal(9,6)	CPU usage of the system, in seconds. This column is not supported in the current version, and the value is **NULL**.
CONTEXT_VOLUNTARY	integer	Number of voluntary context switches. This column is not supported in the current version, and the value is **NULL**.
CONTEXT_INVOLUNTARY	integer	Number of involuntary context switches. This column is not supported in the current version, and the value is **NULL**.
BLOCK_OPS_IN	integer	Number of block input operations. This column is not supported in the current version, and the value is **NULL**.
BLOCK_OPS_OUT	integer	Number of block output operations. This column is not supported in the current version, and the value is **NULL**.
MESSAGES_SENT	integer	Number of communication messages sent. This column is not supported in the current version, and the value is **NULL**.
MESSAGES_RECEIVED	integer	Number of communication messages received. This column is not supported in the current version, and the value is **NULL**.
PAGE_FAULTS_MAJOR	integer	Number of major page faults. This column is not supported in the current version, and the value is **NULL**.
PAGE_FAULTS_MINOR	integer	Number of minor page faults. This column is not supported in the current version, and the value is **NULL**.
SWAPS	integer	Number of swaps. This column is not supported in the current version, and the value is **NULL**.
SOURCE_FUNCTION	varchar(30)	Information indicating where in the source code the profiled state executes. This column is not supported in the current version, and the value is **NULL**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
SOURCE_FILE	varchar(20)	Information indicating where in the source code the profiled state executes. This column is not supported in the current version, and the value is **NULL**.
SOURCE_LINE	integer	Information indicating where in the source code the profiled state executes. This column is not supported in the current version, and the value is **NULL**.

6.1.18 referential_constraints

The referential_constraints view provides information about foreign keys. For details, see **Table 6-20**. This view is read-only. All users have the read permission on this view.

Table 6-20 information_schema.referential_constraints columns

Name	Type	Description
CONSTRAINT_CATALOG	varchar(512)	Name of the database to which the constraint belongs. The value is the database to which the constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_SCHEMA	varchar(64)	Name of the schema to which the constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_NAME	varchar(64)	Name of the constraint.
UNIQUE_CONSTRAINT_CATALOG	varchar(512)	Name of the catalog containing the UNIQUE constraint that the constraint references. The value is the database to which the UNIQUE constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
UNIQUE_CONSTRAINT_SCHEMA	varchar(64)	Name of the schema containing the UNIQUE constraint that the constraint references. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
UNIQUE_CONSTRAINT_NAME	varchar(64)	Name of the UNIQUE constraint that the constraint references. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
MATCH_OPTION	varchar(64)	Value of the **match** constraint attribute.
UPDATE_RULE	varchar(64)	Value of the **on update** constraint attribute. The value can be **cascade**, **set null**, **set default**, **restrict**, or **no action**.
DELETE_RULE	varchar(64)	Value of the **on delete** constraint attribute. The value can be **cascade**, **set null**, **set default**, **restrict**, or **no action**.
TABLE_NAME	varchar(64)	Name of the table on which the constraint is defined. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
REFERENCED_TABLE_NAME	varchar(64)	Name of the table referenced by the constraint. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.

6.1.19 routines

The routines view provides information about stored procedures and stored functions. For details, see **Table 6-21**. This view is read-only. All users have the read permission on this view.

Table 6-21 information_schema.routines columns

Name	Type	Description
SPECIFIC_NAME	varchar(64)	Name of the routine.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
ROUTINE_CATALOG	varchar(512)	Name of the catalog to which the routine belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
ROUTINE_SCHEMA	varchar(64)	Name of the schema to which the routine belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
ROUTINE_NAME	varchar(64)	Name of the routine.
ROUTINE_TYPE	varchar(9)	• **procedure**: stored procedure • **function**: stored function
DATA_TYPE	varchar(64)	Return value data type if the routine is a stored function. If the routine is a stored procedure, it is **void**.
CHARACTER_MAXIMUM_LENGTH	integer	Maximum string length for stored function string return values, in characters. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
CHARACTER_OCTET_LENGTH	integer	Maximum string length for stored function string return values, in bytes. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
NUMERIC_PRECISION	bigint unsigned	Numeric precision for stored function numeric return values. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
NUMERIC_SCALE	integer	Numeric scale for stored function numeric return values. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
DATETIME_PRECISION	bigint unsigned	Fractional seconds precision for stored function temporal return values. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
CHARACTER_SET_NAME	varchar(64)	Character set name for stored function character string return values. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
COLLATION_NAME	varchar(64)	Collation name for stored function character string return values. If the routine is a stored procedure, the value is **null**. This is not supported in this version, and the value is **null**.
DTD_IDENTIFIER	longtext	Return value data type if the routine is a stored function. If the routine is a stored procedure, it is **void**.
ROUTINE_BODY	varchar(8)	Language used for the routine definition. It is always SQL.
ROUTINE_DEFINITION	longtext	Text of the SQL statement executed by the routine.
EXTERNAL_NAME	varchar(64)	The value is always **null**.
EXTERNAL_LANGUAGE	varchar(64)	Language of the stored routine.
PARAMETER_STYLE	varchar(8)	It is always SQL.
IS_DETERMINISTIC	varchar(3)	**yes** or **no**, depending on whether the routine is defined with the DETERMINISTIC characteristic.
SQL_DATA_ACCESS	varchar(64)	Data access characteristic for the routine.
SQL_PATH	varchar(64)	The value is always **null**.
SECURITY_TYPE	varchar(7)	Security verification mode used when a view queries data. The value is **definer** or **invoker**.
CREATED	datetime	Date and time when the routine is created. The value is **null**.
LAST_ALTERED	datetime	Date and time when the routine is last modified. The value is **null**.
SQL_MODE	varchar(8192)	SQL mode in effect when the routine is created or altered. The value is **null**.
ROUTINE_COMMENT	longtext	Comment of the routine. The value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
DEFINER	varchar(93)	Account named in the DEFINER clause (often the user who created the routine). The value is **null**.
CHARACTER_SET_CLIENT	varchar(32)	Session value of a system variable when the routine is created. The value is **null**.
COLLATION_CONNECTION	varchar(32)	Session value of a system variable when the routine is created. The value is null.
DATABASE_COLLATION	varchar(32)	Collation of the database with which the routine is associated. The value is null.

6.1.20 schemata

The schemata view provides information about databases. For details, see **Table 6-22**. This view is read-only. All users have the read permission on this view.

Table 6-22 information_schema.schemata columns

Name	Type	Description
CATALOG_NAME	varchar(512)	Name of the database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
SCHEMA_NAME	varchar(64)	Name of the schema. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
DEFAULT_CHARACTER_SET_NAME	varchar(32)	Name of the default character set of the schema.
DEFAULT_COLLATION_NAME	varchar(32)	Default collation of the schema.
SQL_PATH	varchar(32)	It is always **null**.

6.1.21 schema_privileges

Displays permissions on schemas in GaussDB. For details, see **Table 6-23**. This view is read-only. All users have the read permission on this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-23 information_schema.schema_privileges columns

Name	Type	Description
GRANTEE	varchar(81)	Name of the user or role to which the permission is granted.
TABLE_CATALOG	varchar(512)	Name of the catalog to which the object belongs.
TABLE_SCHEMA	varchar(64)	Schema name of the object.
PRIVILEGE_TYPE	varchar(64)	Permission granted.
IS_GRANTABLE	varchar(3)	Specifies whether to grant permissions. The options include: • yes • no

6.1.22 session_status

The session_status view displays the status of the currently connected servers. For details, see **Table 6-24**. This view is read-only. All users have the read permission on this view.

NOTE

Currently, only **Threads_connected** and **Uptime** are supported, which are used to display the number of session database connections and running time, respectively.

Table 6-24 information_schema.session_status columns

Name	Type	Description
VARIABLE_NAME	varchar(64)	Variable name of the server system status information of the current session.
VARIABLE_VALUE	varchar(1024)	Variable value of the server system status information of the current session.

6.1.23 session_variables

The session_variables view displays the system variable values that are valid for the current connection. For details, see **Table 6-25**. This view is read-only. All users have the read permission on this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-25 information_schema.session_variables columns

Name	Type	Description
VARIABLE_NAME	varchar(64)	Name of a variable.
VARIABLE_VALUE	varchar(1024)	Value of the variable.

6.1.24 statistics

The statistics view provides information about complete columns of table indexes. For details, see **Table 6-26**. This view is read-only. All users have the read permission on this view. Some information in the view is obtained based on statistics. Run **ANALYZE** before checking the information. (If data is updated in the database, you are advised to delay running **ANALYZE**.) If an index column is not a complete column in the table, the index column is not recorded in this view.

Table 6-26 information_schema.statistics columns

Name	Type	Description
TABLE_CATALOG	varchar(512)	Name of the database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the table containing the index belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table containing the index. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
NON_UNIQUE	bigint	Specifies whether the index is a non-unique index. If the index is unique, the value is **0**; otherwise, the value is **1**.
INDEX_SCHEMA	varchar(64)	Name of the schema to which the index belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
INDEX_NAME	varchar(64)	Name of the index. If the index is the primary key, the name is always **PRIMARY**. Otherwise, the name is the actual index name.
SEQ_IN_INDEX	bigint	Sequence number of an index column in an index.
COLUMN_NAME	varchar(64)	Name of an index column.
COLLATION	varchar(1)	Sorting mode of a column in an index. ● **A** (ascending order) ● **D** (descending order)
CARDINALITY	bigint	Estimate of the number of unique values in the index.
SUB_PART	bigint	This column is not supported in the current version, and the value is **null**.
PACKED	varchar(10)	This column is not supported in the current version, and the value is **null**.
NULLABLE	varchar(3)	Specifies whether the index column can contain null values. ● **yes**: It can contain null. ● **''**: It cannot contain null.
INDEX_TYPE	varchar(16)	Index method used, such as **BTREE** and **UBTREE**.
COMMENT	varchar(16)	This column is not supported in the current version, and the value is **null**.
INDEX_COMMENT	varchar(1024)	Comment of an index.

Run **ANALYZE** to update the statistics and then check the view.

```
m_regression=# CREATE TABLE t1(a int) DISTRIBUTE BY hash(a);
CREATE TABLE
m_regression=# INSERT INTO t1 VALUES (1),(2);
INSERT 0 2
m_regression=# CREATE INDEX t1_index_1 ON t1(a);
CREATE INDEX
m_regression=# ANALYZE;
ANALYZE
m_regression=# select table_catalog,table_name,index_name,index_type, cardinality from
information_schema.statistics where table_name='t1';
 table_catalog | table_name | index_name | index_type | cardinality
---------------+------------+------------+------------+-------------
 m_regression  | t1         | t1_index_1 | UBTREE     |           2
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6.1.25 tables

The tables view provides information about tables in databases. For details, see Table 6-27. This view is read-only. All users have the read permission on this view. Some information in the view is obtained based on statistics. Run **ANALYZE** before checking the information. (If data is updated in the database, you are advised to delay running **ANALYZE**.)

Table 6-27 information_schema.tables columns

Name	Type	Description
TABLE_CATALOG	varchar(512)	Name of the database. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema where the table is located. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_TYPE	varchar(64)	Type of the table. **BASE TABLE** for a table, **VIEW** for a view, and **SYSTEM VIEW** for a table or view under the INFORMATION_SCHEMA table.
ENGINE	varchar(64)	Storage engine of the table. This column is not supported in the current version, and the value is **null**.
VERSION	bigint unsigned	This column is not supported in the current version, and the value is **null**.
ROW_FORMAT	varchar(10)	This column is not supported in the current version, and the value is **null**.
TABLE_ROWS	bigint unsigned	If the table specifies a view, the value is **null**. Otherwise, the value is the number of live tuples.
AVG_ROW_LENGTH	bigint unsigned	If there is no live or dead tuple in the table, the value is **null**. Otherwise, the value is the average row length of the table.
DATA_LENGTH	bigint unsigned	If the table specifies a view, the value is **null**. Otherwise, the value is the data file length.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
MAX_DATA_LENGTH	bigint unsigned	This column is not supported in the current version, and the value is **null**.
INDEX_LENGTH	bigint unsigned	If the table specifies a view, the value is **null**. Otherwise, the value is the size of the index file.
DATA_FREE	bigint unsigned	If there is no live or dead tuple in the table, the value is **null**. Otherwise, the value is the number of bytes that have been allocated but not used.
AUTO_INCREMENT	bigint unsigned	Next automatic increment. If this parameter is not set, the value is **null**. This field is not supported in the current version and its value is null.
CREATE_TIME	datetime	Time when the table is created. If a view is created, the actual table creation time is displayed.
UPDATE_TIME	datetime	Time when the table data was updated last time. The value is **null** for a table or view provided by the M-compatible database.
CHECK_TIME	datetime	This column is not supported in the current version, and the value is null.
TABLE_COLLATION	varchar(32)	Collation of the table.
CHECKSUM	bigint unsigned	This column is not supported in the current version, and the value is **null**.
CREATE_OPTIONS	varhcar(255)	Options of the table.
TABLE_COMMENT	varchar(2048)	Comment of the table.

Run **ANALYZE** to update the statistics and then check the view.

```
m_db=#  CREATE TABLE t1(a int);
CREATE TABLE
m_db=# INSERT INTO t1 VALUES (1),(2);
INSERT 0 2
m_db=# ANALYZE;
ANALYZE
m_db=# SELECT TABLE_CATALOG,TABLE_SCHEMA,TABLE_NAME,TABLE_TYPE,TABLE_ROWS,ENGINE FROM
information_schema.tables WHERE TABLE_NAME='t1';
 TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME | TABLE_TYPE | TABLE_ROWS | ENGINE
---------------+--------------+------------+------------+------------+--------
 m_db          | public       | t1         | BASE TABLE | 2          | Ustore
(1 row)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
m_db=# DROP TABLE t1;
DROP TABLE
```

6.1.26 table_constraints

The table_constraints view describes which tables have constraints. For details, see **Table 6-28**. This view is read-only. All users have the read permission on this view.

Table 6-28 information_schema. table_constraints columns

Name	Type	Description
CONSTRAINT_CATALOG	varchar(512)	Name of the database to which the constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_SCHEMA	varchar(64)	Name of the schema to which the constraint belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_NAME	varchar(64)	Name of the constraint. If the constraint is a primary key constraint, **PRIMARY** is displayed.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the table belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the table. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
CONSTRAINT_TYPE	varchar(64)	Type of the constraint. The value can be **unique**, **primary key**, **foreign key**, or **check**.

6.1.27 table_privileges

The table_privileges view provides information about table permissions. For details, see **Table 6-29**. This view is read-only. All users have the read permission on this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 6-29 information_schema.table_privileges columns

Name	Type	Description
GRANTEE	varchar(81)	Name of the account to which the permission is granted.
TABLE_CATALOG	varchar(512)	Name of the catalog to which the table belongs.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the table belongs.
TABLE_NAME	varchar(64)	Name of the table.
PRIVILEGE_TYPE	varchar(64)	Permission granted. The value is any permission that can be granted at the table level.
IS_GRANTABLE	varchar(3)	If the user has the GRANT OPTION permission, the value is **yes**; otherwise, the value is **no**.

6.1.28 triggers

The triggers view provides information about triggers. For details, see **Table 6-30**. This view is read-only. All users have the read permission on this view.

Table 6-30 information_schema.triggers columns

Name	Type	Description
TRIGGER_CATALOG	varchar(512)	Name of the catalog to which the view belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TRIGGER_SCHEMA	varchar(64)	Name of the schema to which the trigger belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TRIGGER_NAME	varchar(64)	Name of the trigger.
EVENT_MANIPULATION	varchar(6)	Trigger event. The value is **INSERT** (inserts a row), **DELETE** (deletes a row), or **UPDATE** (modifies a row).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
EVENT_OBJECT_CATALOG	varchar(512)	Catalog where the trigger association table is located. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
EVENT_OBJECT_SCHEMA	varchar(64)	Schema of the trigger association table. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
EVENT_OBJECT_TABLE	varchar(64)	Name of the trigger association table. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
ACTION_ORDER	bigint	Ordinal position of the event_manipulation trigger's action within the list of triggers on the same table with the same **action_timing**. This column is not supported in the current version, and the value is **null**.
ACTION_CONDITION	longtext	The value is always **null**.
ACTION_STATEMENT	longtext	Trigger body, that is, the statement executed when the trigger activates. This text is encoded using UTF-8.
ACTION_ORIENTATION	varchar(9)	The value is always **row**.
ACTION_TIMING	varchar(6)	Specifies whether the trigger activates before or after the trigger event. The value is **before** or **after**.
ACTION_REFERENCE_OLD_TABLE	varchar(64)	The value is always **null**.
ACTION_REFERENCE_NEW_TABLE	varchar(64)	The value is always **null**.
ACTION_REFERENCE_OLD_ROW	varchar(3)	Identifier of an old column. The value is always **old**.
ACTION_REFERENCE_NEW_ROW	varchar(3)	Identifier of a new column. The value is always **new**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
CREATED	datetime	Date and time when the trigger is created. This is a TIMESTAMP(2) value (with a fractional part in hundredths of seconds) for triggers. This column is not supported in the current version, and the value is **null**.
SQL_MODE	varchar(8192)	SQL mode in effect when the trigger is created, and under which the trigger executes. This column is not supported in the current version, and the value is **null**.
DEFINER	varchar(93)	User who creates the trigger. This column is not supported in the current version, and the value is **null**.
CHARACTER_SET_CLIENT	varchar(32)	Session value of the *character_set_client* system variable when the trigger is created. This column is not supported in the current version, and the value is **null**.
COLLATION_CONNECTION	varchar(32)	Session value of the **collation_connection** system variable when the trigger is created. This column is not supported in the current version, and the value is **null**.
DATABASE_COLLATION	varchar(32)	Collation of the database with which the trigger is associated. This column is not supported in the current version, and the value is **null**.

6.1.29 user_privileges

The user_privileges view displays user permissions on database objects such as tables, views, and schemas. For details, see **Table 6-31**. This view is read-only. All users have the read permission on this view. A user can view only the records that the user has the access permission.

Table 6-31 information_schema.user_privileges columns

Name	Type	Description
GRANTEE	varchar(81)	Name of the user or role to which the permission is granted.
TABLE_CATALOG	varchar(512)	Name of the catalog (database) where the object is located.
PRIVILEGE_TYPE	varchar(64)	Permission granted.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
IS_GRANTABLE	varchar(3)	• **yes**: The user has the permission to grant permissions. • **no**: The user has no permission to grant permissions.

6.1.30 views

The views view provides information about views in databases. For details, see **Table 6-32**.

Table 6-32 information_schema.views columns

Name	Type	Description
TABLE_CATALOG	varchar(512)	Name of the catalog (database) to which the view belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_SCHEMA	varchar(64)	Name of the schema to which the view belongs. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
TABLE_NAME	varchar(64)	Name of the view. When **lower_case_table_names** is set to **0**, the value of this column is case-sensitive. When **lower_case_table_names** is set to **1**, the value of this column is case-insensitive.
VIEW_DEFINITION	longtext	Statement that provides the definition of the view.
CHECK_OPTION	varchar(8)	Value of the **check_option** attribute. The value can be **NONE**, **CASCADE**, or **LOCAL**.
IS_UPDATABLE	varchar(3)	Specifies whether the view can be updated.
DEFINER	varchar(93)	This column is not supported in the current version, and the value is **null**.
SECURITY_TYPE	varchar(7)	This column is not supported in the current version, and the value is **null**.
CHARACTER_SET_CLIENT	varchar(32)	This column is not supported in the current version, and the value is **null**.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
COLLATION_CONNECTION	varchar(32)	This column is not supported in the current version, and the value is **null**.

6.2 m_schema

The m_schema schema encapsulates the new views table_priv, columns_priv, procs_priv, proc, and func.

6.2.1 tables_priv

The m_schema.tables_priv view displays database users' permissions on tables. For details, see **Table 6-33**.

Table 6-33 m_schema.tables_priv columns

Name	Type	Description
Host	char(60)	Name of the host.
Db	char(64)	Name of the database (schema).
User	char(32)	Name of the user.
Table_name	char(64)	Name of the table.
Grantor	char(93)	Name of the user who grants the permission.
Timestamp	timestamp	Current timestamp. This column is not supported in the current version, and the value is **null**.

NOTE

- If the value of **sql_mode** contains **ansi_quotes**, add double quotation marks when querying the **user** column.
  ```
  SELECT "user" FROM m_schema.tables_priv WHERE "user" = 'PUBLIC';
  ```
- If the value of **sql_mode** does not contain **ansi_quotes**, add backquotes when querying the **user** column.
  ```
  SELECT `user` FROM m_schema.tables_priv WHERE `user` = 'PUBLIC';
  ```
- In M-compatible databases of the current version, the **Table_priv** and **Column_priv** columns are not displayed in this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6.2.2 columns_priv

The m_schema.columns_priv view displays information about database users' permissions on columns. For details, see **Table 6-34**.

Table 6-34 m_schema.columns_priv columns

Name	Type	Description
Host	char(60)	Name of the host.
Db	char(64)	Name of the database (schema).
User	char(32)	Name of the user.
Table_name	char(64)	Name of the table.
Column_name	char(64)	Name of the column.
Timestamp	timestamp	Current timestamp. This column is not supported in the current version, and the value is **null**.

NOTE

- If the value of **sql_mode** contains **ansi_quotes**, add double quotation marks when querying the **user** column.
  ```
  SELECT "user" FROM m_schema.tables_priv WHERE "user" = 'PUBLIC';
  ```
- If the value of **sql_mode** does not contain **ansi_quotes**, add backquotes when querying the **user** column.
  ```
  SELECT `user` FROM m_schema.tables_priv WHERE `user` = 'PUBLIC';
  ```
- In M-compatible databases of the current version, the **Column_priv** column is not displayed in this view.

6.2.3 procs_priv

The m_schema.procs_priv view displays information about database users' permissions on stored procedures and functions. For details, see **Table 6-35**. This view is read-only. All users have the read permission on this view. A user can view only the records that the user has the access permission.

Table 6-35 m_schema.procs_priv columns

Name	Type	Description
Host	char(60)	Name of the host.
Db	char(64)	Name of the database (schema).

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
User	char(32)	Name of the user.
Routine_name	char(64)	Name of the routine.
Grantor	char(93)	Name of the user who grants the permission.
Timestamp	timestamp	Current timestamp. This column is not supported in the current version, and the value is **null**.

NOTE

- If the value of **sql_mode** contains **ansi_quotes**, add double quotation marks when querying the **user** column.
  ```
  SELECT "user" FROM m_schema.tables_priv WHERE "user" = 'PUBLIC';
  ```
- If the value of **sql_mode** does not contain **ansi_quotes**, add backquotes when querying the **user** column.
  ```
  SELECT `user` FROM m_schema.tables_priv WHERE `user` = 'PUBLIC';
  ```
- In M-compatible databases of the current version, the **Routine_type** and **Proc_priv** columns are not displayed in this view.

6.2.4 proc

The m_schema.proc view displays information about stored procedures of the database. For details, see **Table 6-36**. This view is read-only. All users have the read permission on this view. A user can view only the records that the user has the access permission.

Table 6-36 m_schema.proc columns

Name	Type	Description
db	char(64)	Name of the database (schema).
name	char(64)	Name of the routine.
specific_name	char(64)	Exact name of the routine.
param_list	blob	List of parameters.
returns	longblob	Return value type of the routine. The value is **array**, **user_defined**, or others.
body	longblob	Definition of the routine.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Name	Type	Description
definer	char(93)	User who defines the routine.
created	timestamp	Date and time when the routine is created. This column is not supported in the current version, and the value is **null**.
modified	timestamp	Date and time when the routine is modified. This column is not supported in the current version, and the value is **null**.
comment	text	This column is not supported in the current version, and the value is **null**.
character_set_client	char(32)	Character set used by the client that creates the routine.
collation_connection	char(32)	Collation (and character set) used to create routine connections. This column is not supported in the current version, and the value is **null**.
db_collation	char(32)	Default collation (and character set) of the database when the routine is created. This column is not supported in the current version, and the value is **null**.
body_utf8	longblob	Definition of the routine (UTF-8). This column is not supported in the current version, and the value is **null**.

NOTE

- In M-compatible databases of the current version, the **type**, **language**, **sql_data_access**, **is_deterministic**, **security_type**, and **sql_mode** columns are not displayed in this view.
- This view can be queried only on CNs.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

6.2.5 func

The m_schema.func view displays information about GaussDB functions.

This view is read-only. All users have the read permission on this view. A user can view only the records that the user has the access permission.

Table 6-37 m_schema.func columns

Name	Type	Description
name	char(64)	Name of a UDF.
ret	tinyint	Return value type. This column is not supported in the current version, and the value is **null**.
dl	char(128)	Name of the shared library.

NOTE

In the M-compatible database of the current version, the **type** column is not displayed in this view.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

7 Stored Procedures

Stored procedures are not supported in the current version and will be supported later.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.