GaussDB

Compatibility(Distributed)

Issue	01
Date	2025-08-22

HUAWEI

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 GaussDB Compatibility Overview

In GaussDB, you can create a database whose compatibility mode is set to **ORA**, **MYSQL**, **TD**, **PG**, or **M**, which represent the database is in Oracle-compatible, MySQL-compatible, Teradata-compatible, PostgreSQL-compatible, or M-compatible mode, respectively. The compatibility mode may affect SQL syntax, data types, system functions, and stored procedures. Some compatibility APIs are supported only in the corresponding compatibility mode.

For details about Oracle-compatible mode, M-compatible mode, and MySQL-compatible mode, see **Oracle Compatibility Description**, **M-compatible Mode**, and **MySQL-compatible Mode**. You can select a proper compatibility mode based on actual requirements.

2 Oracle Compatibility Description

2.1 Overview of Oracle Database Compatibility

GaussDB is generally compatible with Oracle Database in terms of basic functions (such as data types, SQL statements, and database objects) and PL/SQL. However, due to architecture design differences, there are still some incompatible items.

This chapter compares the Oracle-compatible mode in GaussDB 505.2.1 with Oracle Database 19c.

2.2 Basic SQL Elements

2.2.1 Data Types

Table 2-1 Numeric types

No.	Oracle Database	GaussDB	Difference
1	NUMBER [(p [, s])]	Supported, with differences.	The precision and usage are different. • When NUMBER contains parameters, the maximum boundary values of precision *p* and scale *s* in GaussDB are greater than those in Oracle Database. • In GaussDB, the default value of *p* when NUMBER does not contain parameters is much greater than the maximum boundary value when NUMBER contains parameters. However, in Oracle Database, the former is equal to the latter. • In GaussDB, the value of *s* cannot be negative. In Oracle Database, a negative *s* value is accurate to an integer.
2	FLOAT [(p)]	Supported.	-
3	BINARY_FLOAT	Not supported.	-
4	BINARY_DOUBLE	Supported.	-

Table 2-2 Date and time types

No.	Oracle Database	GaussDB	Difference
1	DATE	Supported, with differences.	The precision is different. GaussDB supports a wider time range than Oracle Database.
2	TIMESTAMP [(fractional_seconds_precision)]	Supported, with differences.	-

No.	Oracle Database	GaussDB	Difference
3	TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	Supported, with differences.	The timestamptz type of GaussDB is equivalent to the timestampwithlocaltimezone type of Oracle Database. The type corresponding to timestamptz of Oracle Database is missing. Time zone update: In some countries or regions, the time zone information is often updated. Therefore, the database system often needs to modify the time zone file accordingly to ensure that the time is correct. Currently, the GaussDB time zone type involves only timestamp with timezone. When a new time zone file takes effect, the existing data is not changed, and the new data is adjusted based on the time zone file information. Data capabilities of GaussDB are different from those of Oracle Database.
4	TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE	Not supported.	-
5	INTERVAL YEAR [(year_precision)] TO MONTH	Supported.	-
6	INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds_precision)]	Supported.	-

NOTE

- In ORA-compatible mode, the DATE type is replaced by TIMESTAMP(0) WITHOUT TIME ZONE. The differences between DATE and TIMESTAMP(0) WITHOUT TIME ZONE are the same.
- In terms of TIMESTAMP [(fractional_seconds_precision)] and TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE, the differences between GaussDB and Oracle Database are as follows:
  - The value of **fractional_seconds_precision** ranges from 0 to 6 in GaussDB, but ranges from 0 to 9 in Oracle Database.
  - GaussDB uses **DateStyle** to set the display format of date and time values and the rules of resolving ambiguous values. For details, see "SQL Reference > Data Type > Date/Time Types" in *Developer Guide*. Generally, the input format verification and output display in Oracle Database are controlled by the **NLS_TIMESTAMP_FORMAT** and **NLS_TIMESTAMP_TZ_FORMAT** parameters.
  - By default, GaussDB removes zeros from the end of the decimal part of the second. Oracle Database controls the display of the decimal part based on the setting (**FF/FF1-FF9**) of the formatting parameter. For example, '2017-09-01 10:32:19.212000' is displayed as '2017-09-01 10:32:19.212' in GaussDB. In Oracle Database, it is displayed as '2017-09-01 10:32:19.212' if the **format** parameter contains FF, or '2017-09-01 10:32:19.212000000' if the **format** parameter contains **FF9**.
  - GaussDB supports a wider time range than Oracle Database.

Table 2-3 Character types

No.	Oracle Database	GaussDB	Difference
1	VARCHAR2 (size [BYTE | CHAR])	Supported, with differences.	In GaussDB, the unit of **size** only supports **BYTE** and the option **CHAR** is not available. The maximum size is 10 MB. In Oracle Database, however, the unit of **size** can be selected between **BYTE** and **CHAR**. If **MAX_STRING_SIZE** is set to **EXTENDED**, the maximum size is 32767 bytes. If **MAX_STRING_SIZE** is set to **STANDARD**, the maximum size is 4000 bytes. The actual number of characters that can be contained depends on the character set in use.
2	NVARCHAR2 (size)	Supported, with differences.	In GaussDB, the unit of **size** is bytes, and the maximum size is 10 MB. The actual number of characters that can be contained depends on the character set in use. In Oracle Database, when **MAX_STRING_SIZE** is set to **EXTENDED**, the maximum length is 32767 bytes. When **MAX_STRING_SIZE** is set to **STANDARD**, the maximum length is 4000 bytes. The actual number of characters that can be contained depends on the character set in use.

No.	Oracle Database	GaussDB	Difference
3	CHAR [(size [BYTE | CHAR])]	Supported, with differences.	In GaussDB, the unit of **size** only supports **BYTE** and the option **CHAR** is not available. The maximum size is 10 MB. In Oracle Database, however, the unit of **size** can be selected between **BYTE** and **CHAR**. The maximum size is 2000 bytes. The actual number of characters that can be contained depends on the character set in use.
4	NCHAR [(size)]	Supported, with differences.	In GaussDB, the unit of **size** is bytes, and the maximum size is 10 MB. In Oracle Database, however, the unit of **size** is characters, and the maximum size is 2000 bytes. The actual number of characters that can be contained depends on the character set in use.
5	CLOB	Supported, with differences.	Locators are not supported.
6	NCLOB	Not supported.	-
7	LONG	Not supported.	-

Table 2-4 Binary types

No.	Oracle Database	GaussDB	Difference
1	RAW (size)	Supported, with differences.	In GaussDB, **size** indicates the recommended byte length and is not used to verify the byte length of the input raw type.
2	LONG RAW	Not supported.	-
3	BLOB	Supported.	-
4	BFILE	Not supported.	-

Table 2-5 ROWID types

No.	Oracle Database	GaussDB
1	ROWID	Not supported.
2	UROWID	Not supported.

Table 2-6 User-defined types

No.	Oracle Database	GaussDB
1	Object types	Not supported.
2	REF data types	Not supported.
3	Variable arrays	Supported.
4	Nested tables	Supported.

Table 2-7 Pseudo-types

No.	Oracle Database	GaussDB
1	anytype	Not supported.
2	anydata	Not supported.
3	anydataset	Not supported.

Table 2-8 XML types

No.	Oracle Database	GaussDB	Difference
1	XMLType	Supported, with differences.	GaussDB does not support some operations. For example, the XMLELEMENT function is used to convert a character string to the XML type instead of the XMLType type. For details, see "SQL Reference > Data Type > XMLType" in *Developer Guide*.
2	URIType	Not supported.	-

Table 2-9 Spatial types

No.	Oracle Database	GaussDB
1	SDO_GEOMETRY	Not supported.
2	SDO_TOPO_GEOMETRY	Not supported.
3	SDO_GEORASTER	Not supported.

Table 2-10 Lock modes

No.	Oracle Database	GaussDB
1	none	-
2	null	AccessShare
3	RS	RowShare
4	RX	RowExclusive
5	S	ShareUpdateExclusive
6	SRX	Share
7	-	ShareRowExclusive
8	X	Exclusive
9	-	AccessExclusive
10	-	INVALID **NOTE** INVALID of GaussDB indicates that an invalid lock is assigned. An invalid lock is assigned only when a lock that cannot be identified by GaussDB occurs during system running.

2.2.2 Data Type Comparison Rules

Data type comparison (collation) rules apply only when values of the same data type are compared (collated).

Table 2-11 Comparison rules

No.	Oracle Database	GaussDB	Difference
1	Numeric values	Supported.	-

No.	Oracle Database	GaussDB	Difference
2	Datetime values	Supported.	-
3	Binary values	Supported.	-
4	Character values	Supported, with differences.	• GaussDB and Oracle Database support different comparison rules, and the names of the same comparison rules may be different. • GaussDB and Oracle Database differ in specifying comparison rules. For example, table-level comparison rules cannot be specified in GaussDB, but can be specified in Oracle Database. • GaussDB and Oracle Database differ in the syntax for specifying comparison rules. For example, in GaussDB, the **ENCODING**, **LC_CTYPE**, and **LC_COLLATE** parameters are used to specify the character set, character type, and comparison rules used during database creation. For details, see "SQL Reference > SQL Syntax > C > CREATE DATABASE" in *Developer Guide*. In Oracle Database, comparison rules at different levels are usually specified by a series of parameters with the NLS prefix.
5	Object values	Not supported.	-
6	Varrays and nested tables	Supported, with differences.	Both GaussDB and Oracle Database support the comparison of varrays. Different from Oracle Database, GaussDB not only supports the comparison of the number of elements in two varrays, but also supports the comparison between varrays of the same type.
7	Data type precedence	Supported.	-
8	Explicit/ Implicit data conversion	Supported.	-

2.2.3 Literals

Table 2-12 Literals

No.	Oracle Database	GaussDB
1	Text literals	Supported.
2	Numeric literals	Supported.
3	Datetime literals	Supported.
4	Interval literals	Supported.

2.2.4 Format Models

Table 2-13 Formats

No.	Oracle Database	GaussDB	Difference
1	Number formats	Supported, with differences.	GaussDB supports the $, C, TM, TM9, TME, and U formats only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**. In addition, this parameter does not support the TH, PL, or SG format. For details about GaussDB, see Table **Formats for the number type** in "SQL Reference > Functions and Operators > Type Conversion Functions" in *Developer Guide*.
2	Datetime formats	Supported, with differences.	GaussDB: Parameters used for time truncation and rounding are valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**. For details about GaussDB support, see formats for formatting date and time in "SQL References > Functions and Operators > Date and Time Processing Functions and Operators" in *Developer Guide*.
3	Format model modifiers	Supported.	-

No.	Oracle Database	GaussDB	Difference
4	String-to-date conversion rules	Supported, with differences.	GaussDB: The to_timestamp_tz function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**. For details about GaussDB, see to_date, to_timestamp, and to_timestamp_tz in "SQL Reference > Functions and Operators > Type Conversion Functions" in *Developer Guide*.
5	XML format models	Not supported.	-

2.2.5 Nulls

Table 2-14 Nulls

No.	Oracle Database	GaussDB
1	IS NULL and IS NOT NULL	Supported.
2	NULLS in conditions	Supported.

2.2.6 Comments

Table 2-15 Comments

No.	Oracle Database	GaussDB	Difference
1	A slash and an asterisk (/*)	Supported.	-
2	Two hyphens (--)	Supported.	-
3	**COMMENT** command	Supported.	-
4	HINT	Supported, with differences.	GaussDB does not support the '--+' hint format. For details, see "SQL Optimization > Hint-based Tuning" in *Developer Guide*.

2.2.7 Database Objects

Table 2-16 Schema objects

No.	Oracle Database	GaussDB	Difference
1	Analytic views	Not supported.	-
2	Attribute dimensions	Not supported.	-
3	Clusters	Supported.	-
4	Constraints	Supported.	-
5	Database links	Supported.	-
6	Database triggers	Supported.	-
7	Dimensions	Supported.	-
8	External procedure libraries	Not supported.	-
9	Hierarchies	Not supported.	-
10	Index-organized tables	Not supported.	-
11	Indexes	Supported.	-
12	Index types	Not supported.	-
13	Java classes	Not supported.	-
14	Java resources	Not supported.	-
15	Java source code	Not supported.	-
16	Join groups	Not supported.	-

No.	Oracle Database	GaussDB	Difference
17	Materialized views	Supported.	-
18	Materialized view logs	Not supported.	-
19	Mining models	Not supported.	-
20	Object tables	Not supported.	-
21	Object types	Not supported.	-
22	Object views	Not supported.	-
23	Operators	Supported.	-
24	Packages	Supported.	-
25	Sequences	Supported.	-
26	Storage functions	Supported.	-
27	Stored procedures	Supported.	-
28	Synonyms	Supported, with differences.	The names of Oracle Database objects in the same namespace must be unique. In GaussDB, the name of a synonym can be the same as that of a table, view, function, or package in the same namespace. In this case, GaussDB preferentially accesses the table, view, function, or package object with the same name. If no such object is found, GaussDB searches for the synonym object. In addition, the PUBLIC synonym is searched only when the schema name of the object to which the synonym points is a username. For details about the search sequence, see "SQL Reference > SQL Syntax > C > CREATE SYNONYM" in *Developer Guide*.
29	Tables	Supported.	-

No.	Oracle Database	GaussDB	Difference
30	Views	Supported.	-
31	Zone map	Not supported.	-

Table 2-17 Non-schema objects

No.	Oracle Database	GaussDB
1	Contexts	Not supported.
2	Directories	Supported.
3	Editions	Not supported.
4	Flashback archives	Not supported.
5	Lockdown profiles	Not supported.
6	Profiles	Not supported.
7	Restore points	Supported.
8	Roles	Supported.
9	Rollback segments	• Ustore supports rollback segments. • Astore does not support rollback segments.
10	Tablespaces	Supported.
11	Tablespace sets	Not supported.
12	Unified audit policies	Supported.
13	Users	Supported.

2.2.8 Database Object Names and Qualifiers

Table 2-18 Naming rules

No.	Oracle Database	GaussDB	Difference
1	Database object naming rules	Supported, with differences.	GaussDB uses lowercase letters by default.

No.	Oracle Database	GaussDB	Difference
2	Schema object naming rules	Supported.	-

2.2.9 Syntax for Schema Objects and Parts in SQL Statements

Table 2-19 Object reference

No.	Oracle Database	GaussDB
1	General syntax for referencing an object	Supported.
2	Resolving a reference to an object	Supported.
3	Referencing objects in other schemas	Supported.
4	Referencing objects in remote databases	Supported.
5	Referencing partitions and subpartitions of tables and indexes	Supported.

2.3 Pseudocolumns

GaussDB is compatible with sequence and rownum pseudocolumns. Other pseudocolumns are not supported.

Hierarchical Query Pseudocolumns

Table 2-20 Hierarchical query pseudocolumns

No.	Oracle Database	GaussDB
1	connect_by_iscycle	Supported.
2	connect_by_isleaf	Supported.
3	LEVEL pseudocolumn	Supported.

Sequence Pseudocolumns

Table 2-21 Sequences

No.	Oracle Database	GaussDB	Difference
1	currval	Supported, with differences.	It is implemented as a function in GaussDB. The call mode is compatible with Oracle Database.
2	nextval	Supported, with differences.	It is implemented as a function in GaussDB. The call mode is compatible with Oracle Database.

ROWNUM Pseudocolumn

Table 2-22 rownum

No.	Oracle Database	GaussDB	Difference
1	rownum	Supported, with differences.	When Oracle Database uses rownum in the left, right, and full join conditions for filtering, the performance varies according to the conditions. The rownum condition may be ignored or partially ignored. However, GaussDB filters the results after left, right, and full join.

XMLDATA Pseudocolumn

Table 2-23 xmldata

No.	Oracle Database	GaussDB
1	xmldata	Not supported.

2.4 Operators

GaussDB is compatible with operators except hierarchical query.

SQL Operators

Table 2-24 SQL operators

No.	Oracle Database	GaussDB
1	Unary and binary operators	Supported.
2	Operator precedence	Supported.

Arithmetic Operators

Table 2-25 Arithmetic operators

No.	Oracle Database	GaussDB
1	Unary operators: positive (+) and negative (–).	Supported.
2	Binary operators: addition (+) and subtraction (–).	Supported.
3	Binary operators: multiplication (*) and division (/).	Supported.

COLLATE Operator

Table 2-26 COLLATE operator

No.	Oracle Database	GaussDB
1	COLLATE collation_name	Supported.

Connection Operators

Table 2-27 Connection operators

No.	Oracle Database	GaussDB
1	||	Supported.

Hierarchical Query Operators

Table 2-28 Hierarchical query operators

No.	Oracle Database	GaussDB	Difference
1	prior	Supported, with differences.	GaussDB: Only ordinary columns can be called. Functions cannot be called.
2	connect_by_root	Supported, with differences.	GaussDB: When connect_by_root is called, if parentheses are used to modify the operation value, the behavior is the same as that of Oracle Database. If parentheses are not used, this operator can be called only for ordinary columns.

Set Operators

Table 2-29 Set operators

No.	Oracle Database	GaussDB
1	union	Supported.
2	union all	Supported.
3	intersect	Supported.
4	minus	Supported.

Multiset Operators

Table 2-30 Multiset operators

No.	Oracle Database	GaussDB
1	multiset except	Supported.
2	multiset intersect	Supported.
3	multiset union	Supported.

User-defined Operators

Table 2-31 User-defined operators

No.	Oracle Database	GaussDB	Difference
1	CREATE OPERATOR	Supported.	• Oracle Database provides CONTEXT_CLAUSE to define functional estimator functions, which is different from the restriction selectivity estimator function in GaussDB. GaussDB does not support user-defined functional estimator functions. • Optional parameters in GaussDB differ greatly from those in Oracle Database. For details, see the GaussDB parameter description in "SQL Reference > SQL Syntax > C > CREATE OPERATOR" in *Developer Guide*.

Comparison Operators

No.	Oracle Database	GaussDB
1	< =	Supported.
2	< >	Supported.
3	> =	Supported.
4	^ =	Supported.
5	! =	Not supported. For !=, if there is a space between an exclamation mark (!) and an equal sign (=), the exclamation mark will be identified as factorial.

For comparison operators <=, <>, >=, and ^=, if there is a space between two symbols, it does not affect normal operations. For !=, if there is a space between an exclamation mark (!) and an equal sign (=), the exclamation mark will be identified as factorial, which may cause the result to be inconsistent with the expected result.

2.5 Expressions

GaussDB is compatible with most database expressions.

Table 2-32 Expressions

No.	Oracle Database	GaussDB	Difference
1	Simple expressions	Supported.	-
2	Analytic view expressions	Not supported.	-
3	Compound expressions	Supported.	-
4	CASE expressions	Supported.	-
5	Column expressions	Supported.	-
6	CURSOR expressions	Not supported.	-
7	Datetime expressions	Supported, with differences.	GaussDB command output does not contain time zone information, but Oracle Database contains time zone information similar to "PM AMERICA/LOS_ANGELES."
8	Function expressions	Supported.	-
9	Interval expressions	Partially supported.	GaussDB supports statements in the format of "SELECT INTERVAL '999999999 23:59:59.999' day(9) to second FROM DUAL;" but does not support statements in the format of "SELECT(SYSDATE-SYSDATE) DAY TO SECOND FROM DUAL;". They are supported in Oracle Database.
10	JSON object access expressions	Partially supported, with differences.	• GaussDB can extract values from JSON objects in "->'key'" mode, while Oracle Database can extract values in ".key" mode. • For JSONARRY objects, Oracle Database can extract values corresponding to all keys at a time in ".key" mode. However, GaussDB does not support this function.
11	Model expressions	Not supported.	-

No.	Oracle Database	GaussDB	Difference
12	Object expressions	Not supported.	-
13	Placeholder expressions	Partially supported.	GaussDB supports general placeholder expressions such as ":var", but does not support the combination of two general placeholder expressions using the INDICATOR keyword.
14	Scalar subquery expressions	Supported.	-
15	Type constructor expressions	Partially supported.	GaussDB cannot specify the NEW keyword before the type constructor, but Oracle Database can.
16	Expression lists	Supported.	-

2.6 Conditions

This chapter describes common compatible conditions. The conditions include comparison, floating-point, logical, model, multiset, pattern matching, **NULL** value, XML, SQL/JSON, composite, BETWEEN, EXISTS, IN, and IS OF TYPE, as shown in **Table 2-33**.

Table 2-33 Conditions

No.	Oracle Database	GaussDB	Difference
1	Comparison conditions	Supported, with differences.	Differences exist when statements contain the ANY, SOME, and ALL operators. Oracle Database supports operations on list objects, but GaussDB needs to convert list objects into array expressions before performing operations.
2	Floating-point conditions	Not supported.	-
3	Logical conditions	Supported.	-
4	Model conditions	Not supported.	-

No.	Oracle Database	GaussDB	Difference
5	Multiset conditions	Not supported.	-
6	Pattern-matching conditions	Supported.	-
7	NULL conditions	Supported.	-
8	XML conditions	Not supported.	-
9	SQL/JSON conditions	Partially supported, with differences.	• GaussDB does not support the IS JSON and JSON_TEXTCONTAINS conditions. • The JSONB_EQ condition in GaussDB is the same as the JSON_EQUAL condition in Oracle Database. However, GaussDB does not support the ERROR clause. • The JSONB_EXISTS condition in GaussDB is the same as the JSON_EXISTS condition in Oracle Database. However, GaussDB does not support the ERROR, EMPTY, or PASSING clauses.
10	Compound conditions	Supported.	-
11	BETWEEN condition	Supported.	-
12	EXISTS condition	Supported.	-
13	IN condition	Supported.	-
14	IS OF TYPE condition	Not supported.	-

2.7 Drivers

2.7.1 JDBC

2.7.1.1 Array

This section describes the differences between Oracle Database and GaussDB when the JDBC driver of the java.sql.Array type is used.

Table 2-34 Constructor reference

Constructor	Oracle Database	GaussDB	API Difference
1. Use the static constructor of ArrayDescriptor to construct an ArrayDescriptor object. 2. Use ArrayDescriptor to construct an array object.	String typeName = "XXX"; Connection conn = getConnection(); Object[] elements = null; ArrayDescriptor desc = ArrayDescriptor.createDescriptor(typeName, conn); Array array = new ARRAY(desc, conn, elements);	String typeName = "xxx"; Connection conn = getConnection(); Object[] elements = null; ArrayDescriptor desc = ArrayDescriptor.getDescriptor(typeName, conn); Array array = new GaussArray(desc, elements);	• Different names of the static constructor for ArrayDescriptor: In Oracle Database, it is createDescriptor, while in GaussDB it is getDescriptor. • Different constructor names for arrays: In Oracle Database, it is ARRAY(ArrayDescriptor, Connection, Object), while in GaussDB, it is GaussArray(ArrayDescriptor, Object). • Variable description: **typeName** indicates the type name and is case-sensitive. Generally, it uses uppercase in Oracle Database, while in GaussDB it is lowercase. **conn** indicates the connection object for the corresponding database. **elements** indicates the element data of the corresponding type.

NOTE

1. GaussDB currently does not support the constructors not listed in the aforementioned table.
2. If the element type is a character type and the length of the construction input string exceeds that defined by the element type, Oracle Database reports an error during input parameter binding.

   If the array type is the varray type and the number of elements exceeds the maximum length of varray, Oracle Database reports an error during input parameter binding.

   GaussDB does not verify type modifiers when constructing or binding input parameters. When a database receives array objects and executes SQL statements, it decides whether to report an error.

Table 2-35 API reference

Method	Return Value Type	Throws	GaussDB
getBaseTypeName()	String	SQLException	Supported.
getBaseType()	int	SQLException	Supported.
getArray()	Object	SQLException	Supported.
getArray(java.util.Map<String,Class<?>> map)	Object	SQLException	Not supported.
getArray(long index, int count)	Object	SQLException	Supported.
getArray(long index, int count, java.util.Map<String,Class<?>> map)	Object	SQLException	Not supported.
getResultSet()	ResultSet	SQLException	Not supported.
getResultSet(java.util.Map<String,Class<?>> map)	ResultSet	SQLException	Not supported.
getResultSet(long index, int count)	ResultSet	SQLException	Not supported.
getResultSet (long index, int count, java.util.Map<String,Class<?>> map)	ResultSet	SQLException	Not supported.
free()	void	SQLException	Not supported.

Table 2-36 Differences in the getArray() API

Element Database Type	Actual Return Value Type of the getArray API (Oracle Database)	Actual Return Value Type of the getArray API (GaussDB)
CHAR	java.lang.String[]	java.lang.String[]
VARCHAR/ VARCHAR2	java.lang.String[]	java.lang.String[]
NCHAR	java.lang.String[]	java.lang.String[]
NVARCHAR2	java.lang.String[]	java.lang.String[]
NUMBER	java.math.BigDecimal[]	java.math.BigDecimal[]
NUMERIC	java.math.BigDecimal[]	java.math.BigDecimal[]
DECIMAL	java.math.BigDecimal[]	java.math.BigDecimal[]
INTEGER	java.math.BigDecimal[]	java.lang.Integer[]
SMALLINT	java.math.BigDecimal[]	java.lang.Short[]
DOUBLE PRECISION	java.math.BigDecimal[]	java.lang.Double[]
FLOAT	java.math.BigDecimal[]	java.lang.Double[]
REAL	java.math.BigDecimal[]	java.lang.Float[]
BINARY_DOUBLE	java.lang.Double[]	java.lang.Double[]
BINARY_INTEGER	java.math.BigDecimal[]	java.lang.Integer[]
BOOLEAN	java.math.BigDecimal[]	java.lang.Boolean[]
TIMESTAMP	java.sql.Timestamp[]	java.sql.Timestamp[]
TIMESTAMP WITH TIME ZONE	java.time.OffsetDateTime[]	java.sql.Timestamp[]
BLOB	oracle.sql.BLOB[]	java.sql.Blob[]
CLOB	oracle.sql.CLOB[]	java.sql.Clob[]
Set/Array	java.lang.Object[]	java.sql.Array[]
RECORD	java.lang.Object[]	java.sql.Struct[]

NOTE

1. GaussDB currently does not support the types unlisted in the aforementioned table.
2. For details about the differences in the return values of the getArray(long index, int count) API, see the preceding table.
3. The differences in **index** parameter of getArray(long index, int count) are as follows:
   - The value range supported by Oracle Database is [1, Long.MAX_VALUE]. GaussDB supports a range of [1, Integer.MAX_VALUE].
   - If the **index** value is greater than that of **Integer.MAX_VALUE**, it will be truncated in Oracle Database, while an error is reported in GaussDB.

Table 2-37 Differences in the getBaseType() API

Element Database Type	Return Value of the getBaseType API (Oracle Database)	Return Value of the getBaseType API (GaussDB)
CHAR	java.sql.Types.CHAR	java.sql.Types.CHAR
VARCHAR/ VARCHAR2	java.sql.Types.VARCHAR	java.sql.Types.VARCHAR
NCHAR	java.sql.Types.NCHAR	java.sql.Types.CHAR
NVARCHAR2	java.sql.Types.NVARCHAR	java.sql.Types.VARCHAR
NUMBER	java.sql.Types.NUMERIC	java.sql.Types.NUMERIC
NUMERIC	java.sql.Types.DECIMAL	java.sql.Types.NUMERIC
DECIMAL	java.sql.Types.DECIMAL	java.sql.Types.NUMERIC
INTEGER	java.sql.Types.NUMERIC	java.sql.Types.INTEGER
SMALLINT	java.sql.Types.NUMERIC	java.sql.Types.SMALLINT
DOUBLE PRECISION	java.sql.Types.FLOAT	java.sql.Types.DOUBLE
FLOAT	java.sql.Types.FLOAT	java.sql.Types.DOUBLE
REAL	java.sql.Types.FLOAT	java.sql.Types.REAL
BINARY_DOUBLE	oracle.jdbc.OracleTypes.BINARY_DOUBLE	java.sql.Types.DOUBLE
BINARY_INTEGER	java.sql.Types.NUMERIC	java.sql.Types.INTEGER
BOOLEAN	java.sql.Types.NUMERIC	java.sql.Types.BIT
TIMESTAMP	java.sql.Types.TIMESTAMP	java.sql.Types.TIMESTAMP
TIMESTAMP WITH TIME ZONE	oracle.jdbc.OracleTypes.TIMESTAMPTZ	java.sql.Types.TIMESTAMP
BLOB	java.sql.Types.BLOB	java.sql.Types.BLOB

Element Database Type	Return Value of the getBaseType API (Oracle Database)	Return Value of the getBaseType API (GaussDB)
CLOB	java.sql.Types.CLOB	java.sql.Types.CLOB
Set/Array	java.sql.Types.ARRAY	java.sql.Types.ARRAY
RECORD	java.sql.Types.STRUCT	java.sql.Types.STRUCT

NOTE

GaussDB currently does not support the types unlisted in the aforementioned table.

Table 2-38 Differences in the getBaseTypeName() API

Element Database Type	Return Value of the getBaseTypeName API (Oracle Database)	Return Value of the getBaseTypeName API (GaussDB)
CHAR	"CHAR"	"bpchar"
VARCHAR/ VARCHAR2	"VARCHAR"	"varchar"
NCHAR	"NCHAR"	"bpchar"
NVARCHAR2	"NVARCHAR"	"nvarchar2"
NUMBER	"NUMBER"	"numeric"
NUMERIC	"DECIMAL"	"numeric"
DECIMAL	"DECIMAL"	"numeric"
INTEGER	"NUMBER"	"int4"
SMALLINT	"NUMBER"	"int2"
DOUBLE PRECISION	"FLOAT"	"float8"
FLOAT	"FLOAT"	"float8"
REAL	"FLOAT"	"float4"
BINARY_DOUBLE	"BINARY_DOUBLE"	"float8"
BINARY_INTEGER	"NUMBER"	"int4"
BOOLEAN	"NUMBER"	"bool"
TIMESTAMP	"TIMESTAMP"	"timestamp"
TIMESTAMP WITH TIME ZONE	"TIMESTAMP WITH TIME ZONE"	"timestamptz"

Element Database Type	Return Value of the getBaseTypeName API (Oracle Database)	Return Value of the getBaseTypeName API (GaussDB)
BLOB	"BLOB"	"blob"
CLOB	"CLOB"	"clob"
Set/Array	See the description below.	See the description below.
RECORD	See the description below.	See the description below.

NOTE

1. GaussDB currently does not support the types unlisted in the aforementioned table.
2. When an element is of the set, array, or RECORD type that is defined within a package, the return rules for getBaseTypeName are as follows:
   - OJDBC11 returns *schemaName.packageName.typeName*.
   - OJDBC8 generally returns *schemaName.packageName.typeName*, or returns *"schemaName"."packageName.typeName"* in the following condition:

     Any of *schemaName*, *packageName*, or *typeName* does not meet the rule of starting with a letter followed by characters including letters, digits, or underscores.
   - GaussDB generally returns *schemaName.packageName.typeName*, or returns *"schemaName"."packageName"."typeName"* in the following condition:

     Any of *schemaName*, *packageName*, or *typeName* does not meet the rule of starting with a letter or underscore followed by characters including letters, digits, or underscores.
3. When an element is of the set, array, or RECORD type that is defined in the schema (but not in the package), the return rules for getBaseTypeName are as follows:
   - OJDBC11 returns *schemaName.typeName*.
   - OJDBC8 generally returns *schemaName.typeName*, or returns *"schemaName"."typeName"* in the following condition:

     Any of *schemaName* or *typeName* does not meet the rule of starting with a letter followed by characters including letters, digits, or underscores.
   - GaussDB generally returns *schemaName.typeName*, or returns *"schemaName"."typeName"* in the following condition:

     Any of *schemaName* or *typeName* does not meet the rule of starting with a letter or underscore followed by characters including letters, digits, or underscores.
4. If no special processing is performed during element type creation, the getBaseTypeName API typically returns the type name in uppercase in Oracle Database, while returns the type name in lowercase in GaussDB.

Table 2-39 Differences in array construction APIs

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
CHAR	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by element input parameters can be seen in the table.
VARCHAR/ VARCHAR2	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by element input parameters can be seen in the table.
NCHAR	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by element input parameters can be seen in the table.
NVARCHAR2	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by element input parameters can be seen in the table.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.
NUMERIC	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
DECIMAL	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.
INTEGER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. When the input parameter value exceeds the integer range, GaussDB reports an error.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
SMALLINT	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. When the input parameter exceeds the Short range, GaussDB reports an error.
DOUBLE PRECISION	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. Converting a higher-precision type to Double may result in precision loss.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
FLOAT	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. Converting a higher-precision type to Double may result in precision loss.
REAL	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. Converting a higher-precision type to Float may result in precision loss.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
BINARY_DOUBLE	Double and oracle.sql.BINARY_DOUBLE	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API. 3. Converting a higher-precision type to Double may result in precision loss.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
BINARY_INTEGER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, AtomicInteger, AutomicLong, DoubleAccumulator, DoubleAddr, LongAccumulator, LondAdder, Striped64, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The input parameters of the OJDBC11 element support the following types: Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER. 3. The target types are inconsistent. For details, see differences in the getArray() API. 4. When the input value exceeds the Integer range, GaussDB reports an error. Oracle Database OJDBC8 performs data truncation.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
BOOLEAN	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, AtomicInteger, AutomicLong, DoubleAccumulator, DoubleAddr, LongAccumulator, LondAdder, Striped64, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by element input parameters can be seen in the table. 2. The input parameters of the OJDBC8 element support the following types: Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER. 3. The target types are inconsistent. For details, see differences in the getArray() API. 4. In GaussDB, the target data type is Boolean, which only supports inputs of **1**, **0**, **"true"**, and **"false"**.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
TIMESTAMP	byte[], java.sql.Date, Calendar, java.util.Date, LocalDate, LocalDateTime, LocalTime, OffsetDateTime, OffsetTime, String, java.sql.Time, java.sql.Timestamp, oracle.sql.DATE, oracle.sql.TIMESTAMP, oracle.sql.TIMESTAMPTZ, oracle.sql.TIMESTAMPLTZ, and ZonedDateTime	java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, LocalDateTime, and String	The different types supported by element input parameters can be seen in the table.
TIMESTAMP WITH TIME ZONE	java.sql.Date, Calendar, java.util.Date, LocalDate, LocalDateTime, LocalTime, OffsetDateTime, OffsetTime, String, java.sql.Time, java.sql.Timestamp, oracle.sql.DATE, oracle.sql.TIMESTAMP, oracle.sql.TIMESTAMPTZ, oracle.sql.TIMESTAMPLTZ, and ZonedDateTime	java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, LocalDateTime, and String	1. The different types supported by element input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getArray() API.
BLOB	oracle.sql.BLOB and oracle.jdbc.driver.OracleBlob	PGBlob	The different types supported by element input parameters can be seen in the table.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
CLOB	oracle.sql.CLOB, oracle.jdbc.driver.OracleClob, InputStream, and Reader	PGClob	The different types supported by element input parameters can be seen in the table.
Set/Array	Array and Object	GaussArray and Object	1. The different types supported by element input parameters can be seen in the table. 2. Oracle Database: No error is reported when the input parameter is of the Array type even if the Array type differs from the element type. GaussDB: An error is reported when the input parameter is of the GaussArray type that differs from the element type. 3. When the input parameter of an element is of the Object type, refer to the differences in array construction APIs.

Element Database Type	List of Java Types Supported by Element Input Parameters (Oracle Database OJDBC8)	List of Java Types Supported by Element Input Parameters (GaussDB)	Difference
RECORD	Struct and Object[]	GaussStruct and Object[]	1. The different types supported by element input parameters can be seen in the table. 2. Oracle Database: No error is reported when the input parameter is of the Struct type even if the Struct type differs from the element type. GaussDB: An error is reported when the input parameter is of the GaussStruct type that differs from the element type. 3. When the input parameter of an element is of the Object[] type, refer to the differences in struct construction APIs.

NOTE

1. When an array is constructed, if the Java type of the input element does not match the target type, an implicit conversion operation is performed. For details on the Java types of input elements supported by various database element types, refer to the preceding table.
2. GaussDB currently does not support the types unlisted in the aforementioned table.
3. The constructor needs to provide an element array. The preceding table describes the differences between elements in the array.

2.7.1.2 Struct

This section describes the differences between Oracle Database and GaussDB when the JDBC driver of the java.sql.Struct type is used.

Table 2-40 Constructor reference

Constructor	Oracle Database	GaussDB	API Difference
1. Use the static constructor of StructDescriptor to construct a StructDescriptor object. 2. Use StructDescriptor to construct a struct object.	String typeName = "XXX"; Connection conn = getConnection(); Object[] attributes = null; StructDescriptor desc = StructDescriptor.createDescriptor(typeName, conn); Struct struct = new STRUCT(desc, conn, attributes);	String typeName = "xxx"; Connection conn = getConnection(); Object[] elements = null; StructDescriptor desc = StructDescriptor.getDescriptor(typeName, conn); Struct struct = new GaussStruct(desc, attributes);	• Different names of the static constructor for StructDescriptor: In Oracle Database, it is createDescriptor, while in GaussDB it is getDescriptor. • Different constructor names for structs. In Oracle Database, it is defined as STRUCT(StructDescriptor, Connection, Object[]), while in GaussDB, it is defined as GaussStruct(StructDescriptor, Object[]). • Variable description: **typeName** indicates the type name and is case-sensitive. Generally, it uses uppercase in Oracle Database, while in GaussDB it is lowercase. **conn** indicates the connection object for the corresponding database. **attributes** indicates the element data array.

Constructor	Oracle Database	GaussDB	API Difference
Use the createStruct standard API of Connection to construct a struct object.	String typeName = "XXX"; Connection conn = getConnection(); Object[] attributes = null; Struct struct = conn.createStruct(typeName, attributes);	String typeName = "XXX"; Connection conn = getConnection(); Object[] attributes = null; Struct struct = conn.createStruct(typeName, attributes);	• Variable description: **typeName** indicates the type name and is case-sensitive. Generally, it uses uppercase in Oracle Database, while in GaussDB it is lowercase. **conn** indicates the connection object for the corresponding database. **attributes** indicates the element data array.

NOTE

1. GaussDB currently does not support the constructors not listed in the aforementioned table.
2. If the attribute type is a character type and the length of the construction input string exceeds that defined by the element type, Oracle Database reports an error during input parameter binding.

   GaussDB does not verify type modifiers when constructing or binding input parameters. When a database receives struct objects and executes SQL statements, it decides whether to report an error.
3. If the number of array elements exceeds the actual number of columns of the corresponding type, an error is reported during creation.

   When the number of array elements is less than the actual number of columns, the creation of Oracle Database is successful, but an error is reported during parameter input for execution; GaussDB reports an error during creation.

Table 2-41 API reference

Method	Return Value Type	Throws	GaussDB
getSQLTypeName()	String	SQLException	Supported.
getAttributes()	Object[]	SQLException	Supported.

Method	Return Value Type	Throws	GaussDB
getAttributes(java.util.Map<String,Class<?>> map)	Object[]	SQLException	Not supported.

NOTE

The differences in the getSQLTypeName API are as follows:

1. For the package type, the struct constructed in the *packageName.typeName* format, the differences in the getSQLTypeName API are as follows:
   - OJDBC11 returns *packageName.typeName*.
   - OJDBC8 generally returns *packageName.typeName*, or returns *"packageName"."typeName"* when *packageName* and *typeName* meet the following condition:

     Any of *packageName* or *typeName* does not meet the rule of starting with a letter followed by characters including letters, digits, or underscores.
   - GaussDB generally returns *schemaName.packageName.typeName*, or returns *"schemaName"."packageName"."typeName"* when *schemaName*, *packageName*, and *typeName* meet the following condition:

     Any of *schemaName*, *packageName*, or *typeName* does not meet the rule of starting with a letter or underscore followed by characters including letters, digits, or underscores.
2. For the package type, the differences in the getSQLTypeName API in other scenarios are as follows:
   - OJDBC11 returns *schemaName.packageName.typeName*.
   - OJDBC8 generally returns *schemaName.packageName.typeName*, or returns *"schemaName"."packageName.typeName"* when *schemaName*, *packageName*, and *typeName* meet the following condition:

     Any of *schemaName*, *packageName*, or *typeName* does not meet the rule of starting with a letter followed by characters including letters, digits, or underscores.
   - GaussDB generally returns *schemaName.packageName.typeName*, or returns *"schemaName"."packageName"."typeName"* when *schemaName*, *packageName*, and *typeName* meet the following condition:

     Any of *schemaName*, *packageName*, or *typeName* does not meet the rule of starting with a letter or underscore followed by characters including letters, digits, or underscores.
3. For the non-package type, the differences in the getSQLTypeName API are as follows:
   - OJDBC11 returns *schemaName.typeName*.
   - OJDBC8 generally returns *schemaName.typeName*, or returns *"schemaName"."typeName"* when *schemaName* and *typeName* meet the following condition:

     Any of *schemaName* or *typeName* does not meet the rule of starting with a letter followed by characters including letters, digits, or underscores.
   - GaussDB generally returns *schemaName.typeName*, or returns *"schemaName"."typeName"* when *schemaName* and *typeName* meet the following condition:

     Any of *schemaName* or *typeName* does not meet the rule of starting with a letter or underscore followed by characters including letters, digits, or underscores.

Table 2-42 Differences in the getAttributes() API

Database Attribute Type	Java Type of the Corresponding Element in the Return Value (Oracle Database OJDBC8)	Java Type of the Corresponding Element in the Return Value (Oracle Database OJDBC11)	Java Type of the Corresponding Element in the Return Value (GaussDB)
CHAR	String	String	String
VARCHAR/ VARCHAR2	String	String	String
NCHAR	String	String	String
NVARCHAR2	String	String	String
NUMBER	BigDecimal	BigDecimal	BigDecimal
NUMERIC	BigDecimal	BigDecimal	BigDecimal
DECIMAL	BigDecimal	BigDecimal	BigDecimal
INTEGER	BigDecimal	BigDecimal	Integer
SMALLINT	BigDecimal	BigDecimal	Short
DOUBLE PRECISION	BigDecimal	BigDecimal	Double
FLOAT	BigDecimal	BigDecimal	Double
REAL	BigDecimal	BigDecimal	Float
BINARY_DOUBLE	Double	Double	Double
BINARY_INTEGER	BigDecimal	Integer	Integer
BOOLEAN	BigDecimal	Integer	Boolean
TIMESTAMP	Timestamp	Timestamp	Timestamp
TIMESTAMP WITH TIME ZONE	TIMESTAMPTZ	TIMESTAMPTZ	Timestamp
BLOB	BLOB	BLOB	PGBlob
CLOB	CLOB	CLOB	PGClob
Set/Array	ARRAY	ARRAY	GaussArray
RECORD	STRUCT	STRUCT	GaussStruct

NOTE

GaussDB currently does not support the types unlisted in the aforementioned table.

Table 2-43 Differences in struct construction APIs

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
CHAR	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, String, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	1. The different types supported by attribute input parameters can be seen in the table. 2. When the input parameter is of the String type, Oracle Database adds spaces at the end of the string until its length matches the length defined by the type; GaussDB does not add spaces.
VARCHAR/ VARCHAR2	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, String, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by attribute input parameters can be seen in the table.
NCHAR	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, String, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by attribute input parameters can be seen in the table.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
NVARCHAR2	Any Java type	Byte, Short, Integer, Long, BigInteger, BigDecimal, Float, Double, Character, Boolean, String, java.sql.Date, java.sql.Time, java.sql.Timestamp, and PGClob	The different types supported by attribute input parameters can be seen in the table.
NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.
NUMERIC	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
DECIMAL	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. When the input parameter is of the Float, Double, BigDecimal, or String type and the decimal part is 0, Oracle Database truncates the decimal part, while GaussDB retains it.
INTEGER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. When the input parameter value exceeds the integer range, GaussDB reports an error.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
SMALLINT	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. When the input parameter exceeds the Short range, GaussDB reports an error.
DOUBLE PRECISION	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. Converting a higher-precision type to Double may result in precision loss.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
FLOAT	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. Converting a higher-precision type to Double may result in precision loss.
REAL	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. Converting a higher-precision type to Float may result in precision loss.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
BINARY_DOUBLE	byte[], Double, and oracle.sql.BINARY_DOUBLE	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. Converting a higher-precision type to Double may result in precision loss.
BINARY_INTEGER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, AtomicInteger, AutomicLong, DoubleAccumulator, DoubleAddr, LongAccumulator, LondAdder, Striped64, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. If the input value exceeds the integer range, GaussDB reports an error, whereas Oracle Database truncates the value.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
BOOLEAN	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, AtomicInteger, AutomicLong, DoubleAccumulator, DoubleAddr, LongAccumulator, LondAdder, Striped64, String, and oracle.sql.NUMBER	Byte, Short, Integer, Long, Float, Double, Boolean, BigDecimal, BigInteger, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API. 3. In GaussDB, the target data type is Boolean, which only supports inputs of **1**, **0**, **"true"**, and **"false"**.
TIMESTAMP	byte[], java.sql.Date, String, java.sql.Time, java.sql.Timestamp, oracle.sql.TIMESTAMP, and oracle.sql.DATE	java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, LocalDateTime, and String	The different types supported by attribute input parameters can be seen in the table.
TIMESTAMP WITH TIME ZONE	java.sql.Date, Calendar, java.util.Date, LocalDate, LocalDateTime, LocalTime, OffsetDateTime, OffsetTime, String, java.sql.Time, java.sql.Timestamp, oracle.sql.TIMESTAMP, oracle.sql.TIMESTAMPTZ, oracle.sql.TIMESTAMPLTZ, and ZonedDateTime	java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, LocalDateTime, and String	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
BLOB	oracle.sql.BLOB and oracle.jdbc.driver.OracleBlob	PGBlob	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API.
CLOB	oracle.sql.CLOB and oracle.jdbc.driver.OracleClob	PGClob	1. The different types supported by attribute input parameters can be seen in the table. 2. The target types are inconsistent. For details, see differences in the getAttributes() API.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
Set/Array	Array and Object	GaussArray and Object	1. The different types supported by attribute input parameters can be seen in the table. 2. In Oracle Database, no error is reported when the attribute input parameter is of the Array type, even if the Array type differs from the actual type required by the attribute. In GaussDB, an error is reported when the attribute input parameter is of the GaussArray type that differs from the actual type required by the attribute. 3. When the input parameter of an attribute is of the Object type, refer to the differences in array construction APIs.

Database Attribute Type	List of Java Types Supported by Attribute Input Parameters (Oracle Database)	List of Java Types Supported by Attribute Input Parameters (GaussDB)	Difference
RECORD	Struct and Object[]	GaussStruct and Object[]	1. The different types supported by attribute input parameters can be seen in the table. 2. In Oracle Database, no error is reported when the attribute input parameter is of the Struct type, even if the Struct type differs from the actual type required by the attribute. In GaussDB, an error is reported when the attribute input parameter is of the GaussStruct type that differs from the actual type required by the attribute. 3. When the input parameter of an attribute is of the Object[] type, refer to the differences in struct construction APIs.

NOTE

1. When a struct is constructed, if the Java type of the input element does not match the target type, an implicit conversion operation is performed. For details on the Java types of input elements supported by various database element types, refer to the preceding table.
2. GaussDB currently does not support the types unlisted in the aforementioned table.
3. The constructor needs to provide an attribute array. The preceding table describes the differences of each attribute in the array.

2.8 Common SQL DDL Clauses

This chapter describes common compatible SQL DDL clauses, including allocate_extent_clause, constraint, deallocate_unused_clause, file_specification, logging_clause, parallel_clause, physical_attributes_clause, size_clause, storage_clause, and aggregate function nesting. For details, see **Table 2-44**.

Table 2-44 Common SQL DDL clauses

No.	Oracle Database	GaussDB	Difference
1	allocate_extent_clause Syntax: ALLOCATE EXTENT [({ SIZE size_clause | DATAFILE 'filename' | INSTANCE integer } ...)] For example, after the **employees** table is created, change the allocated extent **size** of the table to **10M**. SQL> CREATE TABLE employees(EMPLOYEE_ID NUMBER(38), JOB_ID NUMBER(38), SALARY NUMBER(38), LAST_NAME VARCHAR2(16)); Table created. SQL> ALTER TABLE employees ALLOCATE EXTENT (SIZE 10M); Table altered.	Not supported.	-

No.	Oracle Database	GaussDB	Difference
2	constraint Syntax: { inline_constraint | out_of_line_constraint | inline_ref_constraint | out_of_line_ref_constraint } For example, when you create the **staff** table, the **ID** and **NAME** columns specified in the constraint clause cannot be empty. SQL> CREATE TABLE staff(ID INT NOT NULL, NAME char(8) NOT NULL, AGE INT, ADDRESS CHAR(50), SALARY REAL); Table created.	Supported.	-
3	deallocate_unused_clause Syntax: DEALLOCATE UNUSED [KEEP size_clause] For example, after creating the **employees** table and performing some INSERT and DELETE operations, you want to use the deallocate_unused_clause to release the unused space of the **employees** table. SQL> CREATE TABLE employees(EMPLOYEE_ID NUMBER(38), JOB_ID NUMBER(38), SALARY NUMBER(38), LAST_NAME VARCHAR2(16)); Table created. - Perform some INSERT and DELETE operations. SQL> ALTER TABLE employees DEALLOCATE UNUSED; Table altered.	Not supported.	-

No.	Oracle Database	GaussDB	Difference
4	file_specification Syntax: {['filename' | 'ASM_filename'] [SIZE size_clause] [REUSE] [autoextend_clause]} | {['filename | ASM_filename' | ('filename | ASM_filename' [, 'filename | ASM_filename']...)] [SIZE size_clause] [BLOCKSIZE size_clause [REUSE]} For example, to create a temporary tablespace **tbs_temp_01**, the file_specification clause of the SQL statement specifies that a temporary database file **templ01.dbf** is created in the tablespace. The tablespace can be automatically expanded and allocated to the tablespace group **tbs_grp_01**. SQL> CREATE TEMPORARY TABLESPACE tbs_temp_01 TEMPFILE 'temp01.dbf' AUTOEXTEND ON TABLESPACE GROUP tbs_grp_01; Tablespace created.	Not supported.	-

No.	Oracle Database	GaussDB	Difference

5	logging_clause
	Syntax:
	{ LOGGING | NOLOGGING | FILESYSTEM_LIKE_LOGGING }
		Partially supported, with differences.	• GaussDB does not support the LOGGING and FILESYSTEM_LIKE_LOGGING constraint clauses.
			Example:
			When a table is created in GaussDB with the LOGGING constraint clause, a syntax error is reported.
			gaussdb=# CREATE LOGGING TABLE my_tab(id int, name char(16));
			ERROR: syntax error at or near "LOGGING"
			LINE 1: CREATE LOGGING TABLE my_tab(id int, name char(16));
			^
			When a table is created in GaussDB with the FILESYSTEM_LIKE_LOGGING constraint clause, a syntax error is reported.
			gaussdb=# CREATE FILESYSTEM_LIKE_LOGGING TABLE my_tab(id int, name char(16));
			ERROR: syntax error at or near "FILESYSTEM_LIKE_LOGGING"
			LINE 1: CREATE FILESYSTEM_LIKE_LOGGING TABLE my_tab(id int, name cha...
			^
			• GaussDB supports only table-level UNLOGGED constraints and does not support column-level UNLOGGED constraints. For example, when a table is created in GaussDB with the column-level UNLOGGED constraint clause, a syntax error is reported.
			gaussdb=# CREATE UNLOGGED TABLE my_tab(id int UNLOGGED, name char(16));
			ERROR: syntax error at or near "UNLOGGED"
			LINE 1: CREATE UNLOGGED TABLE my_tab(id int UNLOGGED, name char(16))...
			^
			• GaussDB uses logging clauses only in the CREATE TABLE, CREATE TABLE AS, and SELECT INTO statements.
			For example, when a TABLESPACE statement with the

No.	Oracle Database	GaussDB	Difference
			UNLOGGED constraint clause is created in GaussDB, a syntax error is reported. gaussdb=# CREATE UNLOGGED TABLESPACE tbs1 RELATIVE LOCATION 'tablespace1/tablespace_1'; ERROR: syntax error at or near "TABLESPACE" LINE 1: CREATE UNLOGGED TABLESPACE tbs1 RELATIVE LOCATION 'tablespac... ^
6	parallel_clause Syntax: { NOPARALLEL | PARALLEL [integer] } For example, if you create table **t1** and specify **PARALLEL 4** in the parallel_clause, a maximum of four parallel processes can be used to query and update table **t1**. SQL> CREATE TABLE t1 (id NUMBER, name VARCHAR2(50)) PARALLEL 4; Table created.	Not supported.	-

No.	Oracle Database	GaussDB	Difference

7	physical_attributes_clause
	Syntax:
	[{ PCTFREE integer | PCTUSED integer | INITRANS integer | storage_clause }...]
		Partially supported, with differences.	• GaussDB does not support PCTUSED.
			For example, if you run an SQL statement to create the **tbl1_ind** index in the **tbl1** table and set the space usage **PCTUSED** of the index to **20%** in the physical_attributes_clause of the statement, an error is reported when the SQL statement is executed in GaussDB.
			gaussdb=# CREATE INDEX tbl1_ind ON tbl1 (name) PCTUSED 20;
			ERROR: syntax error at or near "PCTUSED"
			LINE 1: CREATE INDEX tbl1_ind ON tbl1 (name) PCTUSED 20;
			^
			• GaussDB uses physical_attributes_clause only in the CREATE TABLE and CREATE INDEX statements.
			For example, if you run an SQL statement to obtain data from the **tbl1** table, create the materialized view **tbl1_mv**, and set the number of initial transactions of the view to **30** in the physical_attributes_clause, an error is reported when GaussDB executes the statement.
			gaussdb=# CREATE MATERIALIZED VIEW tbl1_mv INITRANS 30 as select * from tbl1;
			ERROR: syntax error at or near "INITRANS"
			LINE 1: CREATE MATERIALIZED VIEW tbl1_mv INITRANS 30 as select * fro...
			^

No.	Oracle Database	GaussDB	Difference
8	size_clause Syntax: integer [K | M | G | T | P | E] For example, create a temporary tablespace **tbs_temp_01** and a temporary database file **templ01.dbf** in the tablespace. The initial size of the tablespace is **5M** as specified by the size_clause in the SQL statement, which can be automatically expanded. The tablespace can be allocated to the tablespace group **tbs_grp_01**. SQL> CREATE TEMPORARY TABLESPACE tbs_temp_01 TEMPFILE 'temp01.dbf' SIZE 5M AUTOEXTEND ON TABLESPACE GROUP tbs_grp_01; Tablespace created.	Not supported.	-

No.	Oracle Database	GaussDB	Difference
9	storage_clause Syntax: STORAGE ({ INITIAL size_clause | NEXT size_clause | MINEXTENTS integer | MAXEXTENTS { integer | UNLIMITED } | maxsize_clause | PCTINCREASE integer | FREELISTS integer | FREELIST GROUPS integer | OPTIMAL [size_clause | NULL] | BUFFER_POOL { KEEP | RECYCLE | DEFAULT } | FLASH_CACHE { KEEP | NONE | DEFAULT } | (CELL_FLASH_CACHE (KEEP | NONE | DEFAULT)) | ENCRYPT } ...)	Partially supported, with differences.	• In Oracle Database, storage parameters are specified by the STORAGE clause. In GaussDB, storage parameters are specified by the WITH clause. Example: To create the **my_tab1** table in Oracle Database, set the initial size of the table to **10M** in the storage_clause, and add 5 MB each time when more space is required, run the following SQL statement: SQL> CREATE TABLE my_tab1 (id NUMBER(10) PRIMARY KEY, name VARCHAR2(50)) STORAGE (INITIAL 10M NEXT 5M); Table created. To create the **my_tab2** table in GaussDB and set the storage engine type to **ustore** in the storage_clause, run the following SQL statement: gaussdb=# CREATE TABLE my_tab2 (id NUMBER(10) PRIMARY KEY, name VARCHAR2(50)) with (storage_type=ustore); NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "my_tab2_pkey" for table "my_tab2" CREATE TABLE • Optional storage parameters in GaussDB are greatly different from those in Oracle Database. For details, see the GaussDB parameter description in "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Developer Guide*. **WITH ({storage_parameter = value} [, ...])** describes the storage parameters supported by the CREATE TABLE statement.

No.	Oracle Database	GaussDB	Difference
10	Aggregate function nesting For example, create the **revenue** table generated by nesting the aggregate functions MIN() and SUM() in the **sales_amount** column of the **sales** table. SQL> CREATE TABLE sales(ID INT, SALES_AMOUNT INT); Table created. SQL> INSERT INTO sales VALUES(1, 100); 1 row created. SQL> INSERT INTO sales VALUES (3, 200); 1 row created. SQL> CREATE TABLE revenue as SELECT SUM(MIN(sales_amount)) as total from sales group by sales_amount; Table created.	Supported.	-
11	Dropping a system schema Syntax: DROP USER schema_name CASCADE; For example, drop the SYS schema as the **SYS** user. SQL> DROP USER SYS; DROP USER SYS * ERROR at line 1: ORA-28050: specified user or role cannot be dropped	Supported.	-

2.9 SQL Queries and Subqueries

GaussDB is compatible with SQL queries and subqueries except hierarchical queries.

Table 2-45 SQL queries and subqueries

No.	Oracle Database	GaussDB	Difference
1	Creating simple queries	Supported.	-
2	Hierarchical queries	Not supported.	-
3	UNION [ALL], INTERSECT, MINUS	Supported.	-
4	Sorting query results	Supported, with differences.	If the GaussDB query does not contain groups and the target column contains both an aggregate function and a set returning function, the sorting of the set returning function column is not ignored.
5	Joins	Supported, with differences.	GaussDB supports only the same join types as Oracle Database, such as left/right, self, natural, and full outer join. Join optimization methods such as IN-MEMORY JOIN GROUPS are not supported.
6	Using subqueries	Supported.	-
7	Unnesting of nested subqueries	Supported, with differences.	HASH_AJ or MERGE_AJ cannot be explicitly specified in GaussDB.
8	Distributed queries	Supported, with differences.	GaussDB requires explicit database link query.
9	Aggregate function nesting	Supported.	-

2.10 PL/SQL Language

GaussDB is compatible with PL/SQL operators, expressions, control statements, collections, and records, but does not support predefined PL/SQL constants, types, and subtypes.

2.10.1 Basic PL/SQL Syntax

Table 2-46 PL/SQL operators

No.	Oracle Database	GaussDB
1	+	Supported.
2	:=	Supported.
3	=>	Supported.
4	%	Supported.
5	'	Supported.
6	.	Supported.
7	||	Supported.
8	/	Supported.
9	**	Not supported.
10	(	Supported.
11	)	Supported.
12	:	Supported.
13	,	Supported.
14	<<	Supported.
15	>>	Supported.
16	/*	Supported.
17	*/	Supported.
18	*	Supported.
19	"	Supported.
20	..	Supported.
21	=	Supported.
22	<>	Supported.
23	!=	Supported.
24	~=	Supported.
25	^=	Supported.
26	<	Supported.
27	>	Supported.
28	<=	Supported.

No.	Oracle Database	GaussDB
29	>=	Supported.
30	@	Supported.
31	--	Supported.
32	;	Supported.
33	-	Supported.

Table 2-47 Logical operators

No.	Oracle Database	GaussDB
1	NOT	Supported.
2	AND	Supported.
3	OR	Supported.

Table 2-48 Comparison expressions

No.	Oracle Database	GaussDB
1	IS [NOT] NULL	Supported.
2	LIKE	Supported.
3	BETWEEN	Supported.
4	IN	Supported.

Table 2-49 CASE expressions

No.	Oracle Database	GaussDB
1	simple CASE	Supported.
2	searched CASE	Supported.

Table 2-50 Parameters related to variable declaration

No.	Oracle Database	GaussDB	Difference
1	%TYPE	Supported, with differences.	• GaussDB does not support *record%type*. • GaussDB does not support **pkg.***record%type* and **schema.pkg.***record%type* as the input and output parameter types. • In GaussDB, *Table/View*.**column.***column%type* or **schema.***Table/View*.**column.***column%type* cannot be nested with one or more layers as variable types or input/output parameter type. • In GaussDB, *record*.**column.***column%type* and **pkg.***record*.**column.***column%TYPE* cannot be nested with a column type of records at one or more layers as the variable type or input/output parameter type.
2	%ROWTYPE	Supported, with differences.	• When GaussDB has multiple CNs, the **%ROWTYPE** and **%TYPE** attributes of the temporary table cannot be declared in a stored procedure. The temporary table is valid only in the current session. During compilation, other CNs cannot view the temporary table of the current CN. Therefore, if there are multiple CNs, the system displays a message indicating that the temporary table does not exist. • GaussDB does not support *view%rowtype* and **schema.***view%rowtype* as the input and output parameter types. • GaussDB does not support the **package.***cursor%rowtype* as the input and output parameter types.

2.10.2 Data Type Compatibility

Table 2-51 Other PL/SQL data types

No.	Oracle Database	GaussDB	Difference
1	CHARACTER	Supported, with differences.	• GaussDB: The length ranges from 1 to 10485760 bytes. • Oracle Database: The length ranges from 1 to 32767 bytes.
2	VARCHAR	Supported, with differences.	• GaussDB: The length ranges from 1 to 10485760 bytes. • Oracle Database: The length ranges from 1 to 32767 bytes.
3	STRING	Not supported.	-
4	PLS_INTEGER	Not supported.	In GaussDB, you can use the INT type instead.
5	BINARY_INTEGER	Supported.	-

2.10.3 Control Statements

Table 2-52 Conditional statements

No.	Oracle Database	GaussDB
1	IF THEN	Supported.
2	IF THEN ELSE	Supported.
3	IF THEN ELSIF	Supported.

No.	Oracle Database	GaussDB
4	**simple CASE:** CASE selector WHEN selector_value_1 THEN statements_1 WHEN selector_value_2 THEN statements_2 ... WHEN selector_value_n THEN statements_n [ELSE else_statements END CASE;]	Supported.
5	**searched CASE:** CASE WHEN condition_1 THEN statements_1 WHEN condition_2 THEN statements_2 ... WHEN condition_n THEN statements_n [ELSE else_statements END CASE;]	Supported.

Table 2-53 LOOP statements

No.	Oracle Database	GaussDB
1	[label] LOOP statements END LOOP [label];	Supported.
2	EXIT;	Supported.
3	EXIT WHEN;	Supported.
4	CONTINUE;	Supported.
5	CONTINUE WHEN;	Supported.

Table 2-54 FOR LOOP statements

No.	Oracle Database	GaussDB	Difference
1	[label] FOR index IN [REVERSE] lower_bound..upper_bound LOOP statements END LOOP [label];	Supported, with differences.	When the keyword REVERSE is used in GaussDB, the lower bound must be greater than or equal to the upper bound; otherwise, the loop body is not executed.
2	EXIT WHEN;	Supported.	-
3	CONTINUE WHEN;	Supported.	-

Table 2-55 WHILE LOOP statement

No.	Oracle Database	GaussDB
1	[label] WHILE condition LOOP statements END LOOP [label];	Supported.

Table 2-56 GOTO statement

No.	Oracle Database	GaussDB
1	GOTO	Supported.

Table 2-57 NULL statement

No.	Oracle Database	GaussDB
1	NULL	Supported.

2.10.4 Collections and Records

Table 2-58 Types

No.	Oracle Database	GaussDB
1	Associative array (or index-by table)	Supported.
2	VARRAY (variable-size array)	Supported.

No.	Oracle Database	GaussDB
3	Nested table	Supported.
4	record	Supported.

Table 2-59 Syntax

No.	Oracle Database	GaussDB	Difference
1	Associative array (or index-by table) syntax: TABLE OF datatype [NOT NULL] INDEX BY { PLS_INTEGER | BINARY_INTEGER | VARCHAR2 (v_size) | data_type }	Supported, with differences.	• GaussDB does not support the PLS_INTEGER type. In GaussDB, the value of **data_type** can be a base data type or a record type, collection type, or array type defined in a stored procedure. The ref cursor type is not supported. • In GaussDB, NOT NULL does not take effect in the syntax. That is, the system does not check whether an element is NULL. • For details, see "Stored Procedure > Arrays, Collections, and Records > Collections" in *Developer Guide*.
2	VARRAY (variable-size array) syntax: { VARRAY | [VARYING] ARRAY } (size_limit) OF datatype [NOT NULL]	Supported, with differences.	• GaussDB does not support the NOT NULL syntax. • In GaussDB, **datatype** cannot be set to **varray** (varray cannot be nested). • To make the size_limit function take effect, enable the **varray_compat** parameter in the GUC parameter **behavior_compat_options**. • For details, see "Stored Procedure > Arrays, Collections, and Records > Arrays" in *Developer Guide*.

No.	Oracle Database	GaussDB	Difference
3	Nested table syntax: TABLE OF datatype [NOT NULL]	Supported, with differences.	• In GaussDB, NOT NULL does not take effect in the syntax. • For details, see "Stored Procedure > Arrays, Collections, and Records > Collections" in *Developer Guide*.
4	record syntax: TYPE record_type IS RECORD (field_definition [, field_definition]...) ;	Supported.	• Record columns can be defined as NOT NULL or a default value can be specified. If the record type is nested in other types, the default value and NOT NULL of the record type do not take effect. If a record variable is created using the package.record_type access type, the default value and NOT NULL of the record variable do not take effect. • For details, see "Stored Procedure > Arrays, Collections, and Records > Records" in *Developer Guide*.

Table 2-60 Constructor

No.	Oracle Database	GaussDB
1	collection_type ([value [, value]...])	Supported.

Table 2-61 Variable assignment

No.	Oracle Database	GaussDB	Difference
1	Associative array (or index-by table)	Supported.	-

No.	Oracle Database	GaussDB	Difference
2	VARRAY (variable-size array)	Supported, with differences.	• Values of different VARRAY data in GaussDB can be assigned to each other, depending on whether elements in the data can be implicitly converted to each other. • For details, see "Stored Procedure > Arrays, Collections, and Records > Arrays" in *Developer Guide*.
3	Nested table	Supported.	-
4	record	Supported, with differences.	• Values of different record data in GaussDB can be assigned to each other, depending on whether columns can be implicitly converted. • For details, see "Stored Procedure > Arrays, Collections, and Records > Records" in *Developer Guide*.

Table 2-62 Collection operators

No.	Oracle Database	GaussDB	Difference
1	=	Supported, with differences.	• Oracle Database: The sequence of collection members is ignored during comparison. • GaussDB: The comparison is performed strictly based on the sequence of collection members.
2	<>	Supported, with differences.	• Oracle Database: The sequence of collection members is ignored during comparison. • GaussDB: The comparison is performed strictly based on the sequence of collection members.
3	IS[NOT] NULL	Supported.	-

No.	Oracle Database	GaussDB	Difference
4	^=	Supported, with differences.	• Oracle Database: The sequence of collection members is ignored during comparison. • GaussDB: The comparison is performed strictly based on the sequence of collection members.
5	~=	Not supported.	-
6	IS[NOT] A SET	Not supported.	-
7	IS [NOT] EMPTY	Not supported.	-
8	expr [NOT] MEMBER [OF] nested_table	Not supported.	-
9	nested_table1 [NOT] SUBMULTISET [OF] nested_table2	Not supported.	-
10	[NOT] IN	Supported.	• Oracle Database: The sequence of collection members is ignored during comparison. • GaussDB: The comparison is performed strictly based on the sequence of collection members.

Table 2-63 MULTISET functions

No.	Oracle Database	GaussDB
1	MULTISET UNION [ALL | DISTINCT]	Supported.
2	MULTISET EXCEPT [ALL | DISTINCT]	Supported.
3	MULTISET INTERSECT [ALL | DISTINCT]	Supported.

Table 2-64 Collection type functions

No.	Oracle Database	GaussDB	Difference
1	exists(idx)	Supported.	-
2	extend[(count[, idx])]	Supported, with differences.	GaussDB supports only nested tables.
3	delete[(idx1[, idx2])]	Supported.	-
4	trim[(n)]	Supported, with differences.	GaussDB supports only nested tables.
5	count	Supported.	-
6	first	Supported.	-
7	last	Supported.	-
8	prior(idx)	Supported.	-
9	next(idx)	Supported.	-
10	limit	Supported, with differences.	GaussDB supports only nested tables.

Table 2-65 Record variable operations

No.	Oracle Database	GaussDB
1	Constructors	Supported.
2	%ROWTYPE to declare a variable	Supported.
3	Defining constants	Not supported.

Table 2-66 Collection-related functions

No.	Oracle Database	GaussDB	Difference
1	unnest_table(anynesttable)	Supported.	-
2	unnest_table(anyindexbytable)	Supported.	-
3	table(anyarray)	Not supported.	GaussDB uses the unnest (anyarray) function for equivalent rewriting.

2.10.5 Static SQL Statements

Table 2-67 Static query SQL statements

No.	Oracle Database	GaussDB	Difference
1	SELECT	Supported, with differences.	GaussDB and Oracle Database are different in some scenarios. In GaussDB, FOR SHARE adds a shared lock to the retrieved rows. The shared locks of different transactions do not block each other. If data is locked by FOR SHARE in one transaction and SELECT FOR SHARE SKIP LOCKED is used in another transaction, SKIP LOCKED does not skip the lock.

Table 2-68 Static DML SQL statements

No.	Oracle Database	GaussDB	Difference
1	INSERT	Supported, with differences.	Oracle Database allows the number of columns in the target table to be greater than the number of columns in the subquery result. However, you must explicitly specify the names of columns to be inserted to ensure that the number of columns matches. In GaussDB, you can omit the names of columns to be inserted. In this case, the value of the first column in the subquery result is inserted into the first column of the target table, and so on. If the target table has more columns, **NULL** (if a column allows NULL) or the default value (if any) is inserted into each column.
2	UPDATE	Supported.	-
3	DELETE	Supported.	-
4	MERGE	Supported.	-
5	LOCK TABLE	Supported.	-

No.	Oracle Database	GaussDB	Difference
6	INSERT ALL	Supported, with differences.	• Oracle Database does not support alias setting for the tables of into_clause, but GaussDB supports. • When into_clause specifies the sequence: – Oracle Database: If nextval is referenced for the first time, the next number of the current value is generated. Otherwise, the same number will always be returned. – GaussDB: The number generated by referencing nextval can increment automatically. • The plan_hint statement can take effect in Oracle Database but does not take effect in GaussDB. • Oracle Database allows the number of columns in the target table to be greater than the number of columns in the subquery result. However, you must explicitly specify the names of columns to be inserted to ensure that the number of columns matches. In GaussDB, you can omit the names of columns to be inserted. In this case, the value of the first column in the subquery result is inserted into the first column of the target table, and so on. If the target table has more columns, **NULL** (if a column allows NULL) or the default value (if any) is inserted into each column.

Table 2-69 Static TCL SQL statements

No.	Oracle Database	GaussDB	Difference
1	COMMIT	Supported.	-
2	ROLLBACK	Supported.	-
3	SAVEPOINT	Supported.	-

No.	Oracle Database	GaussDB	Difference
4	SET TRANSACTION	Supported, with differences.	GaussDB does not support the NAME string and USE ROLLBACK SEGMENT rollback_segment syntax.

Table 2-70 Pseudocolumns

No.	Oracle Database	GaussDB	Difference
1	CURRVAL and NEXTVAL	Supported.	-
2	LEVEL	Not supported.	-
3	OBJECT_VALUE	Not supported.	-
4	ROWID	Not supported.	-
5	ROWNUM	Supported, with differences.	It is not recommended that the ROWNUM condition be used in the JOIN ON clause. In GaussDB, when the ROWNUM condition is used in the JOIN ON clause, the behavior in the LEFT JOIN, RIGHT JOIN, FULL JOIN, and MERGE INTO scenarios is different from that in other databases, causing risks in service migration.

Table 2-71 Implicit cursor attributes

No.	Oracle Database	GaussDB	Difference
1	SQL%FOUND	Supported, with differences.	GaussDB does not update the implicit cursor result after COMMIT or ROLLBACK. Oracle Database updates the implicit cursor result after COMMIT or ROLLBACK.
2	SQL %NOTFOUND	Supported, with differences.	

No.	Oracle Database	GaussDB	Difference
3	SQL %ROWCOUNT	Supported, with differences.	
4	SQL%ISOPEN	Supported, with differences.	
5	SQL %BULK_ROWCOUNT	Not supported.	
6	SQL %BULK_EXCEPTIONS	Not supported.	

Table 2-72 Explicit cursor syntax and keywords

No.	Oracle Database	GaussDB	Difference
1	CURSOR cursor_name [parameter_list] RETURN return_type;	Supported.	-
2	CURSOR cursor_name [parameter_list] [RETURN return_type] IS select_statement;	Supported.	-
3	OPEN	Supported.	-
4	CLOSE	Supported, with differences.	GaussDB is automatically closed in the exception, but Oracle Database is not automatically closed in the exception.
5	FETCH	Supported.	-
6	CURRENT OF CURSOR	Supported.	-

Table 2-73 Explicit cursor attributes

No.	Oracle Database	GaussDB
1	SQL%FOUND	Supported.
2	SQL%NOTFOUND	Supported.

No.	Oracle Database	GaussDB
3	SQL%ROWCOUNT	Supported.
4	SQL%ISOPEN	Supported.

Table 2-74 Cursor loop

No.	Oracle Database	GaussDB
1	FOR LOOP	Supported, with differences. In the FORALL+BULK COLLECT INTO scenario, the *INTO* variable returns only the execution result of a single DML statement in GaussDB, and returns the accumulated execution result of DML statements in Oracle Database.

Table 2-75 Scenarios supported by autonomous transactions

No.	Oracle Database	GaussDB
1	Stored procedures	Supported.
2	Anonymous blocks	Supported.
3	Functions	Supported.
4	Packages	Supported.

2.10.6 Dynamic SQL Statements

Table 2-76 Dynamic SQL statement execution modes

No.	Oracle Database	GaussDB	Difference
1	EXECUTE IMMEDIATE	Supported, with differences.	• GaussDB uses the **dynamic_sql_compat** parameter to determines whether variables with the same name read the same parameter and check whether the input and output parameter types of the bound parameters are the same as those of the statement parameters when the stored procedure is called. • GaussDB does not support scenarios where some bound parameters in anonymous blocks are called. For example, when dynamic statements are nested in anonymous blocks, expressions are used to bind parameters. For details, see "Stored Procedure > Dynamic Statements > Dynamically Calling Anonymous Blocks" in *Developer Guide*. • GaussDB does not support RETURNING or RETURN INTO.
2	OPEN FOR, FETCH, CLOSE	Supported.	GaussDB uses the **dynamic_sql_compat** parameter to determines whether variables with the same name read the same parameter and check whether the input and output parameter types of the bound parameters are the same as those of the statement parameters when the stored procedure is called.

2.10.7 Triggers

Table 2-77 Trigger types

No.	Oracle Database	GaussDB	Difference
1	DML TRIGGER	Supported, with differences.	GaussDB: Compound DML triggers are not supported.

No.	Oracle Database	GaussDB	Difference
2	SYSTEM TRIGGER	Not supported.	-

Table 2-78 CREATE triggers

No.	Oracle Database	GaussDB	Difference
1	CREATE syntaxes: CREATE [OR REPLACE] [EDITIONABLE | NONEDITIONABLE] TRIGGER plsql_trigger_source	Supported, with differences.	GaussDB does not support OR REPLACE or EDITIONABLE | NONEDITIONABLE, but supports some behaviors of plsql_trigger_source.
2	plsql_trigger_source ::= syntax: [schema.] trigger_name [sharing_clause] [default_collation_clause] { simple_dml_trigger | instead_of_dml_trigger | compound_dml_trigger | system_trigger }	Supported, with differences.	GaussDB: The schema, sharing_clause, and default_collation_clause are not supported.
3	simple_dml_trigger ::= syntax: { BEFORE | AFTER } dml_event_clause [referencing_clause] [FOR EACH ROW] [trigger_edition_clause] [trigger_ordering_clause] [ENABLE | DISABLE] [WHEN (condition)] trigger_body	Supported, with differences.	GaussDB does not support referencing_clause, referencing_clause (instead, from referencing_table is used), trigger_edition_clause, trigger_ordering_clause, and ENABLE | DISABLE, but supports some behaviors of trigger_body. No error is reported in GaussDB when a statement-level BEFORE/AFTER TRIGGER is created in a view without INSTEAD OF TRIGGER. An error is reported when DML is executed.

No.	Oracle Database	GaussDB	Difference
4	dml_event_clause ::= syntax: { DELETE | INSERT | UPDATE [OF column [, column]...] } [OR { DELETE | INSERT | UPDATE [OF column [, column]...] }... ON [schema.] { table | view }	Not supported.	-
5	trigger_body ::= syntax: { plsql_block | CALL routine_clause }	Supported, with differences.	GaussDB: The plsql_block is not supported. A function can be executed only in the EXECUTE PROCEDURE function_name (arguments); format. In addition, the function needs to be defined by users and declared that it does not contain parameters and the return type is trigger. It is executed when a trigger is triggered.
6	instead_of_dml_trigger ::= syntax: INSTEAD OF { DELETE | INSERT | UPDATE } [OR { DELETE | INSERT | UPDATE }]... ON [NESTED TABLE nested_table_column OF] [schema.] noneditioning_view [referencing_clause] [FOR EACH ROW] [trigger_edition_clause] [trigger_ordering_clause] [ENABLE | DISABLE] trigger_body	Supported, with differences.	GaussDB: The NESTED TABLE nested_table_column OF, referencing_clause, trigger_edition_clause, trigger_ordering_clause, and ENABLE | DISABLE are not supported.

No.	Oracle Database	GaussDB	Difference
7	compound_dml_trigger ::= syntax: CREATE trigger FOR dml_event_clause ON view COMPOUND TRIGGER INSTEAD OF EACH ROW IS BEGIN statement; END INSTEAD OF EACH ROW;	Not supported.	-
8	system_trigger ::= syntax: { BEFORE | AFTER | INSTEAD OF } { ddl_event [OR ddl_event]... | database_event [OR database_event]... } ON { [schema.] SCHEMA | [PLUGGABLE] DATABASE } [trigger_ordering_clause] [ENABLE | DISABLE] trigger_body	Not supported.	-

Table 2-79 ALTER trigger

No.	Oracle Database	GaussDB	Difference
1	ALTER TRIGGER [schema.] trigger_name { trigger_compile_clause | { ENABLE | DISABLE } | RENAME TO new_name | { EDITIONABLE | NONEDITIONABLE } } ;	Supported, with differences.	GaussDB: The schema, trigger_compile_clause, { ENABLE | DISABLE }, and { EDITIONABLE | NONEDITIONABLE } are not supported.

Table 2-80 drop trigger

No.	Oracle Database	GaussDB	Difference
1	DROP TRIGGER [schema.] trigger ;	Supported, with differences.	GaussDB does not support schemas. You need to add **ON table_name** to the end of **trigger_name**.

The *_TRIGGERS views in Oracle Database collect information about triggers. The views in GaussDB are different from those in Oracle Database. For details, see sections "DB_TRIGGERS", "ADM_TRIGGERS", and "MY_TRIGGERS" in "System Catalogs and System Views > System Views > Other System Views" in *Developer Guide*.

Table 2-81 Compatibilities of nested, package, and standalone subprograms

No.	Oracle Database	GaussDB	Difference
1	Nested subprograms (subblocks)	Supported, with differences.	GaussDB does not support overloading. GaussDB does not support the definition of autonomous transactions. GaussDB does not support SETOF. Only one qualifier can reference nested subprograms or variables of nested subprograms.
2	Package subprograms	Supported.	-
3	Standalone subprograms (including functions and procedures)	Supported.	-
4	Anonymous blocks	Supported.	-

Table 2-82 RETURN statements

No.	Oracle Database	GaussDB
1	Functions	Supported.
2	Procedures	Supported.
3	Anonymous blocks	Supported.

Table 2-83 Function-related parameters

No.	Oracle Database	GaussDB	Difference
1	DETERMINISTIC	Supported, with differences.	In GaussDB, it is IMMUTABLE.
2	PARALLEL_ENABLE	Not supported.	-
3	PIPELINED	Not supported.	-
4	RESULT_CACHE	Not supported.	-

Table 2-84 Parameter formats

No.	Oracle Database	GaussDB
1	IN	Supported.
2	OUT	Supported.
3	IN OUT	Supported.

Table 2-85 CREATE statements

No.	Oracle Database	GaussDB	Difference
1	CREATE FUNCTION	Supported, with differences.	GaussDB does not support the IF NOT EXISTS syntax, sharing_clause, or keywords [EDITIONABLE | NONEDITIONABLE]. Only some clauses that specify the **function** attribute are supported (only the invoker_rights_clause clause is supported). For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE FUNCTION" in *Developer Guide*.
2	CREATE LIBRARY	Not supported.	-

No.	Oracle Database	GaussDB	Difference
3	CREATE PACKAGE	Supported, with differences.	GaussDB does not support the IF NOT EXISTS syntax, sharing_clause, or keywords [EDITIONABLE | NONEDITIONABLE]. Only some clauses that specify the **package** attribute are supported (only the invoker_rights_clause clause is supported). For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE PACKAGE" in *Developer Guide*.
4	CREATE PACKAGE BODY	Supported, with differences.	GaussDB does not support the IF NOT EXISTS syntax, sharing_clause, or keywords [EDITIONABLE | NONEDITIONABLE]. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE PACKAGE" in *Developer Guide*.
5	CREATE PROCEDURE	Supported, with differences.	GaussDB does not support the IF NOT EXISTS syntax, sharing_clause, or keywords [EDITIONABLE | NONEDITIONABLE]. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE PROCEDURE" in *Developer Guide*.
6	CREATE TRIGGER	Supported, with differences.	For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE TRIGGER" in *Developer Guide*.
7	CREATE TYPE	Supported, with differences.	GaussDB does not support the varray, object type, and UNDER syntax. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE TYPE" in *Developer Guide*.
8	CREATE TYPE BODY	Not supported.	-

Table 2-86 ALTER statements

No.	Oracle Database	GaussDB	Difference
1	ALTER FUNCTION	Supported, with differences.	GaussDB does not support keywords [EDITIONABLE | NONEDITIONABLE], REUSE, SETTINGS, and DEBUG. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > A > ALTER FUNCTION" in *Developer Guide*.
2	ALTER LIBRARY	Not supported.	-
3	ALTER PACKAGE	Supported, with differences.	GaussDB does not support keywords [EDITIONABLE | NONEDITIONABLE], REUSE, SETTINGS, and DEBUG. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > A > ALTER PACKAGE" in *Developer Guide*.
4	ALTER PROCEDURE	Supported, with differences.	GaussDB does not support keywords [EDITIONABLE | NONEDITIONABLE], REUSE, SETTINGS, and DEBUG.
5	ALTER TRIGGER	Supported, with differences.	In GaussDB, only the trigger name can be modified. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > A > ALTER TRIGGER" in *Developer Guide*.
6	ALTER TYPE	Supported, with differences.	GaussDB supports only some statements. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > A > ALTER TYPE" in *Developer Guide*.

Table 2-87 DROP statements

No.	Oracle Database	GaussDB	Difference
1	DROP FUNCTION	Supported.	-
2	DROP LIBRARY	Not supported.	-
3	DROP PACKAGE	Supported.	-

No.	Oracle Database	GaussDB	Difference
4	DROP PROCEDURE	Supported.	-
5	DROP TRIGGER	Supported, with differences.	The syntax of GaussDB is different from that of Oracle Database. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > D > DROP TRIGGER" in *Developer Guide*.
6	DROP TYPE	Supported, with differences.	GaussDB does not support keywords FORCE and VALIDATE. For details about GaussDB syntax, see "SQL Reference > SQL Syntax > D > DROP TYPE" in *Developer Guide*.
7	DROP TYPE BODY	Not supported.	-

Table 2-88 Keywords related to functions, procedures, and anonymous blocks

No.	Oracle Database	GaussDB	Difference
1	ACCESSIBLE BY	Not supported.	-
2	AGGREGATE	Supported, with differences.	• GaussDB does not support Oracle Database's aggregate using [schema.] implementation_type. • For details about GaussDB syntax, see "SQL Reference > SQL Syntax > C > CREATE AGGREGATE" in *Developer Guide*. The syntax is different, but the implementation functions are the same.
3	DETERMINISTIC	Supported, with differences.	GaussDB supports the keyword DETERMINISTIC only in syntax, but does not support this function.
4	PIPE ROW	Not supported.	-
5	PIPELINED	Not supported.	-
6	SQL_MACRO	Not supported.	-

No.	Oracle Database	GaussDB	Difference
7	RESTRICT_REFERENCES	Not supported.	-
8	INLINE	Not supported.	-

Table 2-89 Keywords related to exception handling

No.	Oracle Database	GaussDB	Difference
1	EXCEPTION_INIT	Supported, with differences.	Binding with system error codes is not supported in GaussDB.
2	Exception	Supported.	-
3	Exception Handler	Supported.	-
4	SQLCODE	Supported.	-
5	SQLERRM	Supported.	-

Table 2-90 Other PL/SQL keywords

No.	Oracle Database	GaussDB
1	COVERAGE	Not supported.
2	COLLATION	Supported.
3	DEPRECATE	Not supported.
4	FORALL	Supported.
5	NOCOPY	Not supported.
6	RETURNING	Supported.
7	SERIALLY_REUSABLE	Not supported.
8	SHARING	Not supported.
9	BULK COLLECT	Supported.

2.11 System Functions

Compatible functions are classified into single-row functions, user-defined functions, aggregate functions, analytic functions, object reference functions, model functions, and OLAP functions.

2.11.1 Single-Row Functions

No.	Oracle Database	GaussDB
1	**Numeric functions**	Supported, with differences.
2	**Character functions returning character values**	Supported, with differences.
3	**Character functions returning number values**	Supported, with differences.
4	Character set functions	Not supported.
5	Collation functions	Not supported.
6	**Datetime functions**	Supported, with differences.
7	**General comparison functions**	Supported, with differences.
8	**Conversion functions**	Supported, with differences.
9	**Large object functions**	Supported, with differences.
10	Collection functions	Not supported.
11	**Hierarchical functions**	Supported.
12	Data mining functions	Not supported.
13	**XML functions**	Supported, with differences.
14	JSON functions	Not supported.
15	**Encoding and decoding functions**	Supported, with differences.
16	**Null-related functions**	Supported.
17	**Environment and identifier functions**	Supported, with differences.

Table 2-91 Numeric functions

No.	Oracle Database	GaussDB	Difference
1	ABS	Supported.	-

No.	Oracle Database	GaussDB	Difference
2	ACOS	Supported.	-
3	ASIN	Supported.	-
4	ATAN	Supported.	-
5	ATAN2	Supported.	-
6	BITAND	Supported.	-
7	CEIL	Supported.	-
8	COS	Supported.	-
9	COSH	Supported.	-
10	EXP	Supported.	-
11	FLOOR	Supported.	-
12	LN	Supported.	-
13	LOG	Supported.	-
14	MOD	Supported, with differences.	• The return types are different. In Oracle Database, the return types include BINARY_DOUBLE, BINARY_FLOAT, and NUMBER. In GaussDB, the return types include INT2, INT4, INT8, and NUMERIC. • If the first input parameter is of the NUMERIC type, the second parameter must be of the INT or NUMERIC type or can be converted to the NUMERIC type. If **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s6**, and the first parameter is of the TEXT type that can be converted to NUMERIC, the second parameter must be of the INT4 type or a type with the value range smaller than INT4.
15	NANVL	Supported, with differences.	GaussDB: NaN cannot be obtained by directly declaring or dividing a floating-point number by 0.
16	POWER	Supported.	-

No.	Oracle Database	GaussDB	Difference
17	REMAINDER	Supported, with differences.	The data types of the return values are different. GaussDB: ● If one input is of the FLOAT4 type and the other is of the NUMERIC type, values of the FLOAT4 type is returned. ● If both inputs are of the FLOAT4 type, values of the FLOAT4 type is returned. ● If both inputs are of the FLOAT8 type, values of the FLOAT8 type are returned. ● For other data types, values of the NUMERIC type are returned. Oracle Database: The return type is NUMBER.
18	ROUND	Supported, with differences.	● For the FLOAT type of the first parameter **n**, the precision of GaussDB is lower than that of Oracle. ● The returned types are different. If round(n, integer) is used, Oracle Database returns the NUMBER type, and GaussDB returns the NUMERIC type. If round(n) is used, Oracle Database returns the data type of *n*, and GaussDB can return only the FLOAT8 or NUMERIC types, but cannot return FLOAT4. ● The logic for the GaussDB to determine that the input parameter is null and the execution framework to return null is different from that in Oracle Database. `SELECT round(NULL,'q');` Oracle Database reports null, and GaussDB reports the error: invalid input syntax for integer: "q".
19	SIGN	Supported.	-
20	SIN	Supported.	-
21	SINH	Supported.	-
22	SQRT	Supported.	-
23	TAN	Supported.	-

No.	Oracle Database	GaussDB	Difference
24	TANH	Supported, with differences.	The data types of the return values are different. GaussDB: ● If the input is of the FLOAT8 type, a value of the FLOAT8 type is returned. ● If the input is of the NUMERIC type, a value of the NUMERIC type is returned. Oracle Database: The return type is NUMBER.
25	TRUNC	Supported.	-
26	WIDTH_BUCKET	Supported.	-

Table 2-92 Character functions returning character values

No.	Oracle Database	GaussDB	Difference
1	CHR	Supported, with differences.	● If the entered number does not comply with the existing character set, GaussDB reports an error in JDBC and Oracle Database returns garbled characters. ● If you enter **0** or **256**, Oracle Database returns characters whose ASCII code is 0, and GaussDB truncates the characters at **'\0;**.
2	CONCAT	Supported.	-
3	INITCAP	Supported, with differences.	The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database.

No.	Oracle Database	GaussDB	Difference
4	LOWER	Supported, with differences.	• The return value types are different. The data types of Oracle Database are the same as the input types. • The time format is implicitly converted. When the time type is entered, the time type is implicitly converted to a character string and then the lower operation is performed. `SELECT LOWER(TO_DATE('2012-12-10','YYYY-MM-DD'));` Oracle Database returns 10-DEC-12, and GaussDB returns 2012-12-10 00:00:00. • The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database.
5	LPAD	Supported.	-
6	LTRIM	Supported, with differences.	The return value types are different. If the input is of the character data type, Oracle Database returns the VARCHAR2 type. If the input is of the national character set specified during database creation, Oracle Database returns the NVARCHAR2 type. If the input is of the LOB type, Oracle Database returns the LOB type. GaussDB returns the TEXT type.
7	NCHR	Supported, with differences.	• The byte length of the return value is different from that of Oracle Database. • The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database. • When the byte array corresponding to the input parameter is returned, a question mark (?) is returned if a single byte ranges from 0x80 to 0xFF. In Oracle Database, a question mark (?) is returned, no output is returned, or an error is reported.

No.	Oracle Database	GaussDB	Difference
8	NLS_LOWER	Supported, with differences.	• The return value types are different. If the input is of the character data type, Oracle Database returns the VARCHAR2 type. If the input is of the LOB type, Oracle Database returns the LOB type. GaussDB returns the TEXT type. • In Oracle Database, the **nlsparam** parameter can be of a type except nls_sort, and no error is reported. GaussDB supports only nls_sort. • The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database.
9	NLS_UPPER	Supported, with differences.	• The return value types are different. If the input is of the character data type, Oracle Database returns the VARCHAR2 type. If the input is of the LOB type, Oracle Database returns the LOB type. GaussDB returns the TEXT type. • In Oracle Database, the **nlsparam** parameter can be of a type except nls_sort, and no error is reported. GaussDB supports only nls_sort. • The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database.
10	NLSSORT	Supported.	-

No.	Oracle Database	GaussDB	Difference

11	REGEXP_REPLACE	Supported, with differences.	• GaussDB input parameter **source_char** does not support the NCLOB type.
			• The meaning of the **'n'** option in the **match_param** input parameter is different. In GaussDB, the **'n'** option has the same meaning as the **'m'** option, indicating that the multi-row matching mode is used. In Oracle Database, it indicates that the dot (.) can match the '\n' character. If this option is not specified, the '\n' character cannot be matched by default. In GaussDB, the dot (.) matches '\n' by default. You do not need to specify the option.
			• The matching results of some regular expressions may be different.
			`SELECT REGEXP_REPLACE('abc01234xyz', '(.*?)(\d+)(.*)', '#', 'g') FROM DUAL;`
			Oracle Database reports an error, and GaussDB returns **#####xyz**.
			• The matching results may be different when Chinese characters are entered in the UTF-8 character set. Oracle Database needs to implement regular expression matching for Chinese character strings in the GBK character set.
			• The matching results of regular expressions that contain some escape characters may be different.
			`SELECT REGEXP_REPLACE('abcabc', '\abc', '#', 'g') FROM DUAL;`
			Oracle Database reports an error, and GaussDB returns **abcabc**.
			• The matching rules are affected by the **aformat_regexp_match** parameter. For details about the affected specifications, see the REGEXP_REPLACE function in "SQL Reference > Functions and Operators > Character Processing Functions and Operators" in *Developer Guide*.

12	REGEXP_SUBSTR	Supported, with differences.	The matching rules are affected by the **aformat_regexp_match** parameter. For details about the affected specifications, see the REGEXP_SUBSTR function in "SQL Reference > Functions and Operators > Character Processing Functions and Operators" in *Developer Guide*.

No.	Oracle Database	GaussDB	Difference
13	REPLACE	Supported.	-
14	RPAD	Supported.	-
15	RTRIM	Supported.	-
16	SUBSTR	Supported.	-
17	TRANSLATE	Supported.	-
18	TRIM	Supported.	-
19	UPPER	Supported, with differences.	• The return value types are different. The data types of Oracle Database are the same as the input types. GaussDB returns the TEXT type. • The time format is implicitly converted. When the time type is entered, the time type is implicitly converted to a character string and then the upper operation is performed. `SELECT UPPER(TO_DATE('2012-12-10','YYYY-MM-DD'));` Oracle Database returns 10-DEC-12, and GaussDB returns 2012-12-10 00:00:00. • The return value is restricted by the database character set. As a result, the returned result is different from that in Oracle Database.
20	INSTRB	Supported.	-

Table 2-93 Character functions returning number values

No.	Oracle Database	GaussDB	Difference
1	ASCII	Supported, with differences.	The return value types are different. The return value type is UINT4 for Oracle and INT4 for GaussDB.
2	INSTR	Supported.	-
3	LENGTH	Supported.	-

No.	Oracle Database	GaussDB	Difference
4	REGEXP_COUNT	Supported, with differences.	• GaussDB input parameter **source_char** does not support the NCLOB type. • The meaning of the **'n'** option in the **match_param** input parameter is different. In GaussDB, the **'n'** option has the same meaning as the **'m'** option, indicating that the multi-row matching mode is used. In Oracle Database, it indicates that the dot (.) can match the '\n' character. If this option is not specified, the '\n' character cannot be matched by default. In GaussDB, the dot (.) matches '\n' by default. You do not need to specify the option. • The matching results of some regular expressions may be different. • The matching results may be different when Chinese characters are entered in the UTF-8 character set. Oracle Database needs to implement regular expression matching for Chinese character strings in the GBK character set. • The matching results of regular expressions that contain some escape characters may be different. • The matching rules are affected by the **aformat_regexp_match** parameter. For details about the affected specifications, see the REGEXP_COUNT function in "SQL Reference > Functions and Operators > Character Processing Functions and Operators" in *Developer Guide*.
5	REGEXP_INSTR	Supported, with differences.	The matching rules are affected by the **aformat_regexp_match** parameter. For details about the affected specifications, see the REGEXP_INSTR function in "SQL Reference > Functions and Operators > Character Processing Functions and Operators" in *Developer Guide*.
6	LENGTHC	Supported.	-

Table 2-94 Datetime functions

No.	Oracle Database	GaussDB	Difference
1	ADD_MONTHS	Supported, with differences.	• From A.D. to B.C., the difference between GaussDB and Oracle Database is one year. • The earliest year can be –4714 in GaussDB and –4713 in Oracle Database.
2	CURRENT_DATE	Supported, with differences.	GaussDB: The **nls_date_format** parameter cannot be used to set the time display format.
3	CURRENT_TIMESTAMP	Supported, with differences.	The value ranges from 0 to 9 in Oracle Database. The value ranges from 0 to 6 in GaussDB. The trailing zeros in microseconds are not displayed.
4	DBTIMEZONE	Supported, with differences.	GaussDB: The timestamp API with the built-in tz cannot be called.
5	EXTRACT	Supported.	-
6	LAST_DAY	Supported, with differences.	The return value types are different.
7	LOCALTIMESTAMP	Supported, with differences.	The value ranges from 0 to 9 in Oracle Database. The value ranges from 0 to 6 in GaussDB. The trailing zeros in microseconds are not displayed.
8	MONTHS_BETWEEN	Supported, with differences.	The input parameter types are different.
9	NEW_TIME	Supported, with differences.	When the first input parameter of the new_time function is a literal, the literal format and the return value type of the function are different from those in Oracle Database.
10	NEXT_DAY	Supported.	-
11	NUMTODSINTERVAL	Supported, with differences.	GaussDB: The dsinterval type is not supported. Currently, interval is used to be compatible with the dsinterval type.

No.	Oracle Database	GaussDB	Difference
12	NUMTOYMINTERVAL	Supported, with differences.	GaussDB: The yminterval type is not supported. Currently, interval is used to be compatible with the yminterval type.
13	SESSIONTIMEZONE	Supported, with differences.	• The assignment syntax is different. In GaussDB, the SET SESSION TIME ZONE 8 syntax is used. In Oracle Database, alter session set time_zone= '+08:00' is used. • The default values are different. GaussDB: The time zone name is displayed, for example, **PRC**. The offset is used in Oracle Database, for example, **+08:00**.
14	SYS_EXTRACT_UTC	Supported.	-
15	SYSDATE	Supported, with differences.	The return value types are different.
16	SYSTIMESTAMP	Supported, with differences.	GaussDB supports only six digits for millisecond calculation, and Oracle Database supports nine digits.
17	TO_CHAR	Supported, with differences.	The fmt '5' is not included in Oracle Database documents and is not adapted.
18	TO_DSINTERVAL	Supported, with differences.	GaussDB: The dsinterval type is not supported. Currently, interval is used to be compatible with the dsinterval type.
19	TO_TIMESTAMP	Supported, with differences.	GaussDB supports only six digits for millisecond calculation, and Oracle Database supports nine digits.
20	TO_TIMESTAMP_TZ	Supported, with differences.	The timestamptz type of GaussDB is equivalent to the timestampwithlocalti-mezone type of Oracle Database. The type corresponding to timestamptz of Oracle Database is missing. The value of **nls_date_language** can only be **ENGLISH** or **AMERICAN**.
21	TO_YMINTERVAL	Supported, with differences.	GaussDB: The yminterval type is not supported. Currently, interval is used to be compatible with the yminterval type.

No.	Oracle Database	GaussDB	Difference
22	TRUNC	Supported, with differences.	The type returned by GaussDB is the same as the type of the first input parameter. Oracle Database always returns the date type. In addition, the supported formats are different in the two databases. For details about the supported formats, see "SQL Reference > Functions and Operators > Date and Time Processing Functions and Operators" in *Developer Guide*.
23	TZ_OFFSET	Supported, with differences.	When a time zone name is received as an input parameter, the types of the time zone name are less than those of Oracle Database.

Table 2-95 General comparison functions

No.	Oracle Database	GaussDB	Difference
1	GREATEST	Supported, with differences.	• GaussDB: The comparison mode specified by the **NLS_SORT** parameter is not supported. Only binary comparison is supported. • GaussDB: Expressions in multiple languages are not supported.
2	LEAST	Supported, with differences.	• GaussDB: The comparison mode specified by the **NLS_SORT** parameter is not supported. Only binary comparison is supported. • GaussDB: Expressions in multiple languages are not supported.

Table 2-96 Conversion functions

No.	Oracle Database	GaussDB	Difference
1	ASCIISTR	Supported.	-
2	CAST	Supported, with differences.	• GaussDB: The MULTISET clause is not supported. • GaussDB: The **nlsparam** parameter is not supported.

No.	Oracle Database	GaussDB	Difference
3	HEXTORAW	Supported.	-
4	RAWTOHEX	Supported.	-
5	TO_BINARY_DOUBLE	Supported, with differences.	GaussDB: The **nlsparam** parameter is not supported.
6	TO_BINARY_FLOAT	Supported, with differences.	GaussDB: The **nlsparam** parameter is not supported.
7	TO_BLOB	Supported, with differences.	• GaussDB: The long raw type is not supported. • GaussDB: The bfile and mime_type types are not supported.
8	TO_CLOB	Supported.	-
9	TO_DATE	Supported, with differences.	• Multi-language parameters are not supported. • The returned types are different. • The control parameter **NLS_DATE_FORMAT** is missing. • Some formats are not supported. • fmt = 'j'. The output before October 15, 1582 in Oracle Database is inconsistent with that in GaussDB. • If there is no separator, the value may be different from that in Oracle Database. Take to_date('220725','yymmdd') as an example. If yy/rr is parsed based on the fixed length 4, the year is parsed as 2207 and the month is parsed as 25. The month 25 is invalid, and an error will be reported.
10	TO_MULTI_BYTE	Supported.	-
11	TO_NCHAR	Supported, with differences.	• GaussDB: The input parameter type is converted to text. • Oracle Database: The input parameter type is converted to the national character set.

No.	Oracle Database	GaussDB	Difference
12	TO_NUMBER	Supported, with differences.	GaussDB does not support the **NLS_PARAM** parameter. The differences between the format options of GaussDB and Oracle Database are as follows: 1. $ GaussDB does not support this format. 2. Comma (,) GaussDB: Commas (,) can appear at any position of format. Oracle Database: • In format, commas can only appear in the integer part and cannot appear at the beginning of a number. In the original data, commas can appear at the beginning of a number. • The number and position of commas in the format can be different from those in the original data, but the position of the last comma must be the same. • Consecutive commas in the original data and the format are equivalent to no comma. • If the original data does not contain commas, the number of digits after the last comma in the format must be the same as that in the original data. 3. B GaussDB does not support this function. 4. C GaussDB does not support the **NLS** parameter. 5. G GaussDB does not support the **NLS** parameter. 6. L GaussDB does not support the **NLS** parameter. 7. U GaussDB does not support the **NLS** parameter.

No.	Oracle Database	GaussDB	Difference
			8. D GaussDB does not support the **NLS** parameter. 9. PR GaussDB: It is equivalent to S. A negative number is returned. Oracle Database: ● Returns the negative value in the angle brackets (< >). ● Returns a positive value with leading and trailing spaces. ● Restriction: PR format elements can only appear at the last position of the digital format model. 10. RN | rn GaussDB does not support this function. TM| TM9 | TMe GaussDB does not support this function. 11. V GaussDB does not support this function. 12. FM In GaussDB, when there is FM, the comma in the format can be more than that in the original data. In other words, the number of commas is not necessarily the same. In Oracle Database, spaces before and after the return value are retained. 13. EEEE GaussDB does not support this function.
13	TO_SINGLE_BYTE	Supported.	-
14	TREAT	Supported, with differences.	In GaussDB, the period (.) operator cannot be used to obtain values, and the values cannot be converted to the OBJECT type.
15	UNISTR	Supported, with differences.	GaussDB supports only UTF-8 encoding. Oracle Database supports UTF-8 and UTF-16 encodings.

Table 2-97 Large object functions

No.	Oracle Database	GaussDB	Difference
1	EMPTY_BLOB	Supported.	-
2	EMPTY_CLOB	Supported, with differences.	GaussDB: The CLOB type does not support the locator concept in Oracle Database.

Table 2-98 Hierarchical functions

No.	Oracle Database	GaussDB
1	SYS_CONNECT_BY_PATH	Not supported.

Table 2-99 XML functions

No.	Oracle Database	GaussDB	Difference
1	EXISTSNODE	Supported, with differences.	If the input parameter has a namespace, aliases must be defined for both the XPath and namespace.
2	EXTRACTVALUE	Supported, with differences.	Currently, only XPath 1.0 is supported.
3	SYS_XMLAGG	Supported, with differences.	This is an alias of xmlagg and can be replaced with xmlagg.
4	XMLAGG	Supported.	-
5	XMLCOMMENT	Supported.	-
6	XMLCONCAT	Supported.	-

No.	Oracle Database	GaussDB	Difference
7	XMLELEMENT	Supported, with differences.	For xmlelement and xmlattributes, when the value of **name** is **NULL**, the database behavior is different from that in Oracle Database. • When the **name** column of **xmlelement** is set to **NULL**, the name information is empty and the attribute information is not displayed. • When the **name** column of **xmlattributes** is set to **NULL**, the attribute information is not displayed.
8	XMLEXISTS	Supported, with differences.	GaussDB input parameter is of the XML type.
9	XMLFOREST	Supported, with differences.	GaussDB return value is of the XML type. GaussDB does not support the EVALNAME syntax.
10	XMLPARSE	Supported, with differences.	GaussDB return value is of the XML type. GaussDB does not support the WELLFORMED syntax.
11	XMLROOT	Supported, with differences.	GaussDB return value is of the XML type.
12	JSON_OBJECT	Supported.	-
13	XMLTABLE	Supported, with differences.	GaussDB: The XPath 1.0 expression is used to select data from the XML file. The default namespace cannot be declared, multiple groups of inputs and aliases cannot be obtained, the passing_clause clause of the input data cannot be omitted, and the RETURNING SEQUENCE BY REF and (SEQUENCE) BY REF clauses are not supported.
14	GETSTRINGVAL	Supported.	-
15	GETCLOBVAL	Supported.	-
16	XMLSEQUENCE	Supported.	-

Table 2-100 Encoding and decoding functions

No.	Oracle Database	GaussDB	Difference
1	DECODE	Supported.	-
2	DUMP	Supported, with differences.	The returned results of the numeric and time types in GaussDB are inconsistent with those in Oracle Database due to different storage formats. In GaussDB, **select dump(123);** returns **Typ=23 Len=4: 123,0,0,0**. In Oracle Database, **select dump(123) from dual;** returns **Typ=2 Len=3: 194,2,24**.
3	ORA_HASH	Supported, with differences.	GaussDB has the following behaviors: • The input parameter of the time type is converted into the character string type and then hashed. • The **maxbucket** parameter is not supported.
4	VSIZE	Supported, with differences.	The returned results of the numeric and time types in GaussDB are inconsistent with those in Oracle Database due to different storage formats. In GaussDB, **select vsize(999);** returns **4**. In Oracle Database, **select vsize(999) from dual;** returns **3**.

Table 2-101 Null-related functions

No.	Oracle Database	GaussDB
1	COALESCE	Supported.
2	LNNVL	Supported.
3	NULLIF	Supported.
4	NVL	Supported.
5	NVL2	Supported.

Table 2-102 Environment and identifier functions

No.	Oracle Database	GaussDB	Difference
1	SYS_CONTEXT	Supported, with differences.	GaussDB returns NULL for unsupported parameters. The following parameters are not supported: • 'action' • 'is_application_root' • 'is_application_pdb' • 'audited_cursorid' • 'authenticated_identity' • 'authentication_data' • 'authentication_method' • 'cdb_domain' • 'cdb_name' • 'client_identifier' • 'con_id' • 'con_name' • 'current_sql_length' • 'db_domain' • 'db_supplemental_log_level' • 'dblink_info' • 'drain_status' • 'entryid' • 'enterprise_identity' • 'fg_job_id' • 'global_uid' • 'identification_type' • 'instance' • 'is_dg_rolling_upgrade' • 'ldap_server_type' • 'module' • 'network_protocol' • 'nls_calendar' • 'nls_sort' • 'nls_territory' • 'oracle_home' • 'os_user' • 'platform_slash'

No.	Oracle Database	GaussDB	Difference
			• 'policy_invoker' • 'proxy_enterprise_identity' • 'proxy_user' • 'proxy_userid' • 'scheduler_job' • 'session_edition_id' • 'session_edition_name' • 'sessionid' • 'statementid' • 'terminal' • 'unified_audit_sessionid' • 'session_default_collation' • 'client_info' • 'bg_job_id' • 'client_program_name' • 'current_bind' • 'global_context_memory' • 'host' • 'current_sqln'
2	SYS_GUID	Supported.	-
3	USER	Supported, with differences.	The return value types are different.

No.	Oracle Database	GaussDB	Difference
4	USERENV	Supported, with differences.	GaussDB returns NULL for unsupported parameters. The following parameters are not supported: ● 'action' ● 'is_application_root' ● 'is_application_pdb' ● 'audited_cursorid' ● 'authenticated_identity' ● 'authentication_data' ● 'authentication_method' ● 'cdb_domain' ● 'cdb_name' ● 'client_identifier' ● 'con_id' ● 'con_name' ● 'current_sql_length' ● 'db_domain' ● 'db_supplemental_log_level' ● 'dblink_info' ● 'drain_status' ● 'entryid' ● 'enterprise_identity' ● 'fg_job_id' ● 'global_uid' ● 'identification_type' ● 'is_dg_rolling_upgrade' ● 'ldap_server_type' ● 'module' ● 'network_protocol' ● 'nls_calendar' ● 'nls_sort' ● 'nls_territory' ● 'oracle_home' ● 'os_user' ● 'platform_slash' ● 'policy_invoker' ● 'proxy_enterprise_identity'

No.	Oracle Database	GaussDB	Difference
			• 'proxy_user' • 'proxy_userid' • 'scheduler_job' • 'session_edition_id' • 'session_edition_name' • 'sessionid' • 'statementid' • 'terminal' • 'unified_audit_sessionid' • 'session_default_collation' • 'client_info' • 'bg_job_id' • 'client_program_name' • 'current_bind' • 'global_context_memory' • 'host' • 'current_sqln'

2.11.2 Other Functions

No.	Oracle Database	GaussDB
1	Aggregate functions	Supported.
2	Analytic functions	Supported.
3	Object reference functions	Not supported.
4	Models functions	Not supported.
5	OLAP functions	Not supported.
6	Data cartridge functions	Not supported.
7	User-defined functions	Supported.

Table 2-103 Aggregate functions

No.	Oracle Database	GaussDB	Difference
1	AVG	Supported.	-

No.	Oracle Database	GaussDB	Difference
2	CORR	Supported.	-
3	COUNT	Supported.	-
4	COVAR_POP	Supported.	-
5	COVAR_SAMP	Supported.	-
6	CUME_DIST	Supported.	-
7	DENSE_RANK	Supported.	-
8	FIRST	Supported.	The KEEP syntax used in GaussDB is compatible with Oracle Database.
9	GROUPING	Supported.	-
10	LAST	Supported.	The KEEP syntax used in GaussDB is compatible with Oracle Database.
11	LISTAGG	Supported.	-
12	MAX	Supported.	-
13	MEDIAN	Supported.	-
14	MIN	Supported.	-
15	PERCENT_RANK	Supported.	-
16	PERCENTILE_CONT	Supported.	-
17	RANK	Supported.	-
18	REGR_ (Linear Regression)	Supported.	-
19	STDDEV	Supported.	-
20	STDDEV_POP	Supported.	-
21	STDDEV_SAMP	Supported.	-
22	SUM	Supported.	-
23	VAR_POP	Supported.	-
24	VAR_SAMP	Supported.	-
25	VARIANCE	Supported.	-
26	WM_CONCAT	Supported.	-

Table 2-104 Analytic functions

No.	Oracle Database	GaussDB	Difference
1	FIRST_VALUE	Supported.	-
2	LAG	Supported.	-
3	LAST_VALUE	Supported.	-
4	LEAD	Supported.	-
5	NTH_VALUE	Supported, with differences.	• Oracle Database: The FROM FIRST|LAST syntax format is supported. • GaussDB: The FROM FIRST|LAST syntax format is not supported.
6	NTILE	Supported.	-
7	ROW_NUMBER	Supported.	-
8	RATIO_TO_REPORT	Supported.	-

2.12 System Views

GaussDB is compatible with some Oracle Database system views. For details, see the following table.

For details about columns in system views, see "System Views" in *Developer Guide*.

Table 2-105 Supported system views

No.	Oracle Database	GaussDB
1	ALL_ALL_TABLES	DB_ALL_TABLES
2	ALL_COL_PRIVS	DB_COL_PRIVS
3	ALL_COLL_TYPES	DB_COLL_TYPES
4	ALL_IND_COLUMNS	DB_IND_COLUMNS
5	ALL_COL_COMMENTS	DB_COL_COMMENTS
6	ALL_CONS_COLUMNS	DB_CONS_COLUMNS
7	ALL_CONSTRAINTS	DB_CONSTRAINTS
8	ALL_DEPENDENCIES	DB_DEPENDENCIES
9	ALL_DIRECTORIES	DB_DIRECTORIES

No.	Oracle Database	GaussDB
10	ALL_IND_EXPRESSIONS	DB_IND_EXPRESSIONS
11	ALL_IND_PARTITIONS	DB_IND_PARTITIONS
12	ALL_INDEXES	DB_INDEXES
13	ALL_IND_SUBPARTITIONS	DB_IND_SUBPARTITIONS
14	ALL_OBJECTS	DB_OBJECTS
15	ALL_PART_COL_STATISTICS	DB_PART_COL_STATISTICS
16	ALL_PART_KEY_COLUMNS	DB_PART_KEY_COLUMNS
17	ALL_PART_TABLES	DB_PART_TABLES
18	ALL_SCHEDULER_JOB_ARGS	DB_SCHEDULER_JOB_ARGS
19	ALL_SCHEDULER_PROGRAM_ARGS	DB_SCHEDULER_PROGRAM_ARGS
20	ALL_SEQUENCES	DB_SEQUENCES
21	ALL_SUBPART_KEY_COLUMNS	DB_SUBPART_KEY_COLUMNS
22	ALL_SYNONYMS	DB_SYNONYMS
23	ALL_TAB_COL_STATISTICS	DB_TAB_COL_STATISTICS
24	ALL_TAB_COMMENTS	DB_TAB_COMMENTS
25	ALL_TAB_HISTOGRAMS	DB_TAB_HISTOGRAMS
26	ALL_TAB_STATS_HISTORY	DB_TAB_STATS_HISTORY
27	ALL_TYPES	DB_TYPES
28	ALL_PROCEDURES	DB_PROCEDURES
29	ALL_SOURCE	DB_SOURCE
30	ALL_TAB_COLUMNS	DB_TAB_COLUMNS
31	ALL_TAB_PARTITIONS	DB_TAB_PARTITIONS
32	ALL_TAB_SUBPARTITIONS	DB_TAB_SUBPARTITIONS
33	ALL_TABLES	DB_TABLES
34	ALL_TRIGGERS	DB_TRIGGERS
35	ALL_USERS	DB_USERS
36	ALL_VIEWS	DB_VIEWS
37	DBA_AUDIT_OBJECT	ADM_AUDIT_OBJECT

No.	Oracle Database	GaussDB
38	DBA_AUDIT_SESSION	ADM_AUDIT_SESSION
39	DBA_AUDIT_STATEMENT	ADM_AUDIT_STATEMENT
40	DBA_AUDIT_TRAIL	ADM_AUDIT_TRAIL
41	DBA_COL_COMMENTS	ADM_COL_COMMENTS
42	DBA_COL_PRIVS	ADM_COL_PRIVS
43	DBA_COLL_TYPES	ADM_COLL_TYPES
44	DBA_ARGUMENTS	ADM_ARGUMENTS
45	DBA_CONSTRAINTS	ADM_CONSTRAINTS
46	DBA_DATA_FILES	ADM_DATA_FILES
47	DBA_CONS_COLUMNS	ADM_CONS_COLUMNS
48	DBA_DEPENDENCIES	ADM_DEPENDENCIES
49	DBA_DIRECTORIES	ADM_DIRECTORIES
50	DBA_PART_COL_STATISTICS	ADM_PART_COL_STATISTICS
51	DBA_PART_TABLES	ADM_PART_TABLES
52	DBA_ROLE_PRIVS	ADM_ROLE_PRIVS
53	DBA_ROLES	ADM_ROLES
54	DBA_SCHEDULER_JOB_ARGS	ADM_SCHEDULER_JOB_ARGS
55	DBA_SCHEDULER_PROGRAMS	ADM_SCHEDULER_PROGRAMS
56	DBA_SCHEDULER_PROGRAM_ARGS	ADM_SCHEDULER_PROGRAM_ARGS
57	DBA_HIST_SNAPSHOT	ADM_HIST_SNAPSHOT
58	DBA_HIST_SQL_PLAN	ADM_HIST_SQL_PLAN
59	DBA_HIST_SQLSTAT	ADM_HIST_SQLSTAT
60	DBA_HIST_SQLTEXT	ADM_HIST_SQLTEXT
61	DBA_ILMDATAMOVEMENTPOLICIES	GS_ADM_ILMDATAMOVEMENTPOLICIES
62	DBA_ILMEVALUATIONDE-TAILS	GS_ADM_ILMEVALUATIONDETAILS
63	DBA_ILMOBJECTS	GS_ADM_ILMOBJECTS

No.	Oracle Database	GaussDB
64	DBA_ILMPARAMETERS	GS_ADM_ILMPARAMETERS
65	DBA_ILMPOLICIES	GS_ADM_ILMPOLICIES
66	DBA_ILMRESULTS	GS_ADM_ILMRESULTS
67	DBA_ILMTASKS	GS_ADM_ILMTASKS
68	DBA_IND_COLUMNS	ADM_IND_COLUMNS
69	DBA_IND_EXPRESSIONS	ADM_IND_EXPRESSIONS
70	DBA_IND_PARTITIONS	ADM_IND_PARTITIONS
71	DBA_INDEXES	ADM_INDEXES
72	DBA_OBJECTS	ADM_OBJECTS
73	DBA_PART_INDEXES	ADM_PART_INDEXES
74	DBA_PROCEDURES	ADM_PROCEDURES
75	DBA_SCHEDULER_JOBS	ADM_SCHEDULER_JOBS
76	DBA_SCHEDULER_RUNNING_JOBS	ADM_SCHEDULER_RUNNING_JOBS
77	DBA_SEGMENTS	ADM_SEGMENTS
78	DBA_SEQUENCES	ADM_SEQUENCES
79	DBA_SOURCE	ADM_SOURCE
80	DBA_IND_SUBPARTITIONS	ADM_IND_SUBPARTITIONS
81	DBA_SUBPART_KEY_COLUMNS	ADM_SUBPART_KEY_COLUMNS
82	DBA_SYS_PRIVS	ADM_SYS_PRIVS
83	DBA_TAB_COL_STATISTICS	ADM_TAB_COL_STATISTICS
84	DBA_TAB_HISTOGRAMS	ADM_TAB_HISTOGRAMS
85	DBA_TAB_STATISTICS	ADM_TAB_STATISTICS
86	DBA_TAB_STATS_HISTORY	ADM_TAB_STATS_HISTORY
87	DBA_TABLESPACES	ADM_TABLESPACES
88	DBA_TYPES	ADM_TYPES
89	DBA_USERS	ADM_USERS
90	DBA_SYNONYMS	ADM_SYNONYMS
91	DBA_TAB_COLS	ADM_TAB_COLS
92	DBA_TAB_COLUMNS	ADM_TAB_COLUMNS

No.	Oracle Database	GaussDB
93	DBA_TAB_COMMENTS	ADM_TAB_COMMENTS
94	DBA_TABLES	ADM_TABLES
95	DBA_TAB_PARTITIONS	ADM_TAB_PARTITIONS
96	DBA_TAB_SUBPARTITIONS	ADM_TAB_SUBPARTITIONS
97	DBA_TRIGGERS	ADM_TRIGGERS
98	DBA_TYPE_ATTRS	ADM_TYPE_ATTRS
99	DBA_VIEWS	ADM_VIEWS
100	ROLE_ROLE_PRIVS	ROLE_ROLE_PRIVS
101	ROLE_SYS_PRIVS	ROLE_SYS_PRIVS
102	ROLE_TAB_PRIVS	ROLE_TAB_PRIVS
103	USER_COL_COMMENTS	MY_COL_COMMENTS
104	USER_COL_PRIVS	MY_COL_PRIVS
105	USER_COLL_TYPES	MY_COLL_TYPES
106	USER_CONSTRAINTS	MY_CONSTRAINTS
107	USER_DEPENDENCIES	MY_DEPENDENCIES
108	DICT	DICT
109	DICTIONARY	DICTIONARY
110	DUAL	DUAL
111	NLS_DATABASE_PARAMETERS	NLS_DATABASE_PARAMETERS
112	NLS_INSTANCE_PARAMETERS	NLS_INSTANCE_PARAMETERS
113	PLAN_TABLE	PLAN_TABLE
114	USER_ILMDATAMOVEMENTPOLICIES	GS_MY_ILMDATAMOVEMENTPOLICIES
115	USER_ILMEVALUATIONDETAILS	GS_MY_ILMEVALUATIONDETAILS
116	USER_ILMOBJECTS	GS_MY_ILMOBJECTS
117	USER_ILMPOLICIES	GS_MY_ILMPOLICIES
118	USER_ILMRESULTS	GS_MY_ILMRESULTS
119	USER_ILMTASKS	GS_MY_ILMTASKS
120	USER_IND_COLUMNS	MY_IND_COLUMNS

No.	Oracle Database	GaussDB
121	USER_IND_EXPRESSIONS	MY_IND_EXPRESSIONS
122	USER_IND_PARTITIONS	MY_IND_PARTITIONS
123	USER_IND_SUBPARTITIONS	MY_IND_SUBPARTITIONS
124	USER_INDEXES	MY_INDEXES
125	USER_JOBS	MY_JOBS
126	USER_OBJECTS	MY_OBJECTS
127	USER_PART_COL_STATISTICS	MY_PART_COL_STATISTICS
128	USER_PART_INDEXES	MY_PART_INDEXES
129	USER_PART_TABLES	MY_PART_TABLES
130	USER_PROCEDURES	MY_PROCEDURES
131	USER_SCHEDULER_JOB_ARGS	MY_SCHEDULER_JOB_ARGS
132	USER_SCHEDULER_PROGRAM_ARGS	MY_SCHEDULER_PROGRAM_ARGS
133	USER_SEQUENCES	MY_SEQUENCES
134	USER_SOURCE	MY_SOURCE
135	USER_SUBPART_KEY_COLUMNS	MY_SUBPART_KEY_COLUMNS
136	USER_SYNONYMS	MY_SYNONYMS
137	USER_SYS_PRIVS	MY_SYS_PRIVS
138	USER_TAB_COL_STATISTICS	MY_TAB_COL_STATISTICS
139	USER_TAB_COLUMNS	MY_TAB_COLUMNS
140	USER_TAB_COMMENTS	MY_TAB_COMMENTS
141	USER_TAB_HISTOGRAMS	MY_TAB_HISTOGRAMS
142	USER_TAB_PARTITIONS	MY_TAB_PARTITIONS
143	USER_TAB_STATISTICS	MY_TAB_STATISTICS
144	USER_TAB_STATS_HISTORY	MY_TAB_STATS_HISTORY
145	USER_TABLES	MY_TABLES
146	USER_TABLESPACES	MY_TABLESPACES

No.	Oracle Database	GaussDB
147	USER_TRIGGERS	MY_TRIGGERS
148	USER_TYPE_ATTRS	MY_TYPE_ATTRS
149	USER_TYPES	MY_TYPES
150	USER_VIEWS	MY_VIEWS
151	V$NLS_PARAMETERS	V$NLS_PARAMETERS
152	V$SESSION_WAIT	V$SESSION_WAIT
153	V$SYSSTAT	V$SYSSTAT
154	V$SYSTEM_EVENT	V$SYSTEM_EVENT
155	V$VERSION	V$VERSION
156	V$INSTANCE	V_INSTANCE
157	GV$INSTANCE	GV_INSTANCE
158	V$MYSTAT	V_MYSTAT
159	V$SESSION	V_SESSION
160	GV$SESSION	GV_SESSION
161	V$SESSION_LONGOPS	DV_SESSION_LONGOPS
162	V$SESSION	DV_SESSIONS
163	ALL_ARGUMENTS	DB_ARGUMENTS
164	USER_CONS_COLUMNS	MY_CONS_COLUMNS
165	USER_PART_KEY_COLUMNS	MY_PART_KEY_COLUMNS
166	USER_ROLE_PRIVS	MY_ROLE_PRIVS
167	DBA_TAB_PRIVS	ADM_TAB_PRIVS
168	USER_SCHEDULER_JOBS	MY_SCHEDULER_JOBS
169	V$LOCK	V$LOCK
170	V$DBLINK	V$DBLINK
171	V$GLOBAL_TRANSACTION	V$GLOBAL_TRANSACTION
172	V$OPEN_CURSOR	V$OPEN_CURSOR
173	V$GLOBAL_OPEN_CURSOR	V$GLOBAL_OPEN_CURSOR
174	ALL_TAB_PRIVS	DB_TAB_PRIVS

No.	Oracle Database	GaussDB
175	ALL_TAB_MODIFICATIONS	DB_TAB_MODIFICATIONS
176	USER_TAB_MODIFICATIONS	MY_TAB_MODIFICATIONS
177	USER_AUDIT_TRAIL	MY_AUDIT_TRAIL

2.13 Advanced Packages

GaussDB is compatible with some advanced packages in Oracle Database. For details, see the following table.

For more information about the advanced packages, see "Advanced Packages" in *Developer Guide*.

Table 2-106 Supported advanced packages

No.	Oracle Database	GaussDB	Difference
1	**DBMS_LOB**	DBE_LOB	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_LOB" in *Developer Guide*.
2	**DBMS_RANDOM**	DBE_RANDOM	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_RANDOM" in *Developer Guide*.
3	**DBMS_OUTPUT**	DBE_OUTPUT	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_OUTPUT" in *Developer Guide*.
4	**UTL_RAW**	DBE_RAW	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_RAW" in *Developer Guide*.
5	**DBMS_SCHEDULER**	DBE_SCHEDULER	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_SCHEDULER" in *Developer Guide*.

No.	Oracle Database	GaussDB	Difference
6	DBMS_UTILITY	DBE_UTILITY	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_UTILITY" in *Developer Guide*.
7	DBMS_SQL	DBE_SQL	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_SQL" in *Developer Guide*.
8	UTL_FILE	DBE_FILE	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_FILE" in *Developer Guide*.
9	DBMS_SESSION	DBE_SESSION	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_SESSION" in *Developer Guide*.
10	UTL_MATCH	DBE_MATCH	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_MATCH" in *Developer Guide*.
11	DBMS_APPLICATION_INFO	DBE_APPLICATION_INFO	For details about how to use it in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_APPLICATION_INFO" in *Developer Guide*.
12	DBMS_XMLDOM	DBE_XMLDOM	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_XMLDOM" in *Developer Guide*.
13	DBMS_XMLPARSER	DBE_XMLPARSER	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_XMLPARSER" in *Developer Guide*.

No.	Oracle Database	GaussDB	Difference
14	DBMS_ILM	DBE_ILM	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_ILM" in *Developer Guide*.
15	DBMS_ILM_ADMIN	DBE_ILM_ADMIN	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_ILM_ADMIN" in *Developer Guide*.
16	DBMS_COMPRESSION	DBE_COMPRESSION	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_COMPRESSION" in *Developer Guide*.
17	DBMS_HEAT_MAP	DBE_HEAT_MAP	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_HEAT_MAP" in *Developer Guide*.
18	DBMS_DESCRIBE	DBE_DESCRIBE	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_DESCRIBE" in *Developer Guide*.
19	DBMS_XMLGEN	DBE_XMLGEN	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_XMLGEN" in *Developer Guide*.
20	DBMS_STATS	DBE_STATS	For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.

Table 2-107 DBMS_LOB compatibility

No.	Oracle Database	GaussDB	Difference
1	APPEND Procedures	APPEND Procedures	-

No.	Oracle Database	GaussDB	Difference
2	CLOB2FILE Procedure	Not supported.	-
3	CLOSE Procedure	BFILECLOSE Procedure	GaussDB: The parameter type is BFILE, and no function overloading exists. Oracle Database: There are three overloaded procedures. The procedures have three parameters **lob_loc**, **lob_loc**, and **file_loc**, which are of the BLOB, CLOB CHARACTER SET ANY_CS, and BFILE types, respectively.
4	COMPARE Functions	COMPARE Functions	GaussDB: There are three overloaded functions. The third parameter (**len**) is of the BIGINT type. Oracle Database: There are three overloaded functions. The third parameter (**amount**) is of the INTEGER type.
5	CONVERTTOBLOB Procedure	LOB_CONVERTTOBLOB Procedure	GaussDB: This procedure has five parameters, and the third, fourth, and fifth parameters are of the BIGINT type. Oracle Database: This procedure has eight parameters. In addition to all GaussDB parameters, the **blob_csid**, **lang_context**, and **warning** parameters are added, which are of the NUMBER, INTEGER, and INTEGER types, respectively. The third, fourth, and fifth parameters are of the INTEGER type.

No.	Oracle Database	GaussDB	Difference
6	CONVERTTOCLOB Procedure	LOB_CONVERTTOCLOB Procedure	GaussDB: This procedure has five parameters. The third, fourth, and fifth parameters are of the BIGINT type. Oracle Database: This procedure has eight parameters. The third, fourth, and fifth parameters are of the INTEGER type. In addition to all GaussDB parameters, this procedure in Oracle Database adds three parameters: **blob_csid**, **lang_context**, and **warning**, which are of NUMBER, INTEGER, and INTEGER types, respectively.
7	COPY Procedures	LOB_COPY Functions	-
8	COPY_DBFS_LINK Procedures	Not supported.	-
9	COPY_FROM_DBFS_LINK	Not supported.	-

No.	Oracle Database	GaussDB	Difference
10	CREATETEMPOracleRY Procedures	CREATE_TEMPOracleRY Procedures	GaussDB: There are two overloaded procedures. The first parameter (**lob_loc**) of the first overloaded procedure is of the BLOB type, and the first parameter (**lob_loc**) of the second overloaded procedure is of the CLOB type. The third parameter (**dur**) of the two overloaded procedures is of the INTEGER type, and the default value is **10**. Oracle Database: There are two overloaded procedures. The first parameter (**lob_loc**) of the first overloaded procedure is of the BLOB type. The first parameter (**lob_loc**) of the second overloaded procedure is of the CLOB type. The third parameter (**dur**) of the two overloaded procedures is of the PLS_INTEGER type. The default value of **dur** of the first overloaded procedure is **DBMS_LOB.SESSION**, and the default value of **dur** of the second overloaded procedure is **10**.
11	DBFS_LINK_GENERATE_PATH Functions	Not supported.	-
12	ERASE Procedures	LOB_ERASE Procedures	-
13	FILECLOSE Procedure	Not supported.	-
14	FILECLOSEALL Procedure	Not supported.	-
15	FILEEXISTS Function	Not supported.	-
16	FILEGETNAME Procedure	Not supported.	-

No.	Oracle Database	GaussDB	Difference
17	FILEISOPEN Function	Not supported.	-
18	FILEOPEN Procedure	Not supported.	-
19	FRAGMENT_DELETE Procedure	Not supported.	-
20	FRAGMENT_INSERT Procedures	Not supported.	-
21	FRAGMENT_MOVE Procedure	Not supported.	-
22	FRAGMENT_REPLACE Procedures	Not supported.	-
23	FREETEMPOracleRY Procedures	Not supported.	-
24	GET_DBFS_LINK Functions	Not supported.	-
25	GET_DBFS_LINK_STATE Procedures	Not supported.	-
26	GETCHUNKSIZE Functions	GETCHUNKSIZE Functions	-
27	GETCONTENTTYPE Functions	Not supported.	-
28	GETLENGTH Functions	Not supported.	-
29	GETOPTIONS Functions	Not supported.	-
30	GET_STOracleGE_LIMIT Function	Not supported.	-

No.	Oracle Database	GaussDB	Difference
31	INSTR Functions	MATCH Functions	GaussDB: There are three overloaded functions. The third and fourth parameters of the three overloaded functions are of the BIGINT type. Oracle Database: There are three overloaded functions. The third and fourth parameters of the three overloaded functions are of the INTEGER type.
32	ISOPEN Functions	Not supported.	-
33	ISREMOTE Function	Not supported.	-
34	ISSECUREFILE Function	Not supported.	-
35	ISTEMPOracleRY Functions	Not supported.	-
36	LOADBLOBFROMFILE Procedure	LOADBLOBFROMFILE Procedure	-
37	LOADCLOBFROMFILE Procedure	LOADCLOBFROMFILE Procedure	-
38	LOADFROMFILE Procedure	LOADFROMFILE Procedure	-
39	MOVE_TO_DBFS_LINK Procedures	Not supported.	-

No.	Oracle Database	GaussDB	Difference
40	OPEN Procedures	BFILEOPEN Procedure	GaussDB: There is no overloaded procedure. The first parameter (**bfile**) is of the DBE_LOB.BFILE type, and the second parameter (**open_mode**) is of the TEXT type. Only the read mode is supported. Oracle Database: There are three overloaded procedures. In the first overloaded procedure, the first parameter (**lob_loc**) is of the NOCOPY BLOB type, and the second parameter (**openmode**) is of the BINARY_INTEGER type. In the second overloaded procedure, the first parameter (**lob_loc**) is of the NOCOPY CLOB CHARACTER SET ANY_CS type, and the second parameter (**openmode**) is of the BINARY_INTEGER type. In the third overloaded procedure, the first parameter (**file_loc**) is of the NOCOPY BFILE type, and the second parameter (**openmode**) is of the BINARY_INTEGER type, and the value can only be **file_readonly**.
41	READ Procedures	READ Procedures	GaussDB: There are two overloaded procedures. Oracle Database: There are three overloaded procedures. The first two overloaded procedures are the same as those in GaussDB. The third overloaded procedure includes four parameters: **file_loc**, **amount**, **offset**, and **buffer**, which are of the BFILE, NOCOPY INTEGER, INTEGER, and RAW types, respectively.
42	SET_DBFS_LINK Procedures	Not supported.	-

No.	Oracle Database	GaussDB	Difference
43	SETCONTENTTYPE Procedure	Not supported.	-
44	SETOPTIONS Procedures	Not supported.	-
45	SUBSTR Functions	LOB_SUBSTR Functions	-
46	TRIM Procedures	STRIP Functions	GaussDB: There are two overloaded procedures. The second parameter (**newlen**) of the two overloaded procedures is of the BIGINT type. Oracle Database: There are two overloaded procedures. The second parameter (**newlen**) of the two overloaded procedures is of the INTEGER type.
47	WRITE Procedures	WRITE Functions	-
48	WRITEAPPEND Procedures	WRITEAPPEND Functions	-

Table 2-108 DBMS_RANDOM compatibility

No.	Oracle Database	GaussDB	Difference
1	INITIALIZE Procedure	Not supported.	-
2	NORMAL Function	Not supported.	-
3	RANDOM Function	Not supported.	-

No.	Oracle Database	GaussDB	Difference
4	SEED Procedures	DBE_RANDOM.SET_SEED Function	GaussDB: There is no overloaded function. The parameter is of the INTEGER type. Oracle Database: There are two overloaded procedures. The parameter types of the two overloaded procedures are VARCHAR2 and BINARY_INTEGER, respectively.
5	STRING Function	Not supported.	-
6	TERMINATE Procedure	Not supported.	-
7	VALUE Functions	DBE_RANDOM.GET_VALUE Function	GaussDB: There is no overloaded function. Oracle Database: The VALUE function without parameters is overloaded, and the return type is NUMBER.

Table 2-109 DBMS_OUTPUT compatibility

No.	Oracle Database	GaussDB	Difference
1	DISABLE Procedure	DISABLE Function	-
2	ENABLE Procedure	ENABLE Function	-
3	GET_LINE Procedure	GET_LINE Function	GaussDB: There is no overloaded function. The first parameter (**lines**) is of the VARCHAR[] type. Oracle Database: There are two overloaded procedures. The first parameters (**lines**) of the two overloaded procedures are of the CHARARR and DBMSOUTPUT_LINESARRAY types, respectively.
4	GET_LINES Procedure	GET_LINES Function	-

No.	Oracle Database	GaussDB	Difference
5	NEW_LINE Procedure	NEW_LINE Function	-
6	PUT Procedure	PUT Function	GaussDB: If the character set of the database server (**server_encoding**) is not encoded in UTF-8 and the character encoding of the input parameter is valid UTF-8, this function converts character encoding based on the relationship "UTF8 > server_encoding" and then outputs the result regardless of the data type of the input parameter. Oracle Database: If the character set of the database server (**server_encoding**) is not encoded in UTF-8, the character encoding of the input parameter is valid UTF-8, and the input parameter type is NVARCHAR2, this procedure coverts the character encoding based on the relationship "UTF8 > server_encoding" and then outputs the result. If the input parameter is of other character types, the character encoding is regarded as an invalid character and output as a placeholder.

No.	Oracle Database	GaussDB	Difference
7	PUT_LINE Procedure	PUT_LINE Function	GaussDB: If the character set of the database server (**server_encoding**) is not encoded in UTF-8 and the character encoding of the input parameter is valid UTF-8, this function converts character encoding based on the relationship "UTF8 > server_encoding" and then outputs the result regardless of the data type of the input parameter. Oracle Database: If the character set of the database server (**server_encoding**) is not encoded in UTF-8, the character encoding of the input parameter is valid UTF-8, and the input parameter type is NVARCHAR2, this procedure coverts the character encoding based on the relationship "UTF8 > server_encoding" and then outputs the result. If the input parameter is of other character types, the character encoding is regarded as an invalid character and output as a placeholder.

Table 2-110 UTL_RAW compatibility

No.	Oracle Database	GaussDB	Difference
1	BIT_AND Function	BIT_AND Function	-
2	BIT_COMPLEMENT Function	BIT_COMPLEMENT Function	-
3	BIT_OR Function	BIT_OR Function	GaussDB: The two parameters are defined as the TEXT type and returned as the TEXT type. Oracle Database: The two parameters are of the RAW type and returned as the RAW type.

No.	Oracle Database	GaussDB	Difference
4	BIT_XOR Function	BIT_XOR Function	-
5	CAST_FROM_BINARY_DOUBLE Function	CAST_FROM_BINARY_DOUBLE_TO_RAW Function	-
6	CAST_FROM_BINARY_FLOAT Function	CAST_FROM_BINARY_FLOAT_TO_RAW Function	GaussDB: The **n** parameter is of the FLOAT4 type. Oracle Database: The **n** parameter is of the FLOAT type.
7	CAST_FROM_BINARY_INTEGER Function	CAST_FROM_BINARY_INTEGER_TO_RAW Function	GaussDB: The **value** parameter is of the BIGINT type. Oracle Database: The **value** parameter is of the INTEGER type.
8	CAST_FROM_NUMBER Function	CAST_FROM_NUMBER_TO_RAW Function	GaussDB: The **n** parameter is of the NUMERIC type. Oracle Database: The **n** parameter is of the NUMBER type.
9	CAST_TO_BINARY_DOUBLE Function	CAST_FROM_RAW_TO_BINARY_DOUBLE Function	-
10	CAST_TO_BINARY_FLOAT Function	CAST_FROM_RAW_TO_BINARY_FLOAT Function	GaussDB: The function returns the FLOAT4 type. Oracle Database: The function returns the FLOAT type.
11	CAST_TO_BINARY_INTEGER Function	CAST_FROM_RAW_TO_BINARY_INTEGER Function	GaussDB: The **endianess** parameter is of the INTEGER type, and the function returns the INTEGER type. Oracle Database: The **endianess** parameter is of the PLS_INTEGER type, and the function returns the BINARY_INTEGER type.
12	CAST_TO_NUMBER Function	CAST_FROM_RAW_TO_NUMBER Function	GaussDB: The function returns the NUMERIC type. Oracle Database: The function returns the NUMBER type.

No.	Oracle Database	GaussDB	Difference
13	CAST_TO_NVARCHAR2 Function	CAST_FROM_RAW_TO_NVARCHAR2 Function	-
14	CAST_TO_RAW Function	CAST_FROM_VARCHAR2_TO_RAW Function	-
15	CAST_TO_VARCHAR2 Function	CAST_TO_VARCHAR2 Function	-
16	COMPARE Function	COMPARE Function	GaussDB: The function returns the INTEGER type. Oracle Database: The function returns the NUMBER type.
17	CONCAT Function	CONCAT Function	-
18	CONVERT Function	CONVERT Function	-
19	COPIES Function	COPIES Function	GaussDB: The **n** parameter is of the NUMERIC type. Oracle Database: The **n** parameter is of the NUMBER type.
20	LENGTH Function	GET_LENGTH Function	-
21	OVERLAY Function	OVERLAY Function	-
22	REVERSE Function	REVERSE Function	-
23	SUBSTR Function	SUBSTR Function	GaussDB: The **lob_loc** parameter is of the BLOB type. The **off_set** parameter is of the INTEGER type and its default value is **1**. The **amount** parameter is of the INTEGER type and its default value is **32767**. Oracle Database: The **r** parameter is of the RAW type. The **pos** parameter is of the BINARY_INTEGER type and has no default value. The **len** parameter is of the BINARY_INTEGER type and its default value is **NULL**.

No.	Oracle Database	GaussDB	Difference
24	TRANSLATE Function	TRANSLATE Function	-
25	TRANSLITERATE Function	TRANSLITERATE Function	-
26	XRANGE Function	XRANGE Function	GaussDB: The **start_byte** and **end_byte** parameters do not have default values. Oracle Database: The default values of the **start_byte** and **end_byte** parameters are **NULL**.

Table 2-111 DBMS_SCHEDULER compatibility

No.	Oracle Database	GaussDB
1	ADD_EVENT_QUEUE_SUBSCRIBER Procedure	Not supported.
2	ADD_GROUP_MEMBER Procedure	Not supported.
3	ADD_JOB_EMAIL_NOTIFICATION Procedure	Not supported.
4	ADD_TO_INCOMPATIBILITY Procedure	Not supported.
5	ALTER_CHAIN Procedure	Not supported.
6	ALTER_RUNNING_CHAIN Procedure	Not supported.
7	CLOSE_WINDOW Procedure	Not supported.
8	COPY_JOB Procedure	Not supported.
9	CREATE_CHAIN Procedure	Not supported.
10	CREATE_CREDENTIAL Procedure	CREATE_CREDENTIAL Procedure
11	CREATE_DATABASE_DESTINATION Procedure	Not supported.
12	CREATE_EVENT_SCHEDULE Procedure	Not supported.
13	CREATE_FILE_WATCHER Procedure	Not supported.
14	CREATE_GROUP Procedure	Not supported.
15	CREATE_INCOMPATIBILITY Procedure	Not supported.

No.	Oracle Database	GaussDB
16	CREATE_JOB Procedure	CREATE_JOB Procedure
17	CREATE_JOB_CLASS Procedure	CREATE_JOB_CLASS Procedure
18	CREATE_JOBS Procedure	Not supported.
19	CREATE_PROGRAM Procedure	CREATE_PROGRAM Procedure
20	CREATE_RESOURCE Procedure	Not supported.
21	CREATE_SCHEDULE Procedure	CREATE_SCHEDULE Procedure
22	CREATE_WINDOW Procedure	Not supported.
23	DEFINE_ANYDATA_ARGUMENT Procedure	Not supported.
24	DEFINE_CHAIN_EVENT_STEP Procedure	Not supported.
25	DEFINE_CHAIN_RULE Procedure	Not supported.
26	DEFINE_CHAIN_STEP Procedure	Not supported.
27	DEFINE_METADATA_ARGUMENT Procedure	Not supported.
28	DEFINE_PROGRAM_ARGUMENT Procedure	DEFINE_PROGRAM_ARGUMENT Procedure
29	DISABLE Procedure	DISABLE Procedure
30	DROP_AGENT_DESTINATION Procedure	Not supported.
31	DROP_CHAIN Procedure	Not supported.
32	DROP_CHAIN_RULE Procedure	Not supported.
33	DROP_CHAIN_STEP Procedure	Not supported.
34	DROP_CREDENTIAL Procedure	DROP_CREDENTIAL Procedure
35	DROP_DATABASE_DESTINATION Procedure	Not supported.
36	DROP_FILE_WATCHER Procedure	Not supported.
37	DROP_GROUP Procedure	Not supported.
38	DROP_INCOMPATIBILITY Procedure	Not supported.
39	DROP_JOB Procedure	DROP_JOB Procedure
40	DROP_JOB_CLASS Procedure	DROP_JOB_CLASS Procedure
41	DROP_PROGRAM Procedure	DROP_PROGRAM Procedure

No.	Oracle Database	GaussDB
42	DROP_PROGRAM_ARGUMENT Procedure	Not supported.
43	DROP_SCHEDULE Procedure	DROP_SCHEDULE Procedure
44	DROP_WINDOW Procedure	Not supported.
45	ENABLE Procedure	ENABLE Procedure
46	END_DETACHED_JOB_RUN Procedure	Not supported.
47	EVALUATE_CALENDAR_STRING Procedure	EVALUATE_CALENDAR_STRING Procedure
48	EVALUATE_RUNNING_CHAIN Procedure	Not supported.
49	GENERATE_JOB_NAME Function	GENERATE_JOB_NAME Function
50	GET_AGENT_INFO Function	Not supported.
51	GET_AGENT_VERSION Function	Not supported.
52	GET_ATTRIBUTE Procedure	Not supported.
53	GET_FILE Procedure	Not supported.
54	GET_SCHEDULER_ATTRIBUTE Procedure	Not supported.
55	OPEN_WINDOW Procedure	Not supported.
56	PURGE_LOG Procedure	Not supported.
57	PUT_FILE Procedure	Not supported.
58	REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure	Not supported.
59	REMOVE_FROM_INCOMPATIBILITY Procedure	Not supported.
60	REMOVE_GROUP_MEMBER Procedure	Not supported.
61	REMOVE_JOB_EMAIL_NOTIFICATION Procedure	Not supported.
62	RESET_JOB_ARGUMENT_VALUE Procedure	Not supported.
63	RUN_CHAIN Procedure	Not supported.
64	RUN_JOB Procedure	RUN_JOB Procedure

No.	Oracle Database	GaussDB
65	SET_AGENT_REGISTRATION_PASS Procedure	Not supported.
66	SET_ATTRIBUTE Procedure	SET_ATTRIBUTE Procedure
67	SET_ATTRIBUTE_NULL Procedure	Not supported.
68	SET_JOB_ANYDATA_VALUE Procedure	Not supported.
69	SET_JOB_ARGUMENT_VALUE Procedure	SET_JOB_ARGUMENT_VALUE Procedure
70	SET_JOB_ATTRIBUTES Procedure	Not supported.
71	SET_RESOURCE_CONSTRAINT Procedure	Not supported.
72	SET_SCHEDULER_ATTRIBUTE Procedure	Not supported.
73	STOP_JOB Procedure	STOP_JOB Procedure

Table 2-112 DBMS_UTILITY compatibility

No.	Oracle Database	GaussDB	Difference
1	ACTIVE_INSTANCES Procedure	Not supported.	-
2	ANALYZE_DATABASE Procedure	Not supported.	-
3	ANALYZE_PART_OBJECT Procedure	Not supported.	-
4	ANALYZE_SCHEMA Procedure	Not supported.	-
5	CANONICALIZE Procedure	CANONICALIZE Procedure	GaussDB: The default size of the **canon_len** parameter is 1024 bytes. Oracle Database: The **canon_len** parameter has no default value.

No.	Oracle Database	GaussDB	Difference
6	COMMA_TO_TABLE Procedures	COMMA_TO_TABLE Procedures	GaussDB: The **tab** parameter is a VARCHAR2 array. Oracle Database: There are two overloaded procedures. The **tab** parameter can be of the uncl_array or lname_array type.
7	COMPILE_SCHEMA Procedure	Not supported.	-
8	CREATE_ALTER_TYPE_ERROR_TABLE Procedure	Not supported.	-
9	CURRENT_INSTANCE Function	Not supported.	-
10	DATA_BLOCK_ADDRESS_BLOCK Function	Not supported.	-
11	DATA_BLOCK_ADDRESS_FILE Function	Not supported.	-
12	DB_VERSION Procedure	DB_VERSION Procedure	GaussDB: There is only the **version** parameter, which is of the VARCHAR2 type. Oracle Database: There are the **version** and **compatibility** parameters, which are of the VARCHAR2 type.
13	EXEC_DDL_STATEMENT Procedure	EXEC_DDL_STATEMENT Function	GaussDB: The **parse_string** parameter is of the TEXT type. Oracle Database: The **parse_string** parameter is of the VARCHAR2 type.

No.	Oracle Database	GaussDB	Difference
14	EXPAND_SQL_TEXT Procedure	EXPAND_SQL_TEXT Function	GaussDB: The **output_sql_text** parameter is of the CLOB type. Oracle Database: The **output_sql_text** parameter is of the NOCOPY CLOB type. The **OUT** parameter is passed by reference.
15	FORMAT_CALL_STACK Function	FORMAT_CALL_STACK Function	GaussDB: The function returns the TEXT type. Oracle Database: The function returns the VARCHAR2 type.
16	FORMAT_ERROR_BACKTRACE Function	FORMAT_ERROR_BACKTRACE Function	GaussDB: The function returns the TEXT type. Oracle Database: The function returns the VARCHAR2 type.
17	FORMAT_ERROR_STACK Function	FORMAT_ERROR_STACK Function	GaussDB: The function returns the TEXT type. Oracle Database: The function returns the VARCHAR2 type.
18	GET_CPU_TIME Function	GET_CPU_TIME Function	GaussDB: The function returns the BIGINT type. Oracle Database: The function returns the NUMBER type.
19	GET_DEPENDENCY Procedure	Not supported.	-
20	GET_ENDIANNESS Function	GET_ENDIANNESS Function	GaussDB: The function returns the INTEGER type. Oracle Database: The function returns the NUMBER type.

No.	Oracle Database	GaussDB	Difference
21	GET_HASH_VALUE Function	GET_HASH_VALUE Function	GaussDB: The **base** and **hash_size** parameter values and the return value are all of the INTEGER type. Oracle Database: The **base** and **hash_size** values and the return value are all of the NUMBER type.
22	GET_PARAMETER_VALUE Function	Not supported.	-
23	GET_SQL_HASH Function	GET_SQL_HASH Function	GaussDB: The **last4bytes** parameter of the BIGINT type specifies the last four bytes of an MD5 hash value and is displayed as an unsigned integer. The function returns the BIGINT type. Oracle Database: The **pre10ihash** parameter of the NUMBER type is used to store the 4-byte hash value among the 16 bytes calculated by MD5.
24	GET_TIME Function	GET_TIME Function	GaussDB: The function returns the BIGINT type. Oracle Database: The function returns the NUMBER type.
25	GET_TZ_TRANSITIONS Procedure	Not supported.	-
26	INVALIDATE Procedure	Not supported.	-
27	IS_BIT_SET Function	IS_BIT_SET Function	GaussDB: The **n** parameter and return value type are both INTEGER. Oracle Database: The **n** parameter and return value type are both NUMBER.

No.	Oracle Database	GaussDB	Difference
28	IS_CLUSTER_DATABASE Function	IS_CLUSTER_DATABASE Function	-
29	MAKE_DATA_BLOCK_ADDRESS Function	Not supported.	-
30	NAME_RESOLVE Procedure	NAME_RESOLVE Procedure	GaussDB: The **context** and **part1_type** parameters are of the INTEGER type, and the **object_number** parameter is of the OID type. GaussDB does not support implicit conversion from NUMBER to OID. Oracle Database: The **context**, **part1_type**, and **object_number** parameters are of the NUMBER type.
31	NAME_TOKENIZE Procedure	NAME_TOKENIZE Procedure	GaussDB: The **nextpos** parameter is of the INTEGER type. Oracle Database: The **nextpos** parameter is of the BINARY_INTEGER type.
32	OLD_CURRENT_SCHEMA Function	OLD_CURRENT_SCHEMA Function	GaussDB: The function returns the VARCHAR type. Oracle Database: The function returns the VARCHAR2 type.
33	OLD_CURRENT_USER Function	OLD_CURRENT_USER Function	GaussDB: The function returns the TEXT type. Oracle Database: The function returns the VARCHAR2 type.
34	PORT_STRING Function	Not supported.	-
35	SQLID_TO_SQLHASH Function	Not supported.	-

No.	Oracle Database	GaussDB	Difference
36	TABLE_TO_COMMA Procedures	TABLE_TO_COMMA Procedures	GaussDB: The **tab** parameter is a VARCHAR2 array. Oracle Database: There are two overloaded stored procedures. The **tab** parameter can be of the uncl_array or lname_array type.
37	VALIDATE Procedure	Not supported.	-
38	WAIT_ON_PENDING_DML Function	Not supported.	-

Table 2-113 DBMS_SQL compatibility

No.	Oracle Database	GaussDB	Difference
1	BIND_ARRAY Procedures	SQL_BIND_ARRAY Function	-
2	BIND_VARIABLE Procedures	SQL_BIND_VARIABLE Function	-
3	BIND_VARIABLE_PKG Procedure	Not supported.	-
4	CLOSE_CURSOR Procedure	SQL_UNREGISTER_CONTEXT Function	-
5	COLUMN_VALUE Procedure	GET_RESULT Procedure	-
6	COLUMN_VALUE_LONG Procedure	Not supported.	-
7	DEFINE_ARRAY Procedure	SET_RESULTS_TYPE Procedure	-
8	DEFINE_COLUMN Procedures	SET_RESULT_TYPE Procedure	-
9	DEFINE_COLUMN_CHAR Procedure	Not supported.	-
10	DEFINE_COLUMN_LONG Procedure	Not supported.	-
11	DEFINE_COLUMN_RAW Procedure	Not supported.	-

No.	Oracle Database	GaussDB	Difference
12	DEFINE_COLUMN_ROWID Procedure	Not supported.	-
13	DESCRIBE_COLUMNS Procedure	DESCRIBE_COLUMNS Procedure	-
14	DESCRIBE_COLUMNS2 Procedure	Not supported.	-
15	DESCRIBE_COLUMNS3 Procedure	Not supported.	-
16	EXECUTE Function	SQL_RUN Function	GaussDB: The return value is a constant **1**. Currently, the comparison between unknown types in the statement cannot return correct results. Oracle Database: The return value is the number of affected rows for INSERT, UPDATE, and DELETE statements and is meaningless for other statements.
17	EXECUTE_AND_FETCH Function	RUN_AND_NEXT Function	-
18	FETCH_ROWS Function	NEXT_ROW Function	-
19	GET_NEXT_RESULT Procedures	Not supported.	-
20	IS_OPEN Function	IS_ACTIVE Function	-
21	LAST_ERROR_POSITION Function	Not supported.	-
22	LAST_ROW_COUNT Function	LAST_ROW_COUNT Function	-
23	LAST_ROW_ID Function	Not supported.	-
24	LAST_SQL_FUNCTION_CODE Function	Not supported.	-
25	OPEN_CURSOR Functions	REGISTER_CONTEXT Function	-

No.	Oracle Database	GaussDB	Difference
26	PARSE Procedures	Supported, with differences.	In GaussDB, the SQL_SET_SQL function does not support overloading.
27	RETURN_RESULT Procedures	Not supported.	-
28	TO_CURSOR_NUMBER Function	Not supported.	-
29	TO_REFCURSOR Function	Not supported.	-
30	VARIABLE_VALUE Procedures	GET_VARIABLE_RESULT Procedures	-
31	VARIABLE_VALUE_PKG Procedure	Not supported.	-

Table 2-114 DBMS_SQL data type compatibility

No.	Oracle Database	GaussDB
1	DBMS_SQL DESC_REC	DBE_SQL.DESC_REC
2	DBMS_SQL DATE_TABLE	DBE_SQL.DATE_TABLE
3	DBMS_SQL NUMBER_TABLE	DBE_SQL.NUMBER_TABLE
4	DBMS_SQL VARCHAR2_TABLE	DBE_SQL.VARCHAR2_TABLE
5	DBMS_SQL BLOB_TABLE	DBE_SQL.BLOB_TABLE

Table 2-115 UTL_FILE compatibility

No.	Oracle Database	GaussDB	Difference
1	FCLOSE Procedure	CLOSE Procedure	-
2	FCLOSE_ALL Procedure	CLOSE_ALL Procedure	-
3	FCOPY Procedure	COPY Procedure	-
4	FFLUSH Procedure	FLUSH Procedure	-

No.	Oracle Database	GaussDB	Difference
5	FGETATTR Procedure	GET_ATTR Procedure	-
6	FGETPOS Function	GET_POS Function	-
7	FOPEN Function	FOPEN Function	-
8	FOPEN_NCHAR Function	FOPEN_NCHAR Function	-
9	FREMOVE Procedure	REMOVE Procedure	-
10	FRENAME Procedure	RENAME Procedure	-
11	FSEEK Procedure	SEEK Procedure	-
12	GET_LINE Procedure	READ_LINE Procedure	-
13	GET_LINE_NCHAR Procedure	READ_LINE_NCHAR Procedure	-
14	GET_RAW Procedure	GET_RAW Procedure	-
15	IS_OPEN Function	IS_OPEN Function	-
16	NEW_LINE Procedure	Supported, with differences in the NEW_LINE function	GaussDB defines the API as a function.
17	PUT Procedure	Supported, with differences in the WRITE function	GaussDB defines the API as a function.
18	PUT_LINE Procedure	Supported, with differences in the WRITE_LINE function	GaussDB defines the API as a function.
19	PUT_LINE_NCHAR Procedure	Supported, with differences in the WRITE_LINE_NCHAR function	GaussDB defines the API as a function.
20	PUT_NCHAR Procedure	Supported, with differences in the WRITE_NCHAR function	GaussDB defines the API as a function.

No.	Oracle Database	GaussDB	Difference
21	PUTF Procedure	Supported, with differences in the FORMAT_WRITE function	GaussDB defines the API as a function.
22	PUTF_NCHAR Procedure	Supported, with differences in the FORMAT_WRITE_NCHAR function	GaussDB defines the API as a function.
23	PUT_RAW Procedure	Supported, with differences in the PUT_RAW function	GaussDB defines the API as a function.

Table 2-116 DBMS_SESSION compatibility

No.	Oracle Database	GaussDB	Difference
1	CLEAR_ALL_CONTEXT Procedure	Not supported.	-
2	CLEAR_CONTEXT Procedure	CLEAR_CONTEXT Function	-
3	CLEAR_IDENTIFIER Procedure	Not supported.	-
4	CLOSE_DATABASE_LINK Procedure	Not supported.	-
5	CURRENT_IS_ROLE_ENABLED Function	Not supported.	-
6	FREE_UNUSED_USER_MEMORY Procedure	Not supported.	-
7	GET_PACKAGE_MEMORY_UTILIZATION Procedure	Not supported.	-
8	IS_ROLE_ENABLED Function	Not supported.	-
9	IS_SESSION_ALIVE Function	Not supported.	-
10	LIST_CONTEXT Procedures	Not supported.	-

No.	Oracle Database	GaussDB	Difference
11	MODIFY_PACKAGE_STATE Procedure	MODIFY_PACKAGE_STATE Procedure	GaussDB: The scenario where **flags** is set to **1** is supported. Oracle Database: The scenario where **flags** is set to **1** or **2** is supported.
12	RESET_PACKAGE Procedure	Not supported.	-
13	SESSION_IS_ROLE_ENABLED Function	Not supported.	-
14	SESSION_TRACE_DISABLE Procedure	Not supported.	-
15	SESSION_TRACE_ENABLE Procedure	Not supported.	-
16	SET_CONTEXT Procedure	SET_CONTEXT Function	GaussDB: There are the **namespace**, **attribute**, and **value** parameters of the TEXT type. Oracle Database: There are the **namespace**, **attribute**, **value**, **username**, and **client_id** parameters of the VARCHAR2 type.
17	SET_EDITION_DEFERRED Procedure	Not supported.	-
18	SET_IDENTIFIER Procedure	Not supported.	-
19	SET_NLS Procedure	Not supported.	-
20	SET_ROLE Procedure	Not supported.	-
21	SET_SQL_TRACE Procedure	Not supported.	-
22	SLEEP Procedure	Not supported.	-
23	SWITCH_CURRENT_CONSUMER_GROUP Procedure	Not supported.	-
24	UNIQUE_SESSION_ID Function	Not supported.	-

Table 2-117 UTL_MATCH compatibility

No.	Oracle Database	GaussDB	Difference
1	EDIT_DISTANCE Function	Not supported.	-
2	EDIT_DISTANCE_SIMILARITY Function	EDIT_DISTANCE_SIMILARITY Function	GaussDB: The **str1** and **str2** parameters are of the TEXT type, and the function returns the INTEGER type. Oracle Database: The **s1** and **s2** parameters are of the VARCHAR2 type, and the function returns the PLS_INTEGER type.
3	JARO_WINKLER Function	Not supported.	-
4	JARO_WINKLER_SIMILARITY Function	Not supported.	-

Table 2-118 DBMS_APPLICATION_INFO compatibility

No.	Oracle Database	GaussDB	Difference
1	READ_CLIENT_INFO Function	READ_CLIENT_INFO Procedure	GaussDB: The **client_info** parameter is of the TEXT type. Oracle Database: The **client_info** parameter is of the VARCHAR2 type.
2	READ_MODULE Procedure	READ_MODULE Procedure	GaussDB: The **module_name** and **action_name** parameters are of the TEXT type. Oracle Database: The **module_name** and **action_name** parameters are of the VARCHAR2 type.
3	SET_ACTION Procedure	SET_ACTION Procedure	GaussDB: The **action_name** parameter is of the TEXT type. Oracle Database: The **action_name** parameter is of the VARCHAR2 type.

No.	Oracle Database	GaussDB	Difference
4	SET_CLIENT_INFO Function	SET_CLIENT_INFO Procedure	GaussDB: The **str** parameter is of the TEXT type, and the return type is VOID. Oracle Database: The **client_info** parameter is of the VARCHAR2 type and no value is returned. Both of them are written to the client. The maximum length is 64 bytes. If the length exceeds 64 bytes, it will be truncated.
5	SET_MODULE Procedure	SET_MODULE Procedure	GaussDB: The **module_name** and **action_name** parameters are of the TEXT type. Oracle Database: The **module_name** and **action_name** parameters are of the VARCHAR2 type.
6	SET_SESSION_LONGOPS Procedure	Not supported.	-

Table 2-119 DBMS_XMLDOM compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_XMLDOM.APPENDCHILD	DBE_XMLDOM.APPENDCHILD	• GaussDB: The error message "operation not support" is displayed for the APPEND ATTR node under the DOCUMENT node. Oracle Database: No error is reported in this scenario, but the mounting fails. • GaussDB: The error message "operation not support" is displayed for the APPEND ATTR node under the ATTR node. Oracle Database: No error is reported in this scenario, but the mounting fails. • GaussDB: When multiple child nodes of the ATTR type are added to a parent node, the child nodes with the same key value cannot exist under the same parent node. Oracle Database: Child nodes with the same key value can exist under the same parent node.
2	DBMS_XMLDOM.CREATEELEMENT	DBE_XMLDOM.CREATEELEMENT	-
3	DBMS_XMLDOM.CREATETEXTNODE	DBE_XMLDOM.CREATETEXTNODE	-
4	DBMS_XMLDOM.FREEDOCUMENT	Supported, with differences in DBE_XMLDOM.FREEDOCUMENT	GaussDB: Objects are not released immediately. They are released after a certain number of objects are accumulated. All nodes in the document are invalid. Oracle Database: The object is released immediately.
5	DBMS_XMLDOM.FREEELEMENT	DBE_XMLDOM.FREEELEMENT	-
6	DBMS_XMLDOM.FREENODE	DBE_XMLDOM.FREENODE	-

No.	Oracle Database	GaussDB	Difference
7	DBMS_XMLDOM.FREENODELIST	Supported, with differences in DBE_XMLDOM.FREENODELIST	GaussDB: The nodelist will be released. Oracle Database: After the nodelist is released, it can still be queried in the original document.
8	DBMS_XMLDOM.GETATTRIBUTE	DBE_XMLDOM.GETATTRIBUTE	-
9	DBMS_XMLDOM.GETATTRIBUTES	DBE_XMLDOM.GETATTRIBUTES	-
10	DBMS_XMLDOM.GETCHILDNODES	DBE_XMLDOM.GETCHILDNODES	GaussDB: When the document node is used, DTD is included. Oracle Database: DTD is not included.
11	DBMS_XMLDOM.GETCHILDRENBYTAGNAME	DBE_XMLDOM.GETCHILDRENBYTAGNAME	GaussDB: The **ns** parameter of the DBE_XMLDOM.GETCHILDRENBYTAGNAME API does not support the asterisk (*) parameter. To obtain all attributes of a node, use the DBE_XMLDOM.GETCHILDNODES API. Oracle Database: The input parameter ***** is supported.
12	DBMS_XMLDOM.GETDOCUMENTELEMENT	DBE.XMLDOM.GETDOCUMENTELEMENT	-
13	DBMS_XMLDOM.GETFIRSTCHILD	DBE_XMLDOM.GETFIRSTCHILD	-
14	DBMS_XMLDOM.GETLASTCHILD	DBE_XMLDOM.GETLASTCHILD	-
15	DBMS_XMLDOM.GETLENGTH	DBE_XMLDOM.GETLENGTH	-
16	DBMS_XMLDOM.GETLOCALNAME	DBE_XMLDOM.GETLOCALNAME	-
17	DBMS_XMLDOM.GETNAMEDITEM	DBE_XMLDOM.GETNAMEDITEM	-
18	DBMS_XMLDOM.GETNEXTSIBLING	DBE_XMLDOM.GETNEXTSIBLING	-

No.	Oracle Database	GaussDB	Difference
19	DBMS_XMLDOM.GETNODENAME	DBE_XMLDOM.GETNODENAME	-
20	DBMS_XMLDOM.GETNODETYPE	DBE_XMLDOM.GETNODETYPE	-
21	DBMS_XMLDOM.GETTAGNAME	DBE_XMLDOM.GETTAGNAME	-
22	DBMS_XMLDOM.IMPORTNODE	DBE_XMLDOM.IMPORTNODE	-
23	DBMS_XMLDOM.ISNULL	DBE_XMLDOM.ISNULL	GaussDB: When the input parameter is of the DOMNODELIST type, an error is reported if the object does not exist in the hash table. Oracle Database: No error is reported.
24	DBMS_XMLDOM.ITEM	DBE_XMLDOM.ITEM	-
25	DBMS_XMLDOM.MAKENODE	DBE_XMLDOM.MAKENODE	GaussDB: This function cannot be directly returned as the function return value. Oracle Database: It is directly returned as the function return value.

No.	Oracle Database	GaussDB	Difference
26	DBMS_XMLDOM.NEWDOMDOCUMENT	DBE_XMLDOM.NEWDOMDOCUMENT	• GaussDB: The size of the input parameter must be less than 1 GB. Oracle Database: The size is the same as that of the CLOB type. • Currently, external DTD parsing is not supported in GaussDB. Oracle Database: External DTD can be parsed. • GaussDB: The default character set of doc created by newdomdocument is UTF-8. Oracle Database: It is generated based on the character set of the server. • GaussDB: Each doc parsed from the same xmltype instance is independent, and the modification of the doc does not affect the xmltype. Oracle Database: Each doc parsed from the same xmltype instance is not independent but associated. • GaussDB: The **version** column supports only **1.0**. If 1.0 to 1.9 are parsed, a warning is reported but the execution is normal. For versions later than 1.9, an error is reported. Oracle Database: No error is reported. • DTD validation difference: GaussDB reports an error for **!ATTLIST to type (CHECK|check|Check) "Ch..."** because the default value **"Ch..."** is not an enumerated value in the brackets. However, Oracle Database does not report this error. An error will be reported for **<! ENTITY baidu "www.baidu.com">..... &Baidu;&writer**, because the

No.	Oracle Database	GaussDB	Difference
			letters are case-sensitive and **Baidu** cannot correspond to **baidu**. Oracle Database: No error is reported. • Namespace verification difference between GaussDB and Oracle Database: Undeclared namespace tags are parsed in GaussDB. Oracle Database: An error is reported.
27	DBMS_XMLDOM.SETATTRIBUTE	DBE_XMLDOM.SETATTRIBUTE	GaussDB: The attribute **key** cannot be null or an empty string. Oracle Database: The attribute **key** can be null or an empty string.
28	DBMS_XMLDOM.SETCHARSET	DBE_XMLDOM.SETCHARSET	Currently, the following character sets are supported in GaussDB: UTF-8, UCS-4, UCS-2, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP, and ASCII. If you enter other character sets, an error is reported or garbled characters may be displayed.
29	DBMS_XMLDOM.SETDOCTYPE	DBE_XMLDOM.SETDOCTYPE	GaussDB: The total length of name, sysid, and pubid cannot exceed 32,500 bytes. Oracle Database: The maximum size is 32,767 bytes.

No.	Oracle Database	GaussDB	Difference
30	DBMS_XMLDOM.WRITETOBUFFER	Supported, with differences in DBE_XMLDOM.WRITETOBUFFER	• GaussDB: The writetobuffer output buffer is limited to less than 1 GB. Oracle Database: The maximum size is 32,767 bytes. • GaussDB: The output doc will contain the XML declaration version and encoding. Oracle Database: It is not contained unless being specified by users. • GaussDB: If the input parameter is of the domnode type and the node is converted from a doc, the output node contains the XML declaration version and encoding. Oracle Database: It is not contained unless being specified by users. • GaussDB: By default, XML files are output in the UTF-8 character set. Oracle Database: It is generated based on the database character set.

No.	Oracle Database	GaussDB	Difference
31	DBMS_XMLDOM.WRITETOCLOB	DBE_XMLDOM.WRITETOCLOB	• GaussDB: The writetoclob size cannot exceed 1 GB. Oracle Database: The supported size depends on the CLOB size. • GaussDB: The output doc will contain the XML declaration version and encoding. Oracle Database: It is not contained unless being specified by users. • GaussDB: If the input parameter is of the domnode type and the node is converted from a doc, the output node contains the XML declaration version and encoding. Oracle Database: It is not contained unless being specified by users. • GaussDB: By default, XML files are output in the UTF-8 character set. Oracle Database: It is generated based on the database character set.

No.	Oracle Database	GaussDB	Difference
32	DBMS_XMLDOM.WRITETOFILE	DBE_XMLDOM.WRITETOFILE	• GaussDB document input parameter. The length of **filename** cannot exceed 255 bytes. For details about **charset**, see the dbe_xmldom.setcharset API. Oracle Database: The length of **filename** is affected by the OS and is greater than 255 bytes. • GaussDB DOMNode input parameter. The length of **filename** cannot exceed 255 bytes. For details about **charset**, see the dbe_xmldom.setcharset API. Oracle Database: The length of **filename** is affected by the OS and is greater than 255 bytes. • GaussDB: This function adds content such as indentation to format the output. The output doc will contain the XML declaration version and encoding. If the input parameter is of the domnode type and the node is converted from a doc, the output node contains the XML declaration version and encoding. Oracle Database: It is not contained unless being specified by users. • GaussDB: If newdomdocument() is used to create a doc without parameters, no error is reported when **charset** is not specified. The UTF-8 character set is used by default. Oracle Database: An error is reported. • GaussDB: The filename must be in the path created in pg_directory. The backslash (\) in the filename will be

No.	Oracle Database	GaussDB	Difference
			converted to a slash (/). Only one slash (/) is allowed. The file name must be in the pg_directory_name/file_name format. Oracle Database: User input is not escaped.
33	DBMS_XMLDOM.GETNODEVALUE	DBE_XMLDOM.GETNODEVALUE	-
34	DBMS_XMLDOM.GETPARENTNODE	DBE_XMLDOM.GETPARENTNODE	-
35	DBMS_XMLDOM.HASCHILDNODES	DBE_XMLDOM.HASCHILDNODES	-
36	DBMS_XMLDOM.MAKEELEMENT	DBE_XMLDOM.MAKEELEMENT	-
37	DBMS_XMLDOM.SETNODEVALUE	DBE_XMLDOM.SETNODEVALUE	• GaussDB: The input parameter of **nodeValue** can be an empty string or **NULL**, but the node value will not be changed. Oracle Database: If you enter an empty string or NULL, the node value is set to an empty string. • GaussDB: Input parameter of nodeValue. The escape character '&' is not supported. If the character string contains the escape character, the node value will be cleared. Oracle Database: Escape characters are supported.
38	DBMS_XMLDOM.GETELEMENTSBYTAGNAM	DBE_XMLDOM.GETELEMENTSBYTAGNAME	-

Table 2-120 DBMS_XMLPARSER compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_XMLPARSER.FREEPARSER	DBE_XMLPARSER.FREEPARSER	-
2	DBMS_XMLPARSER.GETDOCUMENT	DBE_XMLPARSER.GETDOCUMENT	-
3	DBMS_XMLPARSER.GETVALIDATIONMODE	DBE_XMLPARSER.GETVALIDATIONMODE	-
4	DBMS_XMLPARSER.NEWPARSER	Supported, with differences in DBE_XMLPARSER.NEWPARSER	The maximum number of parser objects in GaussDB is 16,777,215, and that in Oracle Database is about 100 million.

No.	Oracle Database	GaussDB	Difference
5	DBMS_XMLPARSER.PARSEBUFFER	Supported, with differences in DBE_XMLPARSER.PARSEBUFFER	• Difference in parsing columns: Only UTF-8 is supported in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported. • Namespace validation difference: Undeclared namespace tags are parsed. However, Oracle Database reports an error. • Difference in parsing XML predefined entities: **'** and **"** are parsed and escaped to **'** and **"**. However, predefined entities in Oracle Database are not escaped to characters. • DTD validation differences: – An error will be reported for **!ATTLIST to type (CHECK|check|Check) "Ch..."** because the default value **"Ch..."** is not an enumerated value in the brackets. However, Oracle Database does not report this error. – An error will be reported for **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer**, because the letters are case-sensitive and **Baidu** cannot correspond to **baidu**. However, Oracle Database does not report this error.

No.	Oracle Database	GaussDB	Difference
6	DBMS_XMLPARSER.PARSECLOB	Supported, with differences in DBE_XMLPARSER.PARSECLOB	• PARSECLOB cannot parse CLOBs greater than or equal to 2 GB. • Difference in parsing columns: Only UTF-8 is supported in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsed, a warning appears but the execution is normal. For versions later than 1.9, an error is reported. • Namespace validation difference: Undeclared namespace tags are parsed. However, Oracle Database reports an error. • Difference in parsing XML predefined entities: **'** and **"** are parsed and escaped to ' and ". However, predefined entities in Oracle Database are not escaped to characters. • DTD validation differences: – An error will be reported for **!ATTLIST to type (CHECK|check|Check) "Ch..."** because the default value **"Ch..."** is not an enumerated value in the brackets. However, Oracle Database does not report this error. – An error will be reported for **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer**, because the letters are case-sensitive and **Baidu** cannot correspond to **baidu**. However, Oracle Database does not report this error.

No.	Oracle Database	GaussDB	Difference
7	DBMS_XMLPARSER.SETVALIDATIONMODE	DBE_XMLPARSER.SETVALIDATIONMODE	-

Table 2-121 DBMS_ILM compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_ILM.ADD_TO_ILM	Not supported.	-
2	DBMS_ILM.ARCHIVESTATENAME	Not supported.	-
3	DBMS_ILM.EXECUTE_ILM	DBE_ILM.EXECUTE_ILM	• The input parameter **Schema** in GaussDB corresponds to **owner** in Oracle Database. • GaussDB does not support the operation of specifying **ilm_scope** (specifying multiple objects at a time).
4	DBMS_ILM.EXECUTE_ILM_TASK	Not supported.	-
5	DBMS_ILM.PREVIEW_ILM	Not supported.	-
6	DBMS_ILM.REMOVE_FROM_ILM	Not supported.	-
7	DBMS_ILM.STOP_ILM	DBE_ILM.STOP_ILM	-

Table 2-122 DBMS_ILM_ADMIN compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_ILM_ADMIN.CLEAR_HEAT_MAP_ALL	Not supported.	-
2	DBMS_ILM_ADMIN.CLEAR_HEAT_MAP_TABLE	Not supported.	-

No.	Oracle Database	GaussDB	Difference
3	DBMS_ILM_ADMIN.CUSTOMIZE_ILM	DBE_ILM_ADMIN.CUSTOMIZE_ILM	The feature parameters corresponding to the values of input parameters are different. • The value of **param** in GaussDB can be **1**, **2**, **7**, **11**, **12**, **13**, **14**, or **15**. • When the value of **param** in GaussDB is **14**, the corresponding feature parameter is **WIND_DURATION** which is used to control the duration of the execution window in automatic scheduling. However, Oracle Database does not have the corresponding feature parameter.
4	DBMS_ILM_ADMIN.DISABLE_ILM	DBE_ILM_ADMIN.DISABLE_ILM	-
5	DBMS_ILM_ADMIN.ENABLE_AUTO_OPTIMIZE	Not supported.	-
6	DBMS_ILM_ADMIN.ENABLE_ILM	DBE_ILM_ADMIN.ENABLE_ILM	-
7	DBMS_ILM_ADMIN. IGNORE_AUTO_OPTIMIZE_ CRITERIA	Not supported.	-
8	DBMS_ILM_ADMIN.SET_HEAT_MAP_ALL	Not supported.	-
9	DBMS_ILM_ADMIN.SET_HEAT_MAP_START	Not supported.	-
10	DBMS_ILM_ADMIN.SET_HEAT_MAP_TABLE	Not supported.	-

Table 2-123 DBMS_COMPRESSION compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_COMPRESSION.GET_COMPRESSION_RATIO	DBE_COMPRESSION.GET_COMPRESSION_RATIO	• GaussDB cannot obtain compression ratios of LOBs. • For obtaining a compression ratio of a single object: – The value of the input parameter **comptype** in GaussDB can only be **1** (uncompressed) or **2** (advanced compression) while Oracle Database also supports values such as **1024** and **2048**. – The value of the input parameter **objtype** in GaussDB can only be **1** (table object) while Oracle Database also supports the value **2** (index object). – Oracle Database uses the **subset_numrows** parameter to directly determine the number of rows to be sampled (that is, the value of the parameter). GaussDB uses **sample_ratio** (sampling rate) to indirectly determine the number of rows to be sampled.

No.	Oracle Database	GaussDB	Difference
2	DBMS_COMPRESSION.GET_COMPRESSION_TYPE	DBE_COMPRESSION.GET_COMPRESSION_TYPE	• Oracle Database uses a row ID to specify the row whose compression type is to be obtained, while GaussDB uses a CTID to specify the row. • The value of **comptype** is returned. The value difference is the same as that of GET_COMPRESSION_RATIO. • In GaussDB, this API can be called only on DNs. For details, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_COMPRESSION" in *Developer Guide*.

Table 2-124 DBMS_HEAT_MAP compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_HEAT_MAP.BLOCK_HEAT_MAP	Not supported.	-
2	DBMS_HEAT_MAP.EXTENT_HEAT_MAP	Not supported.	-
3	DBMS_HEAT_MAP.OBJECT_HEAT_MAP	Not supported.	-
4	DBMS_HEAT_MAP.SEGMENT_HEAT_MAP	Not supported.	-
5	DBMS_HEAT_MAP.TABLESPACE_HEAT_MAP	Not supported.	-
6	Not supported.	DBE_HEAT_MAP.ROW_HEAT_MAP	In GaussDB, this API can be called only on DNs. For details, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_HEAT_MAP" in *Developer Guide*.

Table 2-125 DBMS_DESCRIBE compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_DESCRIBE.DESCRIBE_PROCEDURE	DBE_DESCRIBE.DESCRIBE_PROCEDURE	• The **datatype** parameter is different from Oracle Database. GaussDB returns the OID of the data type, and Oracle Database returns the ID of the data type within Oracle Database. • The **datalength**, **dataprecision**, and **scale** parameters are set to **0** because type constraints (such as number (7,2) and varchar2(20)) cannot be retained when GaussDB creates stored procedures or functions. Oracle Database can use the %type method to obtain constrained data types. • For details about information in GaussDB, see "Stored Procedure > Advanced Packages > Secondary Encapsulation APIs (Recommended) > DBE_DESCRIBE" in *Developer Guide*.

Table 2-126 DBMS_STATS compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_STATS.ALTER_STATS_HISTORY_RETENTION	Not supported.	-
2	DBMS_STATS.CANCEL_ADVISOR_TASK	Not supported.	-
3	DBMS_STATS.CONFIGURE_ADVISOR_FILTER	Not supported.	-
4	DBMS_STATS.CONFIGURE_ADVISOR_OBJ_FILTER	Not supported.	-
5	DBMS_STATS.CONFIGURE_ADVISOR_OPR_FILTER	Not supported.	-
6	DBMS_STATS.CONFIGURE_ADVISOR_RULE_FILTER	Not supported.	-
7	DBMS_STATS.CREATE_ADVISOR_TASK	Not supported.	-

No.	Oracle Database	GaussDB	Difference
8	DBMS_STATS.CONVERT_RAW_VALUE	Not supported.	-
9	DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR	Not supported.	-
10	DBMS_STATS.CONVERT_RAW_VALUE_ROWID	Not supported.	-
11	DBMS_STATS.COPY_TABLE_STATS	Not supported.	-
12	DBMS_STATS.CREATE_EXTENDED_STATS	Not supported.	-
13	DBMS_STATS.CREATE_STAT_TABLE	DBE_STATS.CREATE_STAT_TABLE	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
14	DBMS_STATS.DELETE_COLUMN_STATS	DBE_STATS.DELETE_COLUMN_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In GaussDB, this API can be used to set expression statistics, but **tabname** must be set to an index name corresponding to the expression.

No.	Oracle Database	GaussDB	Difference
15	DBMS_STATS.DELETE_DATABASE_PREFS	Not supported.	-
16	DBMS_STATS.DELETE_DATABASE_STATS	Not supported.	-
17	DEDBMS_STATS.DELETE_DICTIONARY_STATS	Not supported.	-
18	DBMS_STATS.DELETE_FIXED_OBJECTS_STATS	Not supported.	-
19	DBMS_STATS.DELETE_INDEX_STATS	DBE_STATS.DELETE_INDEX_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
20	DBMS_STATS.DELETE_PENDING_STATS	Not supported.	-
21	DBMS_STATS.DELETE_PROCESSING_RATE	Not supported.	-
22	DBMS_STATS.DELETE_SCHEMA_PREFS	Not supported.	-
23	DBMS_STATS.DELETE_SCHEMA_STATS	DBE_STATS.DELETE_SCHEMA_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
24	DBMS_STATS.DELETE_SYSTEM_STATS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
25	DBMS_STATS.DELETE_TABLE_PREFS	Not supported.	-
26	DBMS_STATS.DELETE_TABLE_STATS	DBE_STATS.DELETE_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
27	DBMS_STATS.DIFF_TABLE_STATS_IN_HISTORY	Not supported.	-
28	DBMS_STATS.DIFF_TABLE_STATS_IN_PENDING	Not supported.	-
29	DBMS_STATS.DIFF_TABLE_STATS_IN_STATTAB	Not supported.	-
30	DBMS_STATS.DROP_ADVISOR_TASK	Not supported.	-
31	DBMS_STATS.DROP_EXTENDED_STATS	Not supported.	-
32	DBMS_STATS.DROP_STAT_TABLE	DBE_STATS.DROP_STAT_TABLE	-
33	DBMS_STATS.EXECUTE_ADVISOR_TASK	Not supported.	-

No.	Oracle Database	GaussDB	Difference
34	DBMS_STATS.EXPORT_COLUMN_STATS	DBE_STATS.EXPORT_COLUMN_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • The exported column-level statistics are consistent with those in the pg_statistic catalog. Multiple columns are consistent with those in the pg_statistic_ext catalog. • Index expression statistics can be exported. **tabname** must be set to an index name, and **colname** must be set to an index expression name. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
35	DBMS_STATS.EXPORT_DATABASE_PREFS	Not supported.	-
36	DBMS_STATS.EXPORT_DATABASE_STATS	Not supported.	-
37	DBMS_STATS.EXPORT_DICTIONARY_STATS	Not supported.	-
38	DBMS_STATS.EXPORT_FIXED_OBJECTS_STATS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
39	DBMS_STATS.EXPORT_INDEX_STATS	DBE_STATS.EXPORT_INDEX_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the exported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
40	DBMS_STATS.EXPORT_PENDING_STATS	Not supported.	-
41	DBMS_STATS.EXPORT_SCHEMA_PREFS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
42	DBMS_STATS.EXPORT_SCHEMA_STATS	DBE_STATS.EXPORT_SCHEMA_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the exported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. The column-level statistics of the exported table are consistent with those of the pg_statistic and pg_statistic_ext catalogs. • Permission: You must have the siud permission on the **stattab** table.
43	DBMS_STATS.EXPORT_SYSTEM_STATS	Not supported.	-
44	DBMS_STATS.EXPORT_TABLE_PREFS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
45	DBMS_STATS.EXPORT_TABLE_STATS	DBE_STATS.EXPORT_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the exported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. The column-level statistics exported in cascading mode are consistent with those in the pg_statistic and pg_statistic_ext catalogs. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
46	DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO	Not supported.	-
47	DBMS_STATS.GATHER_DATABASE_STATS	Not supported.	-
48	DBMS_STATS.GATHER_DICTIONARY_STATS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
49	DBMS_STATS.GATHER_FIXED_OBJECTS_STATS	Not supported.	-
50	DBMS_STATS.GATHER_INDEX_STATS	Not supported.	-
51	DBMS_STATS.GATHER_PROCESSING_RATE	Not supported.	-
52	DBMS_STATS.GATHER_SCHEMA_STATS	Not supported.	-
53	DBMS_STATS.GATHER_SYSTEM_STATS	Not supported.	-
54	DBMS_STATS.GATHER_TABLE_STATS	Not supported.	-
55	DBMS_STATS.GENERATE_STATS	Not supported.	-
56	DBMS_STATS.GET_ADVISOR_OPR_FILTER	Not supported.	-
57	DBMS_STATS.GET_ADVISOR_RECS	Not supported.	-
58	DBMS_STATS.GET_COLUMN_STATS	Not supported.	-
59	DBMS_STATS.GET_INDEX_STATS	Not supported.	-
60	DBMS_STATS.GET_PARAM	Not supported.	-
61	DBMS_STATS.GET_PREFS	Not supported.	-
62	DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY	DBE_STATS.GET_STATS_HISTORY_AVAILABILITY	GaussDB queries the collection time of the earliest historical statistics in the entire database.
63	DBMS_STATS.GET_STATS_HISTORY_RETENTION	DBE_STATS.GET_STATS_HISTORY_RETENTION	-
64	DBMS_STATS.GET_SYSTEM_STATS	Not supported.	-
65	DBMS_STATS.GET_TABLE_STATS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
66	DBMS_STATS.IMPLEMENT_ADVISOR_TASK	Not supported.	-
67	DBMS_STATS.IMPORT_COLUMN_STATS	DBE_STATS.IMPORT_COLUMN_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • The exported single-column **col** statistics are the same as those in the pg_statistic catalog. The exported multi-column **ext-col** statistics are consistent with those in the pg_statistic_ext catalog. • Index expression statistics can be imported. **tabname** must be set to an index name, and **colname** must be set to an index expression name. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
68	DBMS_STATS.IMPORT_DATABASE_PREFS	Not supported.	-
69	DBMS_STATS.IMPORT_DATABASE_STATS	Not supported.	-
70	DBMS_STATS.IMPORT_DICTIONARY_STATS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
71	DBMS_STATS.IMPORT_FIXED_OBJECTS_STATS	Not supported.	-
72	DBMS_STATS.IMPORT_INDEX_STATS	DBE_STATS.IMPORT_INDEX_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the imported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
73	DBMS_STATS.IMPORT_SCHEMA_PREFS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
74	DBMS_STATS.IMPORT_SCHEMA_STATS	DBE_STATS.IMPORT_SCHEMA_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the imported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. The column-level statistics of the imported table are consistent with those of the pg_statistic and pg_statistic_ext catalogs. • Permission: You must have the siud permission on the **stattab** table.
75	DBMS_STATS.IMPORT_SYSTEM_STATS	Not supported.	-
76	DBMS_STATS.IMPORT_TABLE_PREFS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
77	DBMS_STATS.IMPORT_TABLE_STATS	DBE_STATS.IMPORT_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • In the **stattab** table, the imported table-level and partition-level statistics are **numrows**, **numblocks**, and **relallvisible**, which correspond to **reltuples**, **relpages**, and **relallvisible** in the pg_class and pg_partition system catalogs, respectively. The column-level statistics imported in cascading mode are consistent with those in the pg_statistic and pg_statistic_ext catalogs. • Permission: You must have the ANALYZE permission to query tables and the siud permission on the **stattab** table.
78	DBMS_STATS.INTERRUPT_ADVISOR_TASK	Not supported.	-
79	DBMS_STATS.LOCK_PARTITION_STATS	DBE_STATS.LOCK_PARTITION_STATS	• In GaussDB, **ownname** must be set to a **Schema** name.

No.	Oracle Database	GaussDB	Difference
80	DBMS_STATS.LOCK_SCHEMA_STATS	DBE_STATS.LOCK_SCHEMA_STATS	• In GaussDB, **ownname** must be set to a **Schema** name.
81	DBMS_STATS.LOCK_TABLE_STATS	DBE_STATS.LOCK_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name.
82	DBMS_STATS.MERGE_COL_USAGE	Not supported.	-
83	DBMS_STATS.PREPARE_COLUMN_VALUES	Not supported.	-
84	DBMS_STATS.PREPARE_COLUMN_VALUES_ROWID	Not supported.	-
85	DBMS_STATS.PUBLISH_PENDING_STATS	Not supported.	-
86	DBMS_STATS.PURGE_STATS	DBE_STATS.PURGE_STATS	-
87	DBMS_STATS.REMAP_STAT_TABLE	Not supported.	-
88	DBMS_STATS.REPORT_ADVISOR_TASK	Not supported.	-
89	DBMS_STATS.REPORT_COL_USAGE	Not supported.	-
90	DBMS_STATS.REPORT_GATHER_AUTO_STATS	Not supported.	-
91	DBMS_STATS.REPORT_GATHER_DATABASE_STATS	Not supported.	-
92	DBMS_STATS.REPORT_GATHER_DICTIONARY_STATS	Not supported.	-
93	DBMS_STATS.REPORT_GATHER_FIXED_OBJ_STATS	Not supported.	-
94	DBMS_STATS.REPORT_GATHER_SCHEMA_STATS	Not supported.	-
95	DBMS_STATS.REPORT_STATS_OPERATIONS	Not supported.	-
96	DBMS_STATS.RESET_ADVISOR_TASK	Not supported.	-
97	DBMS_STATS.RESET_COL_USAGE	Not supported.	-

No.	Oracle Database	GaussDB	Difference
98	DBMS_STATS.RESET_GLOBAL_PREF_DEFAULTS	Not supported.	-
99	DBMS_STATS.RESET_PARAM_DEFAULTS	Not supported.	-
100	DBMS_STATS.RESTORE_DICTIONARY_STATS	Not supported.	-
101	DBMS_STATS.RESTORE_FIXED_OBJECTS_STATS	Not supported.	-
102	DBMS_STATS.RESTORE_SCHEMA_STATS	DBE_STATS.RESTORE_SCHEMA_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
103	DBMS_STATS.RESTORE_SYSTEM_STATS	Not supported.	-
104	DBMS_STATS.RESTORE_TABLE_STATS	DBE_STATS.RESTORE_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
105	DBMS_STATS.RESUME_ADVISOR_TASK	Not supported.	-
106	DBMS_STATS.SCRIPT_ADVISOR_TASK	Not supported.	-
107	DBMS_STATS.SEED_COL_USAGE	Not supported.	-

No.	Oracle Database	GaussDB	Difference
108	DBMS_STATS.SET_ADVISOR_TASK_PARAMETER	Not supported.	-
109	DBMS_STATS.SET_COLUMN_STATS	DBE_STATS.SET_COLUMN_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*.
110	DBMS_STATS.SET_DATABASE_PREFS	Not supported.	-
111	DBMS_STATS.SET_GLOBAL_PREFS	Not supported.	-
112	DBMS_STATS.SET_INDEX_STATS	DBE_STATS.SET_INDEX_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • The **relallvisible** input parameter is added to GaussDB.
113	DBMS_STATS.SET_PARAM	Not supported.	-
114	DBMS_STATS.SET_PROCESSING_RATE	Not supported.	-
115	DBMS_STATS.SET_SCHEMA_PREFS	Not supported.	-

No.	Oracle Database	GaussDB	Difference
116	DBMS_STATS.SET_SYSTEM_STATS	Not supported.	-
117	DBMS_STATS.SET_TABLE_PREFS	Not supported.	-
118	DBMS_STATS.SET_TABLE_STATS	DBE_STATS.SET_TABLE_STATS	• In GaussDB, **ownname** must be set to a **Schema** name. • GaussDB supports only some input parameter functions. For details, see "Stored Procedure > Advanced Package > Secondary Encapsulation APIs (Recommended) > DBE_STATS" in *Developer Guide*. • The **relallvisible** input parameter is added to GaussDB.
119	DBMS_STATS.SHOW_EXTENDED_STATS_NAME	Not supported.	-
120	DBMS_STATS.TRANSFER_STATS	Not supported.	-
121	DBMS_STATS.UNLOCK_PARTITION_STATS	DBE_STATS.UNLOCK_PARTITION_STATS	In GaussDB, **ownname** must be set to a **Schema** name.
122	DBMS_STATS.UNLOCK_SCHEMA_STATS	DBE_STATS.UNLOCK_SCHEMA_STATS	In GaussDB, **ownname** must be set to a **Schema** name.
123	DBMS_STATS.UNLOCK_TABLE_STATS	DBE_STATS.UNLOCK_TABLE_STATS	In GaussDB, **ownname** must be set to a **Schema** name.
124	DBMS_STATS.UPGRADE_STAT_TABLE	Not supported.	-

Table 2-127 DBMS_XMLGEN compatibility

No.	Oracle Database	GaussDB	Difference
1	DBMS_XMLGEN.CONVERT	DBE_XMLGEN.CONVERT	-
2	DBMS_XMLGEN.NEWCONTEXT	DBE_XMLGEN.NEWCONTEXT	-
3	DBMS_XMLGEN.NEWCONTEXTFROMHIERARCHY	DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY	• The maximum depth of recursive XML files generated by GaussDB cannot exceed 50 million layers. • XML files generated by the CONNECT BY statement in Oracle Database's newcontextfromhierarchy method contain XML headers. However, the directly constructed data does not contain the XML header. In GaussDB, the files contain XML headers.
4	DBMS_XMLGEN.SETCONVERTSPECIALCHARS	DBE_XMLGEN.SETCONVERTSPECIALCHARS	-
5	DBMS_XMLGEN.SETNULLHANDLING	DBE_XMLGEN.SETNULLHANDLING	-
6	DBMS_XMLGEN.SETROWSETTAG	DBE_XMLGEN.SETROWSETTAG	-
7	DBMS_XMLGEN.SETROWTAG	DBE_XMLGEN.SETROWTAG	-
8	DBMS_XMLGEN.USENULLATTRIBUTEINDICATOR	DBE_XMLGEN.USENULLATTRIBUTEINDICATOR	-
9	DBMS_XMLGEN.USEITEMTAGSFORCOLL	DBE_XMLGEN.USEITEMTAGSFORCOLL	-
10	DBMS_XMLGEN.GETNUMROWSPROCESSED	DBE_XMLGEN.GETNUMROWSPROCESSED	-

No.	Oracle Database	GaussDB	Difference
11	DBMS_XMLGEN.SETMAXROWS	DBE_XMLGEN.SETMAXROWS	-
12	DBMS_XMLGEN.SETSKIPROWS	DBE_XMLGEN.SETSKIPROWS	-
13	DBMS_XMLGEN.RESTARTQUERY	DBE_XMLGEN.RESTARTQUERY	In distributed GaussDB, the cursor cannot be moved reversely. Therefore, the restartquery function is unavailable.
14	DBMS_XMLGEN.GETXMLTYPE	DBE_XMLGEN.GETXMLTYPE	-
15	DBMS_XMLGEN.GETXML	DBE_XMLGEN.GETXML	-
16	DBMS_XMLGEN.CLOSECONTEXT	DBE_XMLGEN.CLOSECONTEXT	-

3 MySQL Compatibility Description

3.1 Overview of MySQL Compatibility

Overview of M-compatible Mode

M-compatible Mode compares GaussDB in M-compatible mode (**sql_compatibility** set to **'M'**) with MySQL 5.7. Only compatibility features added later than GaussDB Kernel 505.2.0 are described. You are advised to view the specifications and restrictions of the features in *M Compatibility Developer Guide*.

GaussDB is compatible with MySQL in terms of data types, SQL functions, and database objects.

The execution plan, optimization, and EXPLAIN result in GaussDB are different from those in MySQL.

GaussDB and MySQL implement different underlying frameworks. Therefore, there are still some differences between GaussDB and MySQL.

NOTE

The underlying architecture of GaussDB is different from that of MySQL. Therefore, the performance of querying the same schemas under information_schema and m_schema may be different from that in MySQL. For details, see "Schemas" in *M Compatibility Developer Guide*. For example, the execution of the count function cannot be optimized. The time consumed by the SELECT * and SELECT COUNT(*) statements is similar.

Overview of MySQL-compatible Mode

MySQL-compatible Mode compares GaussDB in MySQL-compatible mode (that is, when **sql_compatibility** is set to **'MYSQL'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**) with MySQL 5.7. Only compatibility features added later than GaussDB Kernel 503.0.0 are described. You are advised to view the specifications and restrictions of the features in *Developer Guide*.

NOTE

- The MySQL-compatible mode delivers high compatibility with MySQL in terms of syntax, data types, metadata, and protocols. The MySQL-compatible mode will not evolve because it is not compatible with the MySQL architecture.
- Only the features described in **MySQL-compatible Mode** are compatible with MySQL. The behavior of other features remain the same as that in GaussDB.
- The implementation logic of the MySQL-compatible mode (**sql_compatibility** set to **'MYSQL'**) is similar to that of the B-compatible mode (**sql_compatibility** set to **'B'**) in centralized deployment.

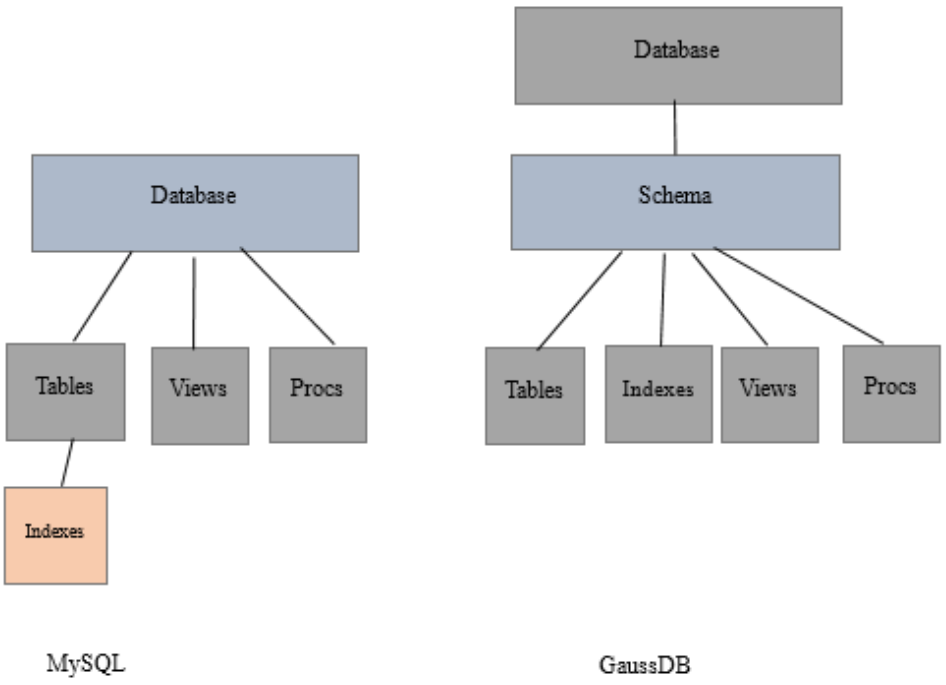
GaussDB is compatible with MySQL in terms of data types, SQL functions, and database objects.

GaussDB and MySQL implement different underlying frameworks. Therefore, there are still some differences between GaussDB and MySQL.

Database and Schema Design

MySQL data objects include database, table, index, view, trigger, and proc, mapping those in GaussDB hierarchically and maybe in a 1:N relationship, as shown in the following figure.

Figure 3-1 Differences between databases and schemas in MySQL and GaussDB



- In MySQL, database and schema are synonyms. In GaussDB, a database can have multiple schemas. In this feature, each database in MySQL is mapped to a schema in GaussDB.
- In MySQL, an index belongs to a table. In GaussDB, an index belongs to a schema. As a result, an index name must be unique in a schema in GaussDB and must be unique in a table in MySQL. This difference will be retained as a current constraint.

3.2 M-compatible Mode

3.2.1 Data Types

3.2.1.1 Numeric Data Types

Unless otherwise specified, the precision, scale, and number of digits cannot be set to floating-point values in M-compatible mode by default. You are advised to use valid integer values.

Table 3-1 Integer types

MySQL	GaussDB	Difference
BOOL	Supported, with differences.	Output format: The output of SELECT TRUE/FALSE in GaussDB is **t** or **f**, and that in MySQL is **1** or **0**. MySQL: The BOOL/BOOLEAN type is actually mapped to the TINYINT type.
BOOLEAN	Supported, with differences.	
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	Input format: • MySQL: If a character string with multiple decimal points (such as "1.2.3.4.5") is entered, MySQL will misparse the character string in loose mode, throw a warning, and insert the character string into the table successfully. For example, after "1.2.3.4.5" is inserted into the table, the value is **12**. • GaussDB: If a character string with multiple decimal points (such as "1.2.3.4.5") is entered in loose mode, the characters after the second decimal point are truncated as invalid characters, a warning is thrown, and the character string is inserted into the table successfully. For example, after "1.2.3.4.5" is inserted into the table, the value is **1**. After "1.6.3.4.5" is inserted into the table, the value is **2**.
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	

MySQL	GaussDB	Difference
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	MySQL requires 3 bytes to store MEDIUMINT data. • The signed range is –8388608 to +8388607. • The unsigned range is 0 to +16777215. GaussDB is mapped to the INT type. Four bytes are required for storage. The value range is determined based on boundary values. • The signed range is –8388608 to +8388607. • The unsigned range is 0 to +16777215. For other differences, see the description below the table.
INT[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	Input format: • MySQL: If a character string with multiple decimal points (such as "1.2.3.4.5") is entered, MySQL will misparse the character string in loose mode, throw a warning, and insert the character string into the table successfully. For example, after "1.2.3.4.5" is inserted into the table, the value is **12**. • GaussDB: If a character string with multiple decimal points (such as "1.2.3.4.5") is entered in loose mode, the characters after the second decimal point are truncated as invalid characters, a warning is thrown, and the character string is inserted into the table successfully. For example, after "1.2.3.4.5" is inserted into the table, the value is **1**. After "1.6.3.4.5" is inserted into the table, the value is **2**.
INTEGER[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	Supported, with differences.	

Table 3-2 Arbitrary precision types

MySQL	GaussDB	Difference
DECIMAL[(M[,D])] [ZEROFILL]	Supported, with differences.	-
NUMERIC[(M[,D])] [ZEROFILL]	Supported, with differences.	

MySQL	GaussDB	Difference
DEC[(M[,D])] [ZEROFILL]	Supported, with differences.	
FIXED[(M[,D])] [ZEROFILL]	Supported, with differences.	

Table 3-3 Floating-point types

MySQL	GaussDB	Difference
FLOAT[(M,D)] [ZEROFILL]	Supported, with differences.	The FLOAT data type does not support partitioned tables with the key partitioning policy.
FLOAT(p) [ZEROFILL]	Supported, with differences.	The FLOAT data type does not support partitioned tables with the key partitioning policy.
DOUBLE[(M,D)] [ZEROFILL]	Supported, with differences.	The DOUBLE data type does not support partitioned tables with the key partitioning policy.
DOUBLE PRECISION[(M,D)] [ZEROFILL]	Supported, with differences.	The DOUBLE PRECISION data type does not support partitioned tables with the key partitioning policy.
REAL[(M,D)] [ZEROFILL]	Supported, with differences.	The REAL data type does not support partitioned tables with the key partitioning policy.

3.2.1.2 Date and Time Data Types

Table 3-4 Date and Time Data Types

MySQL	GaussDB	Difference
DATE	Supported, with differences.	GaussDB supports the DATE data type and differs from MySQL in terms of the following specifications: A backslash (\) is regarded as an escape character in both MySQL and GaussDB. However, MySQL supports \0, but GaussDB does not support \0. Therefore, GaussDB reports an error when the backslash is used as a separator and followed by 0.

MySQL	GaussDB	Difference
DATETIME[(fsp)]	Supported, with differences.	GaussDB supports the DATETIME data type and differs from MySQL in terms of the following specifications: A backslash (\) is regarded as an escape character in both MySQL and GaussDB. However, MySQL supports \0, but GaussDB does not support \0. Therefore, GaussDB reports an error when the backslash is used as a separator and followed by 0.
TIMESTAMP[(fsp)]	Supported, with differences.	GaussDB supports the TIMESTAMP data type and differs from MySQL in terms of the following specifications: • A backslash (\) is regarded as an escape character in both MySQL and GaussDB. However, MySQL supports \0, but GaussDB does not support \0. Therefore, GaussDB reports an error when the backslash is used as a separator and followed by 0. • MySQL supports **explicit_defaults_for_timestamp**. When **explicit_defaults_for_timestamp** is set to **off**, setting the default value of the TIMESTAMP column and inserting **NULL** are non-standard behaviors. The default value of **explicit_defaults_for_timestamp** is **off** in MySQL 5.7 and is **on** in MySQL 8.0. GaussDB does not support **explicit_defaults_for_timestamp**. The behavior is the same as that when **explicit_defaults_for_timestamp** is set to **on** in MySQL. For details about **explicit_defaults_for_timestamp**, see the note below the table.
TIME[(fsp)]	Supported, with differences.	GaussDB supports the TIME data type. Compared with MySQL, GaussDB has the following differences in specifications: • A backslash (\) is regarded as an escape character in both MySQL and GaussDB. However, MySQL supports \0, but GaussDB does not support \0. Therefore, GaussDB reports an error when the backslash is used as a separator and followed by 0. • When the hour, minute, second, and nanosecond of the TIME type are 0, the sign bits of GaussDB and MySQL may be different.
YEAR[(4)]	Supported.	-

NOTE

- GaussDB does not support ODBC syntax literals:

  { d 'str' }

  { t 'str' }

  { ts 'str' }
- If you specify a precision for the DATETIME, TIME, or TIMESTAMP data type greater than the maximum precision supported by the data type, GaussDB truncates the precision to the maximum precision supported by the data type, whereas MySQL reports an error.
- In MySQL, when **explicit_defaults_for_timestamp** is set to **off**, the processing logic of the TIMESTAMP columns is as follows:
  - If **NULL** or **NOT NULL** attribute is not explicitly specified for a column, the **NOT NULL** attribute will be automatically added. When a **NULL** value is inserted into such a column, the **NULL** value is replaced with the current timestamp.
  - If the **NULL** attribute is not specified for the first TIMESTAMP column in a table, the **DEFAULT CURRENT_TIMESTAMP** and **ON UPDATE CURRENT_TIMESTAMP** attributes will be automatically added to the column.
  - If the **NULL** attribute is not specified for the second and subsequent TIMESTAMP columns in a table, the **DEFAULT '0000-00-00 00:00:00'** attribute will be automatically added to the columns.
- In MySQL, when **explicit_defaults_for_timestamp** is set to **off**, the processing logic of the TIMESTAMP columns is as follows:
  - When a **NULL** value is inserted into a TIMESTAMP column, the **NULL** value is not replaced with the current timestamp.
  - If **NULL** or **NOT NULL** attribute is not explicitly specified for a column, the **NULL** attribute will be automatically added.
  - When a **NULL** value is inserted into a column with the **NOT NULL** attribute specified, an error is reported in strict mode, and **'0000-00-00 00:00:00'** is inserted in loose mode.
  - **DEFAULT CURRENT_TIMESTAMP** and **ON UPDATE CURRENT_TIMESTAMP** attributes will not be automatically added to any TIMESTAMP columns.

3.2.1.3 String Data Types

Table 3-5 String Data Types

MySQL	GaussDB	Difference
CHAR(M)	Supported, with differences.	Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty.

MySQL	GaussDB	Difference
VARCHAR(M)	Supported, with differences.	Input format: • GaussDB cannot verify the length of parameters and return values of user-defined functions or the length of stored procedure parameters. However, MySQL supports their verification. • GaussDB can verify the length of temporary variables in user-defined functions and stored procedures, and an error or truncation alarm is reported in strict or loose mode. However, MySQL does not support these functions. • After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty.
TINYTEXT	Supported, with differences.	• Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Default value: The default value cannot be set in MySQL 5.7 but can be set in GaussDB and MySQL 8.0. • Primary key: When creating a primary key, you must specify the prefix length in MySQL, but you cannot specify the prefix length in GaussDB. • Index: In MySQL, the TINYTEXT type does not support other index methods except prefix indexes. GaussDB supports these index methods.

MySQL	GaussDB	Difference
TEXT	Supported, with differences.	• Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Default value: The default value cannot be set in MySQL 5.7 but can be set in GaussDB and MySQL 8.0. • Primary key: When creating a primary key, you must specify the prefix length in MySQL, but you cannot specify the prefix length in GaussDB. • Index: In MySQL, the TEXT type does not support other index methods except prefix indexes. GaussDB supports these index methods.
MEDIUMTEXT	Supported, with differences.	• Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Default value: The default value cannot be set in MySQL 5.7 but can be set in GaussDB and MySQL 8.0. • Primary key: When creating a primary key, you must specify the prefix length in MySQL, but you cannot specify the prefix length in GaussDB. • Index: In MySQL, the MEDIUMTEXT type does not support other index methods except prefix indexes. GaussDB supports these index methods.

MySQL	GaussDB	Difference
LONGTEXT	Supported, with differences.	• Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Default value: The default value cannot be set in MySQL 5.7 but can be set in GaussDB and MySQL 8.0. • Primary key: When creating a primary key, you must specify the prefix length in MySQL, but you cannot specify the prefix length in GaussDB. • Index: In MySQL, the LONGTEXT type does not support other index methods except prefix indexes. GaussDB supports these index methods.

3.2.1.4 Binary Data Types

Table 3-6 Binary Data Types

MySQL	GaussDB	Difference
BINARY[(M)]	Supported, with differences.	• Input format: – After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. – If the length of the inserted string is less than the target length, the padding character is 0x20 in GaussDB and 0x00 in MySQL. • Character set: The default character set is the initialized character set of the database. For MySQL, the default character set is BINARY. • Output formats: – When the JDBC protocol is used, a space at the end of the BINARY type is displayed as a space, and that in MySQL is displayed as \x00. – In loose mode, if characters (such as Chinese characters) of the BINARY type exceed *n* bytes, the excess characters will be truncated. MySQL retains the first *n* bytes. However, garbled characters are displayed in the output. – In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned. **NOTE** Due to the differences between GaussDB and MySQL in BINARY fillers and \0 truncation, GaussDB and MySQL have different performance in scenarios such as operator comparison calculation, character string-related system function calculation, index matching, and data import and export. For details about the difference scenarios, see the examples in this section.

MySQL	GaussDB	Difference
VARBINARY(M)	Supported, with differences.	• Input format: If a binary or hexadecimal character string is input, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Character set: The default character set is the initialized character set of the database. For MySQL, the default character set is BINARY. • Output formats: – When the JDBC protocol is used, a space at the end of the BINARY type is displayed as a space, and that in MySQL is displayed as \x00. – In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned.
TINYBLOB	Supported, with differences.	• Input format: – Default value: The syntax of GaussDB allows you to set a default value for a table column to be created. However, this operation is not allowed in MySQL. – After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Primary key: In MySQL, the TINYBLOB type does not support primary keys, but GaussDB supports. • Index: In MySQL, the TINYBLOB type does not support other index methods except prefix indexes. GaussDB supports these index methods. • Foreign key: In MySQL, the TINYTEXT type cannot be used as the referencing column or referenced column of a foreign key, but GaussDB supports this operation. • Output format: In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned.

MySQL	GaussDB	Difference
BLOB	Supported, with differences.	• Input format: – Default value: The syntax of GaussDB allows you to set a default value for a table column to be created. However, this operation is not allowed in MySQL. – After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Primary key: In MySQL, the BLOB type does not support primary keys, but GaussDB supports. • Index: In MySQL, the BLOB type does not support other index methods except prefix indexes. GaussDB supports these index methods. • Foreign key: In MySQL, the TINYTEXT type cannot be used as the referencing column or referenced column of a foreign key, but GaussDB supports this operation. • Output format: In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned.

MySQL	GaussDB	Difference
MEDIUMBLOB	Supported, with differences.	• Input format: – Default value: The syntax of GaussDB allows you to set a default value for a table column to be created. However, this operation is not allowed in MySQL. – After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Primary key: In MySQL, the MEDIUMBLOB type does not support primary keys, but GaussDB supports. • Index: In MySQL, the MEDIUMBLOB type does not support other index methods except prefix indexes. GaussDB supports these index methods. • Foreign key: In MySQL, the TINYTEXT type cannot be used as the referencing column or referenced column of a foreign key, but GaussDB supports this operation. • Output format: In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned.

MySQL	GaussDB	Difference
LONGBLOB	Supported, with differences.	• Value range: a maximum of 1 GB. MySQL supports a maximum of 4 GB minus 1 byte. • Input format: – Default value: The syntax of GaussDB allows you to set a default value for a table column to be created. However, this operation is not allowed in MySQL. – After a binary or hexadecimal character string is entered, GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • Primary key: In MySQL, the LONGBLOB type does not support primary keys, but GaussDB supports. • Index: In MySQL, the LONGBLOB type does not support other index methods except prefix indexes. GaussDB supports these index methods. • Foreign key: In MySQL, the TINYTEXT type cannot be used as the referencing column or referenced column of a foreign key, but GaussDB supports this operation. • Output format: In MySQL 8.0 and later versions, results starting with 0x are returned by default. In GaussDB, results in the format of "\x...\x...\x..." are returned.
BIT[(M)]	Supported, with differences.	Output formats: • All outputs are displayed as binary character strings. MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty. • In MySQL 8.0 and later versions, 0 is added at the beginning of each result by default. In GaussDB, 0 is not added.

Example:

```
-- GaussDB
m_db=# CREATE TABLE test(a BINARY(10)) DISTRIBUTE BY REPLICATION;
CREATE TABLE
m_db=# INSERT INTO test VALUES(0x8000);
INSERT 0 1
m_db=# SELECT hex(a) FROM test;
        hex
----------------------
```



```
 80202020202020202020
(1 row)

m_db=# SELECT * FROM test WHERE hex(a) = 80000000000000000000;
 a
---
(0 rows)

m_db=# CREATE TABLE test2(a BINARY(10)) DISTRIBUTE BY REPLICATION;
CREATE TABLE
m_db=# INSERT INTO test2 VALUES(0x80008000);
INSERT 0 1
m_db=# SELECT hex(a) FROM test2;
        hex
----------------------
 80202020202020202020
(1 row)

m_db=# DROP TABLE test;
DROP TABLE
m_db=# DROP TABLE test2;
DROP TABLE

-- MySQL
mysql> CREATE TABLE test(a BINARY(10));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO test VALUES(0x8000);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT hex(a) FROM test;
+----------------------+
| hex(a)               |
+----------------------+
| 80000000000000000000 |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM test WHERE hex(a) = 80000000000000000000;
+------------+
| a          |
+------------+
| ▒          |
+------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE test2(a binary(10));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test2 VALUES(0x80008000);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT hex(a) FROM test2;
+----------------------+
| hex(a)               |
+----------------------+
| 80008000000000000000 |
+----------------------+
1 row in set (0.00 sec)
mysql> DROP TABLE test;
Query OK, 0 rows affected (0.00 sec)
mysql> DROP TABLE test2;
Query OK, 0 rows affected (0.00 sec)
```


3.2.1.5 Attributes Supported by Data Types

Table 3-7 Attributes Supported by Data Types

MySQL	GaussDB
NULL	Supported.
NOT NULL	Supported.
DEFAULT	Supported.
ON UPDATE	Supported.
PRIMARY KEY	Supported.
AUTO_INCREMENT	Supported.
CHARACTER SET name	Supported.
COLLATE name	Supported.
ZEROFILL	Supported.

When CREATE TABLE AS is used to create a table and default values are set for columns of the VARBINARY type, the command output of **SHOW CREATE TABLE**, **DESC**, or **\d** is different from that of MySQL. The value displayed in GaussDB is a hexadecimal value, but MySQL displays the original value.

Example:

```
m_db=# CREATE TABLE test_int(
       int_col INT
);
m_db=# CREATE TABLE test_varbinary(
       varbinary_col VARBINARY(20) default 'gauss'
) AS SELECT * FROM test_int;
m_db=# SHOW CREATE TABLE test_varbinary;
     Table      |                          Create Table
----------------+-------------------------------------------------------------------------
 test_varbinary | SET search_path = public;                                                +
                | CREATE TABLE test_varbinary (                                           +
                |     varbinary_col varbinary(20) DEFAULT X'6761757373',                  +
                |     int_col integer                                                     +
                | )                                                                       +
                | CHARACTER SET = "UTF8" COLLATE = "utf8mb4_general_ci"                   +
                | WITH (orientation=row, compression=no, storage_type=USTORE, segment=off);
(1 row)
m_db=# DROP TABLE test_int, test_varbinary;

mysql> CREATE TABLE test_int(
       int_col INT
);
mysql> CREATE TABLE test_varbinary(
       varbinary_col VARBINARY(20) default 'gauss'
) AS SELECT * FROM test_int;
mysql> SHOW CREATE TABLE test_varbinary;
+----------------
+-------------------------------------------------------------------------------------------------------------------+
| Table          | Create
Table                                                                                                              |
+----------------
+-------------------------------------------------------------------------------------------------------------------+
```



```
| test_varbinary | CREATE TABLE `test_varbinary` (
  `varbinary_col` varbinary(20) DEFAULT 'gauss',
  `int_vol` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 |
+---------------
+-----------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> DROP TABLE test_int, test_varbinary;
```

3.2.1.6 Data Type Conversion

Conversion between different data types is supported. Data type conversion is involved in the following scenarios:

- The data types of operands of operators (such as comparison and arithmetic operators) are inconsistent. It is commonly used for comparison operations in query or join conditions.
- The data types of arguments and parameters are inconsistent when a function is called.
- The data types of target columns to be updated by DML statements (including INSERT, UPDATE, MERGE, and REPLACE) and the defined column types are inconsistent.
- Explicit type conversion: CAST(expr AS datatype), which converts an expression to a data type.
- After the target data type of the final projection column is determined by set operations (UNION and EXCEPT), the type of the projection column in each SELECT statement is inconsistent with the target data type.
- In other expression calculation scenarios, the target data type used for comparison or final result is determined based on the data type of different expressions.

There are three types of data type conversion differences: implicit conversion, explicit conversion, and UNION/CASE.

Differences in Double Colon Conversion

- In GaussDB, if you use double colons to convert input parameters of a function to another type, the result may be unexpected. In MySQL, double colons do not take effect.

  Example:

```
m_db=# SELECT POW("12"::VARBINARY,"12"::VARBINARY);
ERROR:  value out of range: overflow
CONTEXT:  referenced column: pow

varbinary col
m_db=# CREATE TABLE test_varbinary (
       A VARBINARY(10)
);
m_db=# INSERT INTO test_varbinary VALUES ('12');
m_db=# SELECT POW(A, A) FROM test_varbinary;
     pow
---------------
 8916100448256
(1 row)
```


Differences in Implicit Type Conversion

- In GaussDB, the conversion rules from small types to small types are used. In MySQL, the conversion rules from small types to large types and from large types to small types are used.
- Due to data type differences, some output formats of implicit conversion in GaussDB are inconsistent.
- During implicit conversion from the BIT data type to the character data type and binary data type in GaussDB, some output behaviors are inconsistent. GaussDB outputs a hexadecimal character string, and MySQL escapes the character string based on the ASCII code table. If the character string cannot be escaped, the output is empty.

  Example:

```
m_db=# CREATE TABLE bit_storage (
    VS_COL1 BIT(4),
    VS_COL2 BIT(4),
    VS_COL3 BIT(4),
    VS_COL4 BIT(4),
    VS_COL5 BIT(4),
    VS_COL6 BIT(4),
    VS_COL7 BIT(4),
    VS_COL8 BIT(4)
) DISTRIBUTE BY REPLICATION;
m_db=# CREATE TABLE string_storage (
    VS_COL1 BLOB,
    VS_COL2 TINYBLOB,
    VS_COL3 MEDIUMBLOB,
    VS_COL4 LONGBLOB,
    VS_COL5 TEXT,
    VS_COL6 TINYTEXT,
    VS_COL7 MEDIUMTEXT,
    VS_COL8 LONGTEXT
) DISTRIBUTE BY REPLICATION;
m_db=# INSERT INTO bit_storage VALUES(B'101', B'101', B'101', B'101', B'101', B'101', B'101', B'101');
m_db=# INSERT INTO string_storage SELECT * FROM bit_storage;
m_db=# SELECT * FROM string_storage;
 VS_COL1 | VS_COL2 | VS_COL3 | VS_COL4 | VS_COL5 | VS_COL6 | VS_COL7 | VS_COL8
---------+---------+---------+---------+---------+---------+---------+---------
 \x05    | \x05    | \x05    | \x05    | \x05    | \x05    | \x05    | \x05
(1 row)
m_db=# DROP TABLE bit_storage, string_storage;

mysql> CREATE TABLE bit_storage (
    VS_COL1 BIT(4),
    VS_COL2 BIT(4),
    VS_COL3 BIT(4),
    VS_COL4 BIT(4),
    VS_COL5 BIT(4),
    VS_COL6 BIT(4),
    VS_COL7 BIT(4),
    VS_COL8 BIT(4)
);
mysql> CREATE TABLE bit_storage (
    VS_COL1 BIT(4),
    VS_COL2 BIT(4),
    VS_COL3 BIT(4),
    VS_COL4 BIT(4),
    VS_COL5 BIT(4),
    VS_COL6 BIT(4),
    VS_COL7 BIT(4),
    VS_COL8 BIT(4)
);
mysql> INSERT INTO bit_storage VALUES(B'101', B'101', B'101', B'101', B'101', B'101', B'101', B'101');
mysql> INSERT INTO string_storage SELECT * FROM bit_storage;
mysql> SELECT * FROM string_storage;
```



```
+---------+---------+---------+---------+---------+---------+---------+---------+
| VS_COL1 | VS_COL2 | VS_COL3 | VS_COL4 | VS_COL5 | VS_COL6 | VS_COL7 | VS_COL8 |
+---------+---------+---------+---------+---------+---------+---------+---------+
|       |       |       |       |       |       |       |       |
+---------+---------+---------+---------+---------+---------+---------+---------+
1 row in set (0.00 sec)
mysql> DROP TABLE bit_storage, string_storage;
```

- When a binary or hexadecimal character string with 0x00 is inserted into the binary data type, GaussDB inserts part of the string and truncates the characters following 0x00. MySQL can insert the entire string.

  Example:

```
m_db=# CREATE TABLE blob_storage (
        A BLOB
) DISTRIBUTE BY REPLICATION;
m_db=# INSERT INTO blob_storage VALUES (0xBB00BB);
m_db=# SELECT hex(A) FROM blob_storage;
 hex
-----
 BB
(1 row)
m_db=# DROP TABLE blob_storage;

mysql> CREATE TABLE blob_storage (
        A BLOB
);
mysql> INSERT INTO blob_storage VALUES (0xBB00BB);
mysql> SELECT hex(A) FROM blob_storage;
+--------+
| hex(a) |
+--------+
| BB00BB |
+--------+
1 row in set (0.01 sec)
mysql> DROP TABLE blob_storage;
```

- When a binary or hexadecimal string with 0x00 in the middle is inserted into the string data type, GaussDB inserts part of the string and truncates the characters following 0x00. In MySQL, the string cannot be inserted in strict mode, and an empty string is inserted in loose mode.

  Example:

```
m_db=# CREATE TABLE text_storage (
        A TEXT
);
m_db=# INSERT INTO text_storage VALUES (b'101110110000000010111011');
m_db=# SELECT hex(A) FROM text_storage;
 hex
-----
 BB
(1 row)
m_db=# DROP TABLE text_storage;

mysql> CREATE TABLE text_storage (
        A TEXT
);
mysql> INSERT INTO text_storage VALUES (b'101110110000000010111011');
ERROR 1366 (HY000): Incorrect string value: '\xBB\x00\xBB' for column 'A' at row 1
mysql> SELECT hex(A) FROM text_storage;
Empty set (0.00 sec)
mysql> SET SQL_MODE='';
mysql> INSERT INTO text_storage VALUES (b'101110110000000010111011');
mysql> SELECT hex(A) FROM text_storage;
+--------+
| hex(A) |
+--------+
|        |
+--------+
```



```
1 row in set (0.01 sec)
mysql> DROP TABLE text_storage;
```

- The WHERE clause contains only common character strings. GaussDB returns **TRUE** for **'t'**, **'true'**, **'yes'**, **'y'**, and **'on'**, returns **FALSE** for **'no'**, **'f'**, **'off'**, **'false'**, and **'n'**, and reports an error for other character strings. MySQL determines whether to return **TRUE** or **FALSE** by converting a character string to an INT1 value.

  Example:

```
m_db=# CREATE TABLE test_where (
       A INT
);
m_db=# INSERT INTO test_where VALUES (1);
m_db=# SELECT * FROM test_where WHERE '111';
ERROR:  invalid input syntax for type boolean: "111"
LINE 1: SELECT * FROM test_where WHERE '111';
m_db=# DROP TABLE test_where;

mysql> CREATE TABLE test_where (
       A INT
);
mysql> INSERT INTO test_where VALUES (1);
mysql> SELECT * FROM test_where WHERE '111';
+------+
| a    |
+------+
|    1 |
+------+
1 row in set (0.01 sec)
mysql> DROP TABLE test_where;
```

- When converting strings of the YEAR type to integers, MySQL uses scientific notation, but GaussDB does not support scientific notation and truncates the strings.

  Example:

```
m_db=# CREATE TABLE test_year (
       A YEAR
);
m_db=# SET sql_mode = '';
m_db=# INSERT INTO test_year VALUES ('2E3x');
WARNING:  Data truncated for column.
LINE 1: INSERT INTO test_year VALUES ('2E3x');
                                      ^
CONTEXT:  referenced column: a
m_db=# SELECT * FROM test_year ORDER BY A;
  a
------
 2002
(1 row)
m_db=# DROP TABLE test_year;

mysql> CREATE TABLE test_year (
       A YEAR
);
mysql> INSERT INTO test_year VALUES ('2E3x');
mysql> SELECT * FROM test_year ORDER BY A;
+------+
| a    |
+------+
| 2000 |
+------+
1 row in set (0.01 sec)
mysql> DROP TABLE test_year;
```


Differences in Explicit Type Conversion

- In GaussDB, the conversion rules for each target type are used. In MySQL, C++ polymorphic overloading functions are used, causing inconsistent behavior in nesting scenarios.

  Example:

```
m_db=# SELECT CAST(GREATEST(date'2023-01-01','2023-01-01') AS SIGNED);
WARNING:  Truncated incorrect INTEGER value: '2023-01-01'
CONTEXT:  referenced column: cast
 cast
------
 2023
(1 row)

mysql> SELECT CAST(GREATEST(date'2023-01-01','2023-01-01') AS SIGNED);
+-----------------------------------------------------------+
| CAST(GREATEST(date'2023-01-01','2023-01-01') AS SIGNED) |
+-----------------------------------------------------------+
|                                                  20230101 |
+-----------------------------------------------------------+
```

Differences Between UNION, CASE, and Related Structures

- In MySQL, POLYGON+NULL, POINT+NULL, and POLYGON+POINT return the GEOMETRY type. They are not involved in GaussDB and considered as errors.
- The SET and ENUM types are not supported currently and are considered as errors.
- When the constant type is aggregated with other types, the precision of the output type is the precision of other types. For example, the precision of the result of "SELECT "helloworld" UNION SELECT p FROM t;" is the precision of attribute p.
- When fixed-point constants and types without precision constraints (non-string types such as int, bool, and year, and the type of the aggregation result is the fixed-point type) are aggregated, the precision constraint is output based on the default precision 31 of fixed-point numbers.
- Differences in merge rules:

  In MySQL 5.7, if YEAR is aggregated with TINYINT, INT, MEDIUMINT, BIGINT, or BOOL, the result is of the type with UNSIGNED. In GaussDB, it is of the type without UNSIGNED. In MySQL, if BIT is aggregated with a numeric type such as INT, NUMERIC, FLOAT, or DOUBLE, the result type is VARBINARY. In GaussDB, the result type is NUMERIC for aggregation between BIT and INT or NUMERIC, DOUBLE for aggregation between BIT and FLOAT or DOUBLE, and UINT8 for aggregation between BIT and unsigned integers.
- In MySQL, BINARY and CHAR use different padding characters. BINARY is padded with '\0', and CHAR is padded with spaces. In GaussDB, BINARY and CHAR are padded with spaces.

3.2.2 System Functions

3.2.2.1 System Function Compatibility Overview

GaussDB is compatible with most MySQL system functions, but there are some differences.

Currently, some system functions in GaussDB with the same names as those in MySQL are not supported in M-compatible mode. For some of them, the message indicating that they are not supported in M-compatible mode is displayed. Other functions still retain the behaviors of the original GaussDB system functions. The behavior of functions with the same name is greatly different from that of MySQL. Therefore, you are advised to avoid using them but use only system functions in M-compatible mode.

The following table lists the functions with the same name.

Table 3-8 Same-name functions for which a message indicating that they are not supported in M-compatible mode is displayed

cot	isEmpty	last_insert_id	mod	octet_length
overlaps	point	radians	regexp_instr	regexp_like
regexp_replace	regexp_substr	stddev_pop	stddev_samp	var_pop
var_samp	variance	-	-	-

Table 3-9 Same-name functions that retain the behaviors of the original GaussDB system functions in M-compatible mode

ceil	decode	encode	format	instr
position	round	stddev	row_num	-

NOTE

- MySQL allows you to add user-defined functions to the database through the loadable functions. When such functions are called, aliases can be specified in the input parameters of the functions. GaussDB does not support loadable functions. When a function is called, aliases cannot be specified for input parameters of the function.
- In M-compatible mode, system functions have the following differences:
  - The return value type of a system function is the same as that of MySQL only when the node type of the input parameter is Var (table data) or Const (constant input). In other cases (for example, the input parameter is a calculation expression or function expression), the return value type may be different from that of MySQL.
  - When an aggregate function uses an expression such as another function, operator, or SELECT clause as the input parameter (for example, **SELECT sum(abs(n)) FROM t;**), the aggregate function cannot obtain the precision passed by the input parameter expression. As a result, the precision of the function result is different from that of MySQL.
  - Calling system functions by pg_catalog.func_name() is not recommended. If the called function has input parameters in the format of syntax (such as SELECT substr('demo' from 1 for 2)), an error may occur when the function is called.

3.2.2.2 Flow Control Functions

Table 3-10 Flow control functions

MySQL	GaussDB	Difference
IF()	Supported, with differences.	If the first parameter is **TRUE** and the third parameter expression contains an implicit type conversion error, or if the first parameter is **FALSE** and the second parameter expression contains an implicit type conversion error, MySQL ignores the error while GaussDB displays a type conversion error.
IFNULL()	Supported, with differences.	If the first parameter is not **NULL** and the expression of the second parameter contains an implicit type conversion error, MySQL ignores the error while GaussDB reports a type conversion error.
NULLIF()	Supported, with differences.	The return value type of a function differs in MySQL 5.7 and MySQL 8.0. Return types are compatible with MySQL 8.0 because it is more appropriate.

3.2.2.3 Date and Time Functions

The date and time functions in the M-compatible mode in GaussDB, with the same behavior as MySQL, are described as follows:

- Functions may use time expressions as their input parameters.

  Time expressions (mainly including TEXT, DATETIME, DATE, and TIME) and types that can be implicitly converted to time expressions can be used as input parameters. For example, a number can be implicitly converted to text and then used as a time expression.

  However, different functions take effect in different ways. For example, the DATEDIFF function calculates only the difference between dates. Therefore, the time expression is parsed as the date type. The TIMESTAMPDIFF function parses the time expression as DATE, TIME, or DATETIME based on the **UNIT** parameter before calculating the time difference.

- The input parameters of functions may contain an invalid date.

  Generally, the supported DATE and DATETIME ranges are the same as those in MySQL. The value of DATE ranges from '0000-01-01' to '9999-12-31', and the value of DATETIME ranges from '0000-01-01 00:00:00' to '9999-12-31 23:59:59'. Although the DATE and DATETIME ranges supported by GaussDB are greater than those supported by MySQL, out-of-bounds dates are still invalid.

  In most cases, time functions report an alarm and return NULL if the input date is invalid, unless the invalid date can be converted by CAST.

Most date and time functions in the GaussDB M-compatible framework are the same as those in MySQL. The following table lists the differences between them in terms of some functions.

Table 3-11 Date and time functions

MySQL	GaussDB	Difference
ADDDATE()	Supported.	-
ADDTIME()	Supported.	-
CONVERT_TZ()	Supported.	-
CURDATE()	Supported.	-
CURRENT_DATE()/ CURRENT_DATE	Supported.	-
CURRENT_TIME()/ CURRENT_TIME	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT CURRENT_TIME(257) == SELECT CURRENT_TIME(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
CURRENT_TIMESTAMP()/ CURRENT_TIMESTAMP	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT CURRENT_TIMESTAMP(257) == SELECT CURRENT_TIMESTAMP(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
CURTIME()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT CURTIME(257) == SELECT CURTIME(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
DATE()	Supported.	-
DATE_ADD()	Supported.	-
DATE_FORMAT()	Supported.	-

MySQL	GaussDB	Difference
DATE_SUB()	Supported.	-
DATEDIFF()	Supported.	-
DAY()	Supported.	-
DAYNAME()	Supported.	-
DAYOFMONTH()	Supported.	-
DAYOFWEEK()	Supported.	-
DAYOFYEAR()	Supported.	-
EXTRACT()	Supported.	-
FROM_DAYS()	Supported.	-
FROM_UNIXTIME()	Supported.	-
GET_FORMAT()	Supported.	-
HOUR()	Supported.	-
LAST_DAY()	Supported.	-
LOCALTIME()/LOCALTIME	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT LOCALTIME(257) == SELECT LOCALTIME(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
LOCALTIMESTAMP/ LOCALTIMESTAMP()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT LOCALTIMESTAMP(257) == SELECT LOCALTIMESTAMP(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
MAKEDATE()	Supported.	-
MAKETIME()	Supported, with differences.	In the distributed pushdown scenario, if no second precision is specified for the TIME type, MySQL supplements six trailing zeros by default, but GaussDB does not supplement anything.

MySQL	GaussDB	Difference
MICROSECOND()	Supported.	-
MINUTE()	Supported.	-
MONTH()	Supported.	-
MONTHNAME()	Supported.	-
NOW()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), for example, **SELECT NOW(257)==SELECT NOW(1)**. GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
PERIOD_ADD()	Supported, with differences.	• Processing of integer overflow. In MySQL 5.7, the maximum value of an input parameter result of this function is 2^32=4294967296. When the accumulated value of the month corresponding to **period** and the value of **month_number** in the input parameter or result exceed the uint32 range, integer wraparound occurs. This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0. • Performance when the value of **period** is negative: In MySQL 5.7, a negative year is parsed as an abnormal value instead of an error. Conversely, GaussDB reports an error when any input parameter or result is negative (for example, January 100 minus 10000 months). This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0. • Performance when the month in **period** exceeds the range: When dealing with a month greater than 12 or equal to 0, for example, **200013** or **199900**, MySQL 5.7 postpones it to the next year or views month 0 as December of the previous year. GaussDB reports an error for months beyond the range. This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0.

MySQL	GaussDB	Difference
PERIOD_DIFF()	Supported, with differences.	• Processing of integer overflow. In MySQL 5.7, the maximum value of an input parameter result of this function is 2^32=4294967296. When the accumulated value of the month corresponding to **period** and the value of **month_number** in the input parameter or result exceed the uint32 range, integer wraparound occurs. This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0. • Performance when the value of **period** is negative: In MySQL 5.7, a negative year is parsed as an abnormal value instead of an error. Conversely, GaussDB reports an error when any input parameter or result is negative (for example, January 100 minus 10000 months). This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0. • Performance when the month in **period** exceeds the range: When dealing with a month greater than 12 or equal to 0, for example, **200013** or **199900**, MySQL 5.7 postpones it to the next year or views month 0 as December of the previous year. GaussDB reports an error for months beyond the range. This issue has been resolved in MySQL 8.0. The performance of this function in GaussDB is the same as that in MySQL 8.0.
QUARTER()	Supported.	-
SEC_TO_TIME()	Supported.	-
SECOND()	Supported.	-
STR_TO_DATE()	Supported, with differences.	GaussDB returns values of the text type, while MySQL returns values of the datetime or date type.
SUBDATE()	Supported.	-
SUBTIME()	Supported.	-
SYSDATE()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value). GaussDB does not support wraparound.

MySQL	GaussDB	Difference
TIME()	Supported.	-
TIME_FORMAT()	Supported.	-
TIME_TO_SEC()	Supported.	-
TIMEDIFF()	Supported.	-
TIMESTAMP()	Supported.	-
TIMESTAMPADD()	Supported.	-
TIMESTAMPDIFF()	Supported.	-
TO_DAYS()	Supported.	-
TO_SECONDS()	Supported.	-
UNIX_TIMESTAMP()	Supported, with differences.	MySQL determines whether to return a fixed-point value or an integer based on whether an input parameter contains decimal places. When operators or functions are nested in the input parameter, GaussDB may return a value of the type different from that in MySQL. If the inner node returns a value of the fixed-point, floating-point, string, or time type (excluding the date type), MySQL may return an integer, while GaussDB returns a fixed-point value.
UTC_DATE()	Supported.	-
UTC_TIME()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value). GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
UTC_TIMESTAMP()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value). GaussDB supports only valid values ranging from 0 to 6. For other values, an error is reported.
WEEK()	Supported.	-
WEEKDAY()	Supported.	-
WEEKOFYEAR()	Supported.	-

MySQL	GaussDB	Difference
YEAR()	Supported.	-
YEARWEEK()	Supported.	-

3.2.2.4 String Functions

Table 3-12 String functions

MySQL	GaussDB	Difference
ASCII()	Supported.	-
BIT_LENGTH()	Supported.	-
CHAR_LENGTH()	Supported, with differences.	In GaussDB, if the character set is SQL_ASCII, CHAR_LENGTH() returns the number of bytes instead of characters.
CHARACTER_LENGTH()	Supported, with differences.	In GaussDB, if the character set is SQL_ASCII, CHARACTER_LENGTH() returns the number of bytes instead of characters.
CONCAT()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
CONCAT_WS()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
HEX()	Supported.	-
LENGTH()	Supported.	-

MySQL	GaussDB	Difference
LPAD()	Supported, with differences.	• The default maximum padding length in MySQL is **1398101**, and that in GaussDB is **1048576**. The maximum padding length varies depending on the character set. For example, if the character set is GBK, the default maximum padding length in GaussDB is **2097152**. • When GaussDB uses the SQL_ASCII, the server 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 the input and output of the function contain any non-ASCII data, the database cannot convert or verify non-ASCII characters. As a result, the behavior of the function is greatly different from that of MySQL. • For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
REPEAT()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
REPLACE()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.

MySQL	GaussDB	Difference
RPAD()	Supported, with differences.	• The default maximum padding length in MySQL is **1398101**, and that in GaussDB is **1048576**. The maximum padding length varies depending on the character set. For example, if the character set is GBK, the default maximum padding length in GaussDB is **2097152**. • When GaussDB uses the SQL_ASCII, the server 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 the input and output of the function contain any non-ASCII data, the database cannot convert or verify non-ASCII characters. As a result, the behavior of the function is greatly different from that of MySQL. • For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
SPACE()	Supported.	-
STRCMP()	Supported, with differences.	When GaussDB uses the SQL_ASCII, the server 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 the input and output of the function contain any non-ASCII data, the database cannot convert or verify non-ASCII characters. As a result, the behavior of the function is greatly different from that of MySQL.

MySQL	GaussDB	Difference
FIND_IN_SET()	Supported, with differences.	When GaussDB uses the SQL_ASCII, the server 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 the input and output of the function contain any non-ASCII data, the database cannot convert or verify non-ASCII characters. As a result, the behavior of the function is greatly different from that of MySQL. For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT.
LCASE()		
LEFT()		
LOWER()		
LTRIM()		
REVERSE()		
RIGHT()		
RTRIM()		
SUBSTR()		
SUBSTRING()		
SUBSTRING_INDEX()		
TRIM()		
UCASE()		
UPPER()		
UNHEX()	Supported, with differences.	The return value type in MySQL is BINARY, VARBINARY, BLOB, MEDIUMBLOB, or LONGBLOB, while the return value type in GaussDB is fixed to LONGBLOB.
FIELD()	Supported.	-
FORMAT()	Supported.	-

3.2.2.5 Forced Conversion Functions

Table 3-13 Forced conversion functions

MySQL	GaussDB	Difference

CAST()	Supported, with differences	• In GaussDB, CAST(expr AS CHAR[(N)] charset_info or CAST(expr AS NCHAR[(N)]) cannot be used to convert character sets.
		• In GaussDB, you can use **CAST(expr AS FLOAT[(p)])** or **CAST(expr AS DOUBLE)** to convert an expression to the one of the floating-point type. MySQL 5.7 does not support this conversion.
		• In GaussDB, CAST(expr AS JSON) cannot be used to convert expressions to JSON.
		• In the CAST nested subquery scenario, if the subquery statement returns the FLOAT type, an accurate value is returned in GaussDB while a distorted value is returned in MySQL 5.7. The same rule applies to the BINARY function implemented using CAST.
		--GaussDB
		m_db=# CREATE TABLE sub_query_table(myfloat float) DISTRIBUTE BY REPLICATION;
		CREATE TABLE
		m_db=# INSERT INTO sub_query_table(myfloat) VALUES (1.23);
		INSERT 0 1
		m_db=# SELECT binary(SELECT myfloat FROM sub_query_table) FROM sub_query_table;
		binary
		-------- 	
		1.23
		(1 row)
		m_db=# SELECT cast((SELECT myfloat FROM sub_query_table) AS char);
		cast
		------ 	
		1.23
		(1 row)
		--MySQL 5.7
		mysql> CREATE TABLE sub_query_table(myfloat float);
		Query OK, 0 rows affected (0.02 sec)
		mysql> INSERT INTO sub_query_table(myfloat) VALUES (1.23);
		Query OK, 1 row affected (0.00 sec)
		mysql> SELECT binary(SELECT myfloat FROM sub_query_table) FROM sub_query_table;
		+--+
		| binary(SELECT myfloat FROM sub_query_table) |
		+--+
		| 1.2300000190734863 |
		+--+
		1 row in set (0.00 sec)
		mysql> SELECT cast((SELECT myfloat FROM sub_query_table) AS char);
		+---+
		| cast((SELECT myfloat FROM sub_query_table) AS char) |
		+---+
		| 1.2300000190734863 |

MySQL	GaussDB	Difference
		+----------+ 1 row in set (0.00 sec)
CONVERT()	Supported, with differences	• In GaussDB, CONVERT(expr, CHAR[(N)] charset_info or CAST(expr, NCHAR[(N)]) cannot be used to convert character sets. • In GaussDB, you can use **CONVERT(expr, FLOAT[(p)])** or **CONVERT(expr, DOUBLE)** to convert an expression to the one of the floating-point type. MySQL 5.7 does not support this conversion. • In GaussDB, CONVERT(expr, JSON) cannot be used to convert expressions to JSON.

3.2.2.6 Encryption Functions

Table 3-14 Encryption functions

MySQL	GaussDB	Difference
AES_DECRYPT()	Supported, with differences	• GaussDB does not support ECB mode, which is an insecure encryption mode, but uses CBC mode by default. • When characters are specified to be encoded in SQL_ASCII for GaussDB, the server parses byte values 0 to 127 according to the ASCII standard, and byte values 128 to 255 cannot be parsed. If the input and output of the function contain any non-ASCII characters, the database cannot convert or verify them. • The return value type in MySQL is BINARY, VARBINARY, BLOB, MEDIUMBLOB, or LONGBLOB, while the return value type in GaussDB is fixed to LONGBLOB.
AES_ENCRYPT()	Supported, with differences	
SHA()/ SHA1()	Supported.	-
SHA2()	Supported.	-

3.2.2.7 Comparison Functions

Table 3-15 Comparison functions

MySQL	GaussDB	Difference
COALESCE()	Supported, with differences.	In the union distinct scenario, the precision of the return value is different from that in MySQL. If there is an implicit type conversion error in the subsequent parameter expression of the first parameter that is not **NULL**, MySQL ignores the error while GaussDB displays a type conversion error. When the parameter is a MIN or MAX function, the return value type is different from that in MySQL.
INTERVAL()	Supported.	-
GREATEST()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT. If the input parameter of the function contains **NULL** and the function is called after the WHERE keyword, the returned result is inconsistent with that of MySQL 5.7. This problem lies in MySQL 5.7. Since MySQL 8.0 has resolved this problem, GaussDB are consistent with MySQL 8.0.
LEAST()	Supported, with differences.	For binary return values, MySQL offers various options (including BINARY, VARBINARY, and BLOB), while GaussDB offers only one—LONGBLOB. For non-binary return values, MySQL offers various options (including CHAR, VARCHAR, and TEXT), while GaussDB only offers TEXT. If the input parameter of the function contains **NULL** and the function is called after the WHERE keyword, the returned result is inconsistent with that of MySQL 5.7. This problem lies in MySQL 5.7. Since MySQL 8.0 has resolved this problem, GaussDB are consistent with MySQL 8.0.
ISNULL()	Supported.	-

3.2.2.8 Aggregate Functions

Table 3-16 Aggregate functions

MySQL	GaussDB	Difference
AVG()	Supported, with differences.	• If **DISTINCT** is specified and the SQL statement contains the GROUP BY clause, GaussDB does not sort the results, while MySQL sorts the results. • In GaussDB, if the columns in **expr** are of the BIT, BOOL, or integer type and the sum of all rows exceeds the range of BIGINT, overflow occurs, reversing integers.
BIT_AND()	Supported.	-
BIT_OR()	Supported.	-
BIT_XOR()	Supported.	-
COUNT()	Supported, with differences.	If **DISTINCT** is specified and the SQL statement contains the GROUP BY clause, GaussDB does not sort the results, while MySQL sorts the results.
GROUP_CONCAT()	Supported, with differences.	• If **DISTINCT** is specified and the SQL statement contains the GROUP BY clause, GaussDB does not sort the results, while MySQL sorts the results. • If GROUP_CONCAT is to return a binary type, only the BLOB type is returned. In other cases, the TEXT type is returned. MySQL may return the LONGTEXT, TINYTEXT, LONGBLOB, or TINYBLOB type based on the return length. • In GaussDB, if the parameters in GROUP_CONCAT contain both the DISTINCT and ORDER BY syntaxes, all expressions following ORDER BY must be in the DISTINCT expression. • In GaussDB, **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. • In GaussDB, the **group_concat_max_len** parameter is used to limit the maximum return length of GROUP_CONCAT. If the return length exceeds the maximum, the length is truncated. Currently, the maximum length that can be returned is **1073741823**, which is smaller than that in MySQL.

<table>
<tr><th>MySQL</th><th>GaussDB</th><th>Difference</th></tr>
<tr><td>MAX()
MIN()</td><td>Supported, with differences.</td><td>

If DISTINCT is specified and the SQL statement contains the GROUP BY clause, GaussDB does not sort the results, while MySQL sorts the results.
When the parameter is not a table column, the return value types of the MAX and MIN functions are different from those in MySQL 5.7.
When the parameter is of the FLOAT type, the return values of the MAX and MIN functions are the same as those in MySQL 5.7. The behavior of MySQL 5.7 and MySQL 8.0 is different. As a result, the return values of the MAX and MIN functions with CAST(expr AS FLOAT[(p)]) nested are different from those in MySQL 8.0.

<pre>
-- GaussDB:
m_db=# CREATE TABLE t1(c1 float);
CREATE TABLE

m_db=# INSERT INTO t1 VALUES(1.2);
INSERT 0 1

m_db=# SELECT MAX(c1) FROM t1;
        max

 1.2000000476837158
(1 row)

m_db=# SELECT MAX(CAST(1.2 AS FLOAT));
        max

 1.2000000476837158
(1 row)

m_db=# DROP TABLE t1;
DROP TABLE

-- MySQL 5.7:
mysql> CREATE TABLE t1(c1 float);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO t1 VALUES(1.2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT MAX(c1) FROM t1;
+--------------------+
| MAX(c1) |
+--------------------+
| 1.2000000476837158 |
+--------------------+
1 row in set (0.00 sec)
-- MySQL 5.7 does not support the CAST(expr AS FLOAT[(p)]) expression.
mysql> SELECT MAX(CAST(1.2 AS FLOAT));
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'FLOAT))' at line 1
</pre>
</td></tr>
</table>

MySQL	GaussDB	Difference

		mysql> DROP TABLE t1;
		Query OK, 0 rows affected (0.01 sec)
		
		-- MySQL 8.0:
		mysql> CREATE TABLE t1(c1 float);
		Query OK, 0 rows affected (0.03 sec)
		
		mysql> INSERT INTO t1 VALUES(1.2);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT MAX(c1) FROM t1;
		+---------+
		| MAX(c1) |
		+---------+
		| 1.2 |
		+---------+
		1 row in set (0.00 sec)
		
		mysql> SELECT MAX(CAST(1.2 AS FLOAT));
		+-------------------------+
		| MAX(CAST(1.2 AS FLOAT)) |
		+-------------------------+
		| 1.2 |
		+-------------------------+
		1 row in set (0.00 sec)
		
		mysql> DROP TABLE t1;
		Query OK, 0 rows affected (0.01 sec)

SUM()	Supported, with differences.	• If **DISTINCT** is specified and the SQL statement contains the GROUP BY clause, GaussDB does not sort the results, while MySQL sorts the results. • In GaussDB, if the columns in **expr** are of the BIT, BOOL, or integer type and the sum of all rows exceeds the range of BIGINT, overflow occurs, reversing integers.

3.2.2.9 Numeric Operation Functions

Table 3-17 Numeric operation functions

MySQL	GaussDB	Difference
ABS()	Supported.	-
ACOS()	Supported.	-
ASIN()	Supported.	-
ATAN()	Supported.	-
ATAN2()	Supported.	-

MySQL	GaussDB	Difference
CEILING()	Supported, with differences.	Some operation result types in GaussDB are inconsistent with those in MySQL. If the derived result is of the NUMERIC or integer type and can be stored as an integer type, the result type in MySQL is integer, but is still NUMERIC in GaussDB.
COS()	Supported.	-
DEGREES()	Supported.	-
EXP()	Supported.	-
FLOOR()	Supported, with differences.	The return value types of the FLOOR function in GaussDB are different from those in MySQL. When the input parameter type is INT, the return value type is BIGINT in GaussDB, but is INT in MySQL. Some operation result types in GaussDB are inconsistent with those in MySQL. If the derived result is of the NUMERIC or integer type and can be stored as an integer type, the result type in MySQL is integer, but is still NUMERIC in GaussDB.
LN()	Supported.	-
LOG()	Supported.	-
LOG10()	Supported.	-
LOG2()	Supported.	-
PI()	Supported, with differences.	The precision of the return value of the PI function in GaussDB is different from that in MySQL. It is rounded off to 15 decimal places in GaussDB but to six decimal places in MySQL.
POW()	Supported.	-
POWER()	Supported.	-
RAND()	Supported.	-
SIGN()	Supported.	-
SIN()	Supported.	-
SQRT()	Supported.	-
TAN()	Supported.	-
TRUNCATE()	Supported.	-
CEIL()	Supported.	-

3.2.2.10 Other Functions

Table 3-18 Other functions

MySQL	GaussDB	Difference
DATABASE()	Supported.	-
UUID()	Supported.	-
UUID_SHORT()	Supported.	-

3.2.3 Operators

GaussDB is compatible with most MySQL operators, but there are some differences. If not listed, the operator behavior is the native behavior of GaussDB by default. Currently, there are statements that are not supported by MySQL but supported by GaussDB. In MySQL-compatible mode, they are usually used inside the system, so they are not recommended.

Operator Differences

- NULL values in ORDER BY are sorted in different ways. MySQL sorts NULL values first, while GaussDB sorts NULL values last. In GaussDB, you can use **NULLS FIRST** and **NULLS LAST** to set the sorting sequence of NULL values.
- If ORDER BY is used, the output sequence of GaussDB is the same as that of MySQL. Without ORDER BY, GaussDB does not guarantee that the results are ordered.
- When using MySQL operators, use parentheses to ensure the combination of expressions. Otherwise, an error is reported. For example, SELECT 1 regexp ('12345' regexp '123').

  The GaussDB M-compatible operators can be successfully executed without using parentheses to strictly combine expressions.
- NULL values are displayed in different ways. MySQL displays a NULL value as **"NULL"**. GaussDB displays a NULL value as empty.

  MySQL output:

  ```
  mysql> SELECT NULL;
  +------+
  | NULL |
  +------+
  | NULL |
  +------+
  1 row in set (0.00 sec)
  ```

  GaussDB output:

  ```
  m_db=# SELECT NULL;
   ?column?
  ----------

  (1 row)
  ```


- After the operator is executed, the column names are displayed in different ways. MySQL displays a NULL value as **"NULL"**. GaussDB displays a NULL value as empty.
- When character strings are being converted to the double type but there is an invalid one, the alarm is reported differently. MySQL reports an error when there is an invalid constant character string, but does not report an error for an invalid column character string. GaussDB reports an error in either situation.
- The results returned by the comparison operator are different. For MySQL, **1** or **0** is returned. For GaussDB, **t** or **f** is returned.

Table 3-19 Operators

MySQL	GaussDB	Difference
<>	Supported, with differences.	MySQL supports indexes, but GaussDB does not.
<=>	Supported, with differences.	MySQL supports indexes, but GaussDB does not support indexes, hash joins, or merge joins.

MySQL	GaussDB	Difference

Row expressions	Supported, with differences.	• MySQL supports row comparison using the <=> operator, but GaussDB does not support row comparison using the <=> operator.
		• MySQL does not support comparison between row expressions and NULL values. In GaussDB, the <, <=, =, >=, >, and <> operators can be used to compare row expressions with NULL values.
		• IS NULL or ISNULL operations on row expressions are not supported in MySQL. However, they are supported in GaussDB.
		• For operations by using operators that cannot be performed on row expressions, the error information in GaussDB is inconsistent with that in MySQL.
		GaussDB:
		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)
		m_db=# SELECT ROW(0,0) BETWEEN ROW(1,1) AND ROW(2,2);
		ERROR: un support type
		MySQL:
		mysql> SELECT (1,2) <=> row(2,3);
		+--------------------+
		| (1,2) <=> row(2,3) |
		+--------------------+
		| 0 |
		+--------------------+
		1 row in set (0.00 sec)
		
		mysql> SELECT (1,2) < NULL;
		ERROR 1241 (21000): Operand should contain 2 column(s)
		mysql> SELECT (1,2) <> NULL;
		ERROR 1241 (21000): Operand should contain 2 column(s)
		mysql> SELECT (1, 2) IS NULL;
		ERROR 1241 (21000): Operand should contain 1 column(s)

MySQL	GaussDB	Difference

		mysql> SELECT ISNULL((1, 2));
		ERROR 1241 (21000): Operand should contain 1 column(s)
		mysql> SELECT NULL BETWEEN NULL AND ROW(2,2);
		ERROR 1241 (21000): Operand should contain 1 column(s)

--	Supported, with differences.	MySQL indicates that an operand is negated twice and the result is equal to the original operand. GaussDB indicates a comment.
!!	Supported, with differences.	MySQL: The meaning of !! is the same as that of !, indicating NOT. GaussDB: ! indicates NOT. If there is a space between two exclamation marks (! !), it indicates NOT for twice. If there is no space between them (!!), it indicates factorial. **NOTE** • In GaussDB, when both factorial (!!) and NOT (!) are used, a space must be added between them. Otherwise, an error is reported. • In GaussDB, when multiple NOT operations are required, use a space between exclamation marks (! !).

MySQL	GaussDB	Difference
[NOT] REGEXP	Supported, with differences.	• GaussDB and MySQL support different metacharacters in regular expressions. For example, GaussDB allows \d to indicate digits, \w to indicate letters, digits, and underscores (_), and \s to indicate spaces. However, MySQL does not support these metacharacters and considers them as normal character strings. • In GaussDB, "\b" can match "\\b", but in MySQL, the matching will fail. • In GaussDB, a backslash (\) indicates an escape character. In MySQL, two backslashes (\\) are used. • MySQL does not support two operators to be used together. • If the input parameter of the pattern string is invalid with only the right parenthesis ()), GaussDB and MySQL 5.7 will report an error, but MySQL 8.0 will not. • In the rule of matching the de|abc sequence with de or abc, when there are empty values on the left and right of the pipe symbol (|), MySQL 5.7 will report an error, but GaussDB and MySQL 8.0 will not. • The regular expression of the tab character "\t" can match the character class [:blank:] in GaussDB and MySQL 8.0 but cannot in MySQL 5.7. • GaussDB supports non-greedy pattern matching. That is, the number of matching characters is as small as possible. A question mark (?) is added after some special characters, for example, ?? *? +? {n}? {n,}? {n,m}? MySQL 5.7 does not support non-greedy pattern matching, and the error message "Got error 'repetition-operator operand invalid' from regexp" is displayed. MySQL 8.0 already supports this function. • In the BINARY character set, the TEXT and BLOB types are converted to the BYTEA type. The REGEXP operator does not support the BYTEA type. Therefore, the two types cannot be matched.

MySQL	GaussDB	Difference
LIKE	Supported, with differences.	MySQL: The left operand of LIKE can only be an expression of a bitwise or arithmetic operation, or expression consisting of parentheses. The right operand of LIKE can only be an expression consisting of unary operators (excluding NOT) or parentheses. GaussDB: The left and right operands of LIKE can be any expression.
[NOT] BETWEEN AND	Supported, with differences.	MySQL: [NOT] BETWEEN AND is nested from right to left. The first and second operands of [NOT] BETWEEN AND can only be expressions of bitwise or arithmetic operations, or expressions consisting of parentheses. GaussDB: [NOT] BETWEEN AND is nested from left to right. The first and second operands of [NOT] BETWEEN AND can be any expression.
IN	Supported, with differences.	MySQL: The left operand of IN can only be an expression of a bitwise or arithmetic operation, or expression consisting of parentheses. GaussDB: The left operand of IN can be any expression.
!	Supported, with differences.	MySQL: The operand of ! can only be an expression consisting of unary operators (excluding NOT) or parentheses. GaussDB: The operand of ! can be any expression.
#	Not supported	MySQL supports the comment tag (#), but GaussDB does not.
BINARY	Supported, with differences.	Expressions (including some functions and operators) supported by GaussDB are different from those supported by MySQL. For GaussDB-specific expressions such as "~" and "IS DISTINCT FROM", due to the higher priority of the BINARY keyword, when BINARY expr is used, BINARY is combined with the left parameters of "~" and "IS DISTINCT FROM" first. As a result, an error is reported.
Negation (-)	Supported, with differences.	If the number of consecutive negation times exceeds 1, GaussDB identifies the negations as comments. As a result, it returns results different from MySQL.

MySQL	GaussDB	Difference

XOR, |, & , < , > , <=, >=, =, and !=	Supported, with differences.	The execution mechanism of MySQL is as follows: After the left operand is executed, the system checks whether the result is empty and then determines whether to execute the right operand.
		As for the execution mechanism of GaussDB, after the left and right operands are executed, the system checks whether the result is empty.
		If the result of the left operand is empty and an error is reported during the execution of the right operand, MySQL does not report an error but directly returns an error. GaussDB reports an error during the execution.
		Behavior in MySQL:
		mysql> SELECT version();
		+------------------+
		| version() |
		+------------------+
		| 5.7.44-debug-log |
		+------------------+
		1 row in set (0.00 sec)
		
		mysql> dROP TABLE IF EXISTS data_type_table;
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> CREATE TABLE data_type_table (
		-> MyBool BOOL,
		-> MyBinary BINARY(10),
		-> MyYear YEAR
		->);
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> INSERT INTO data_type_table VALUES (TRUE, 0x1234567890, '2021');
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT (MyBool % MyBinary) | (MyBool - MyYear) FROM data_type_table;
		+--+
		| (MyBool % MyBinary) | (MyBool - MyYear) |
		+--+
		| NULL |
		+--+
		1 row in set, 2 warnings (0.00 sec)
		Behavior in GaussDB:
		m_db=# DROP TABLE IF EXISTS data_type_table;
		DROP TABLE
		m_db=# CREATE TABLE data_type_table (
		m_db(# MyBool BOOL,
		m_db(# MyBinary BINARY(10),
		m_db(# MyYear YEAR
		m_db(#);
		CREATE TABLE
		m_db=# INSERT INTO data_type_table VALUES (TRUE, 0x1234567890, '2021');
		INSERT 0 1
		m_db=# SELECT (MyBool % MyBinary) | (MyBool - MyYear) FROM data_type_table;
		WARNING: Truncated incorrect double value: '4Vx '
		CONTEXT: referenced column: (MyBool % MyBinary) |

MySQL	GaussDB	Difference
		(MyBool - MyYear) WARNING: division by zero CONTEXT: referenced column: (MyBool % MyBinary) | (MyBool - MyYear) ERROR: Bigint is out of range. CONTEXT: referenced column: (MyBool % MyBinary) | (MyBool - MyYear)

Table 3-20 Differences in operator combinations

Example of Operator Combination	MySQL	GaussDB	Description
SELECT 1 LIKE 3 & 1;	Not supported	Supported.	The right operand of LIKE cannot be an expression consisting of bitwise operators.
SELECT 1 LIKE 1 +1;	Not supported	Supported.	The right operand of LIKE cannot be an expression consisting of arithmetic operators.
SELECT 1 LIKE NOT 0;	Not supported	Supported.	The right operand of LIKE can only be an expression consisting of unary operators (such as +, -, or ! but except NOT) or parentheses.
SELECT 1 BETWEEN 1 AND 2 BETWEEN 2 AND 3;	Right-to-left combination	Left-to-right combination	You are advised to add parentheses to specify the calculation priority to prevent result deviation caused by sequence differences.
SELECT 2 BETWEEN 1=1 AND 3;	Not supported	Supported.	The second operand of BETWEEN cannot be an expression consisting of comparison operators.
SELECT 0 LIKE 0 BETWEEN 1 AND 2;	Not supported	Supported.	The first operand of BETWEEN cannot be an expression consisting of pattern matching operators.
SELECT 1 IN (1) BETWEEN 0 AND 3;	Not supported	Supported.	The first operand of BETWEEN cannot be an expression consisting of IN operators.
SELECT 1 IN (1) IN (1);	Not supported	Supported.	The second left operand of the IN expression cannot be an expression consisting of INs.

Example of Operator Combination	MySQL	GaussDB	Description
SELECT ! NOT 1;	Not supported	Supported.	The operand of ! can only be an expression consisting of unary operators (such as +, -, or ! but except NOT) or parentheses.

Index Differences

- Currently, GaussDB supports only UB-tree and B-tree indexes.
- For fuzzy match (LIKE operator), the default index created can be used in MySQL, but cannot be used in GaussDB. You need to use the following syntax to specify **opclass** to, for example, **text_pattern_ops**, so that LIKE operators can be used as indexes:
  ```
  CREATE INDEX indexname ON tablename(col [opclass]);
  ```
- In the B-tree/UB-tree index scenario, the original logic of the native GaussDB is retained. That is, index scan supports comparison of types in the same operator family, but does not support other index types currently.
- When GaussDB JDBC is used to connect to the database, the YEAR type of GaussDB cannot use indexes in the PBE scenario that contains bind parameters.
- In the operation scenarios involving index column type and constant type, the conditions that indexes of a WHERE clause are supported in GaussDB is different from those in MySQL. For details, see **Table 3-21**. For example, GaussDB does not support indexes in the following statement:
  ```
  CREATE TABLE t(_int int);
  CREATE INDEX idx ON t(_int) USING BTREE;
  SELECT * FROM t WHERE _int > 2.0;
  ```

  NOTE

  In the operation scenarios involving index column type and constant type in the WHERE clause, you can use the cast function to explicitly convert the constant type to the column type for indexing.

  ```
  SELECT * FROM t WHERE _int > cast(2.0 AS signed);
  ```

Table 3-21 Differences in index support

Index Column Type	Constant Type	Supported by GaussDB	Supported by MySQL
Integer	Integer	Yes	Yes
Floating-point	Floating-point	Yes	Yes
Fixed-point	Fixed-point	Yes	Yes
String	String	Yes	Yes
Binary	Binary	Yes	Yes
Time with date	Time with date	Yes	Yes

Index Column Type	Constant Type	Supported by GaussDB	Supported by MySQL
TIME	TIME	Yes	Yes
Time with date	Type that can be converted to time type with date (for example, integers such as 20231130)	Yes	Yes
Time with date	TIME	Yes	Yes
TIME	Constants that can be converted to the TIME type (for example, integers such as 203008)	Yes	Yes
Floating-point	Integer	Yes	Yes
Floating-point	Fixed-point	Yes	Yes
Floating-point	String	Yes	Yes
Floating-point	Binary	Yes	Yes
Floating-point	Time with date	Yes	Yes
Floating-point	TIME	Yes	Yes
Fixed-point	Integer	Yes	Yes
String	Time with date	Yes	No
String	TIME	Yes	No
Binary	String	Yes	Yes
Binary	Time with date	Yes	No
Binary	TIME	Yes	No
Integer	Floating-point	No	Yes
Integer	Fixed-point	No	Yes
Integer	String	No	Yes
Integer	Binary	No	Yes
Integer	Time with date	No	Yes
Integer	TIME	No	Yes
Fixed-point	Floating-point	No	Yes

Index Column Type	Constant Type	Supported by GaussDB	Supported by MySQL
Fixed-point	String	No	Yes
Fixed-point	Binary	No	Yes
Fixed-point	Time with date	No	Yes
Fixed-point	TIME	No	Yes
String	Binary	No	Yes
Time with date	Integer (that cannot be converted to the time type with date)	No	Yes
Time with date	Floating-point (that cannot be converted to the time type with date)	No	Yes
Time with date	Fixed-point (that cannot be converted to the time type with date)	No	Yes
TIME	Integer (that cannot be converted to the TIME type)	No	Yes
TIME	Character string (that cannot be converted to the TIME type)	No	Yes
TIME	Binary (that cannot be converted to the TIME type)	No	Yes
TIME	Time with date	No	Yes
YEAR	YEAR	Yes	Yes
YEAR	Constants that can be converted to the YEAR type (for example, integers such as 2034)	Yes	Yes

Index Column Type	Constant Type	Supported by GaussDB	Supported by MySQL
BIT	BIT	No	Yes

Table 3-22 Whether index use is supported

Index Column Type	Constant Type	Use Index or Not in GaussDB	Use Index or Not in MySQL
String	Integer	No	No
String	Floating-point	No	No
String	Fixed-point	No	No
Binary	Integer	No	No
Binary	Floating-point	No	No
Binary	Fixed-point	No	No
Time with date	Character string (that cannot be converted to the time type with date)	No	No
Time with date	Binary (that cannot be converted to the time type with date)	No	No
TIME	Floating-point (that cannot be converted to the TIME type)	No	No
TIME	Fixed-point (that cannot be converted to the TIME type)	No	No
BIT	String	No	No

3.2.4 Character Sets

GaussDB allows you to specify the following character sets for databases, schemas, tables, or columns.

Table 3-23 Character sets

MySQL	GaussDB
utf8mb4	Supported.
utf8	Supported.
gbk	Supported.
gb18030	Supported.
binary	Supported.

NOTE

- **utf8** and **utf8mb4** refer to the same character set in GaussDB. The maximum code length is 4 bytes. If the current character set is **utf8** and the collation is set to **utf8mb4_bin**, **utf8mb4_general_ci**, **utf8mb4_unicode_ci**, or **utf8mb4_0900_ai_ci** (for example, by running **SELECT _utf8'a' collate utf8mb4_bin**), MySQL reports an error but GaussDB does not. The difference also exists when the character set is **utf8mb4** and the collation is set to **utf8_bin**, **utf8_general_ci**, or **utf8_unicode_ci**.
- The lexical syntax is parsed based on the byte stream. 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.

3.2.5 Collation Rules

GaussDB allows you to specify the following collation rules for schemas, tables, or columns.

NOTE

Differences in collation rules:

- Currently, collation rules can only be specified for the character string type and some binary types. You can check the typcollation attribute of a type in the pg_type system catalog. If it is not **0**, the type supports the collation. The collation can be specified for all types in MySQL. However, collation rules are meaningless except those for character strings and binary types.
- The current collation rules (except binary) can be specified only when the corresponding character set is the same as the database-level character set. In GaussDB, the character set must be the same as the database character set, and multiple character sets cannot be used together in a table.
- The default collation of the utf8mb4 character set is utf8mb4_general_ci, which is the same as that in MySQL 5.7.

Table 3-24 Collation rules

MySQL	GaussDB
utf8mb4_general_ci	Supported.
utf8mb4_unicode_ci	Supported.

MySQL	GaussDB
utf8mb4_bin	Supported.
gbk_chinese_ci	Supported.
gbk_bin	Supported.
gb18030_chinese_ci	Supported.
gb18030_bin	Supported.
binary	Supported.
utf8mb4_0900_ai_ci	Supported.
utf8_general_ci	Supported.
utf8_bin	Supported.
utf8_unicode_ci	Supported.

3.2.6 Transactions

GaussDB is compatible with MySQL transactions, but there are some differences. This section describes transaction-related differences in GaussDB M-compatible databases.

Default Transaction Isolation Levels

The default isolation level of an M-compatible database is READ COMMITTED, and that of MySQL is REPEATABLE READ.

```
-- View the current transaction isolation level.
m_db=# SHOW transaction_isolation;
```

Sub-transactions

In an M-compatible database, SAVEPOINT is used to create a savepoint (sub-transaction) in the current transaction, and ROLLBACK TO SAVEPOINT is used to roll back to a savepoint (sub-transaction). After the sub-transaction is rolled back, the parent transaction can continue to run, the rollback of a sub-transaction does not affect the transaction status of the parent transaction.

No savepoint (sub-transaction) can be created in MySQL.

Nested Transactions

A nested transaction refers to a new transaction started in a transaction block.

In an M-compatible database, if a new transaction is started in a normal transaction block, a warning is displayed indicating that an ongoing transaction exists and the start command is ignored. If a new transaction is started in an abnormal transaction block, an error is reported. The transaction can be executed

only after **ROLLBACK** or **COMMIT** is executed. If **ROLLBACK** or **COMMIT** is executed, the previous statement is rolled back.

In MySQL, before a new transaction is started in a normal transaction block, the previous one must be committed. If a new transaction is started in an abnormal transaction block, the error is ignored and the previous correct statement is committed.

```
-- In an M-compatible database, if a new transaction is started in a normal transaction block, a warning is generated and the transaction is ignored.
m_db=# DROP TABLE IF EXISTS test_t;
m_db=# CREATE TABLE test_t(a int, b int);
m_db=# BEGIN;
m_db=# INSERT INTO test_t values(1, 2);
m_db=# BEGIN; -- The warning "There is already a transaction in progress" is displayed.
m_db=# SELECT * FROM test_t ORDER BY 1;
m_db=# COMMIT;

-- In an M-compatible database, if a new transaction is started in an abnormal transaction block, an error is reported. The transaction can be executed only after ROLLBACK or COMMIT is executed.
m_db=# BEGIN;
m_db=# ERROR sql; -- Error statement.
m_db=# BEGIN; -- An error is reported.
m_db=# COMMIT; -- It can be executed only after ROLLBACK/COMMIT is executed.
```

Statements Committed Implicitly

An M-compatible database uses GaussDB for storage and inherits the GaussDB transaction mechanism. If a DDL or DCL statement is executed in a transaction, the transaction is not automatically committed.

In MySQL, if DDL, DCL, management-related, or lock-related statements are executed, the transaction is automatically committed.

```
-- In M-compatible database, table creation and GUC parameter setting support rollback.
m_db=# DROP TABLE IF EXISTS test_table_rollback;
m_db=# BEGIN;
m_db=# CREATE TABLE test_table_rollback(a int, b int);
m_db=# \d test_table_rollback;
m_db=# ROLLBACK;
m_db=# \d test_table_rollback; -- This table does not exist.
```

Differences in SET TRANSACTION

In an M-compatible database, if SET TRANSACTION is used to set the isolation level or transaction access mode for multiple times, only the last setting takes effect. Transaction features can be separated by spaces or commas (,).

In MySQL, SET TRANSACTION cannot be used to set the isolation level or transaction access mode for multiple times. Transaction features can only be separated by commas (,).

Table 3-25 Differences in SET TRANSACTION

Syntax	Description	Difference
SET TRANSACTION	Sets transactions.	In M-compatible mode, if the **m_format_dev_version** parameter is not set to **'s2'**, SET TRANSACTION takes effect at the session level, with the same functionality as SET SESSION TRANSACTION. If the **m_format_dev_version** parameter is set to **'s2'**, SET TRANSACTION sets the next transaction feature. In MySQL, SET TRANSACTION takes effect in the next transaction.
SET SESSION TRANSACTION	Sets session-level transactions.	-
SET GLOBAL TRANSACTION	Sets global session-level transactions. This feature applies to subsequent sessions and has no impact on the current session.	In an M-compatible database, GLOBAL takes effect in global session-level transactions and is applicable only to the current database instance. In MySQL, this feature takes effect in all databases.

```
-- SET TRANSACTION takes effect in session-level transactions.
m_db=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
m_db=# SHOW transaction_isolation;
m_db=# SHOW transaction_read_only;
-- In an M-compatible database, if the isolation level or transaction access mode is set for multiple times, only the last setting takes effect.
m_db=# SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED, ISOLATION LEVEL REPEATABLE READ, READ WRITE, READ ONLY;
m_db=# SHOW transaction_isolation; -- repeatable read
m_db=# SHOW transaction_read_only; -- on
```

Differences in START TRANSACTION

In an M-compatible database, when START TRANSACTION is used to start a transaction, the isolation level can be set. If the isolation level or transaction access mode is set for multiple times, only the last setting takes effect. In the current version, consistency snapshot cannot be enabled immediately. Transaction features can be separated by spaces or commas (,).

In MySQL, if START TRANSACTION is used to start a transaction, the isolation level cannot be set and the transaction access mode cannot be set for multiple times. Transaction features can only be separated by commas (,).

```
-- Start a transaction and set the isolation level.
m_db=# START TRANSACTION ISOLATION LEVEL READ COMMITTED;
```



```
m_db=# COMMIT;
-- Set the access mode for multiple times.
m_db=# START TRANSACTION READ ONLY, READ WRITE;
m_db=# COMMIT;
```

Transaction-related GUC Parameters

Table 3-26 Differences in transaction-related GUC parameters

GUC Parameter	Description	Difference
autocommit	Sets the automatic transaction commit mode.	-
transaction_isolation	Sets the isolation level of the current transaction in an M-compatible database. Sets the isolation level of a session-level transaction in MySQL.	• In GaussDB, you can only change the isolation level of the current transaction by running the **SET transaction_isolation =** *value* command. To change the session-level isolation level, use **default_transaction_isolation**. In MySQL, you can run the **SET** command to change the isolation level of a session-level transaction. • The supported range is different. MySQL supports the following isolation levels, which are case-insensitive but space-sensitive: – READ-COMMITTED – READ-UNCOMMITTED – REPEATABLE READ – SERIALIZABLE GaussDB supports the following isolation levels, which are case-sensitive and space-sensitive: – read committed – read uncommitted – repeatable read – serializable – default (The level is set to be the same as the default isolation level in the session.) – If **m_format_dev_version** is set to **'s2'**, the isolation levels of MySQL can be set. • In GaussDB, the value of **transaction_isolation** of a new transaction is initialized to the value of **default_transaction_isolation**.

GUC Parameter	Description	Difference
tx_isolation	Sets the transaction isolation level. **tx_isolation** and **transaction_isolation** are synonyms.	This parameter does not support query or modification in an M-compatible database. You are advised to use **transaction_isolation** for query.
default_transaction_isolation	Sets the transaction isolation level.	In an M-compatible database, the **SET** command is used to change the transaction isolation level for a session. MySQL does not support this system parameter.
transaction_read_only	Sets the access mode of a transaction.	• In an M-compatible database, only the access mode of the current transaction can be changed by using the **SET** command. If you want to change the access mode of a session-level transaction, you can use **default_transaction_read_only**. In MySQL, you can run the **SET** command to change the isolation level of a session-level transaction. • In GaussDB, the value of **transaction_read_only** of a new transaction is initialized to the value of **default_transaction_read_only**.
tx_read_only	Sets the access mode of a transaction. **tx_read_only** and **transaction_read_only** are synonyms.	This parameter does not support query or modification in an M-compatible database. You are advised to use **transaction_read_only** for query.
default_transaction_read_only	Sets the access mode of a transaction.	In an M-compatible database, the **SET** command is used to change the access mode of a session-level transaction. MySQL does not support this system parameter.

3.2.7 SQL

3.2.7.1 Keywords

The constraint differences are as follows:

- If a keyword is a reserved one in M-compatible mode but non-reserved in MySQL, it cannot be a table name, column name, column alias, AS column alias, AS table alias, table alias, function name, or variable name in M-compatible mode, but can be any of these names or aliases in MySQL.
- If a keyword is a non-reserved one in M-compatible mode but reserved in MySQL, it can be a table name, column name, column alias, AS column alias, AS table alias, table alias, function name, or variable name in M-compatible mode, but cannot be any of these names or aliases in MySQL.
- If a keyword is a reserved one (function or type) both in M-compatible mode and MySQL, it can be a column alias, AS column alias, function name, or variable name in M-compatible mode, but cannot be any of these names or aliases in MySQL.
- If a keyword is a reserved one (function or type) in M-compatible mode but non-reserved in MySQL, it cannot be a table name, column name, AS table alias, or table alias in M-compatible mode, but can be one of these names or aliases in MySQL.
- If a keyword is a non-reserved one (excluding function and type) in M-compatible mode but reserved in MySQL, it can be a table name, column name, column alias, AS column alias, AS table alias, table alias, function name, or variable name in M-compatible mode, but cannot be any of these names or aliases in MySQL.
- If a keyword is a non-reserved one (excluding function and type) both in M-compatible mode and MySQL, it cannot be a function name in M-compatible mode, but can be a function name in MySQL.

  NOTE

  Among non-reserved keywords, reserved keywords (functions or types), and non-reserved keywords (not functions or types) in M-compatible mode, the following keywords cannot be used as column aliases:

  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.

  SIGNED and WITHOUT can be used as column aliases in MySQL.

3.2.7.2 Identifiers

Differences in identifiers in M-compatible mode are as follows:

- In GaussDB, unquoted identifiers cannot start with a dollar sign ($). In MySQL unquoted identifiers can start with a dollar sign ($).
- GaussDB unquoted identifiers support case-sensitive database objects.
- GaussDB identifiers support extended characters from U+0080 to U+00FF. MySQL identifiers support extended characters from U+0080 to U+FFFF.
- As for unquoted identifier, a table that starts with a digit and ends with an e or E as the identifier cannot be created in GaussDB. For example:


```
-- GaussDB reports an error indicating that this operation is not supported. MySQL supports this
operation.
m_db=# CREATE TABLE 23e(c1 int);
ERROR:  syntax error at or near "23"
LINE 1: CREATE TABLE 23e(c1 int);
                     ^
m_db=# CREATE TABLE t1(23E int);
ERROR:  syntax error at or near "23"
LINE 1: CREATE TABLE t1(23E int);
                        ^
```

- As for quoted identifiers, tables whose column names contain only digits or scientific computing cannot be directly used in GaussDB. You need to use them in quotes. This rule also applies to the dot operator (.) scenarios. For example:

```
-- Create a table whose column names contain only numbers or scientific computing.
m_db=# CREATE TABLE t1(`123` int, `1e3` int, `1e` int);
CREATE TABLE

-- Insert data into the table.
m_db=# INSERT INTO t1 VALUES(7, 8, 9);
INSERT 0 1

-- The result is not as expected, but is the same as that in MySQL.
m_db=# SELECT 123 FROM t1;
 ?column?
----------
      123
(1 row)

-- The result is not as expected, but is the same as that in MySQL.
m_db=# SELECT 1e3 FROM t1;
 ?column?
----------
     1000
(1 row)

-- The result is not as expected and is not the same as that in MySQL.
m_db=# SELECT 1e FROM t1;
 e
---
 1
(1 row)

-- The correct way to use is as follows:
m_db=# SELECT `123` FROM t1;
 123
-----
   7
(1 row)

m_db=# SELECT `1e3` FROM t1;
 1e3
-----
   8
(1 row)

m_db=# SELECT `1e` FROM t1;
 1e
----
  9
(1 row)

-- Dot operator scenarios are not supported by GaussDB but supported by MySQL.
m_db=# SELECT t1.123 FROM t1;
ERROR:  syntax error at or near ".123"
LINE 1: SELECT t1.123 FROM t1;
                 ^
m_db=# SELECT t1.1e3 FROM t1;
```



```
ERROR:  syntax error at or near "1e3"
LINE 1: SELECT t1.1e3 FROM t1;
                  ^
m_db=# SELECT t1.1e FROM t1;
ERROR:  syntax error at or near "1"
LINE 1: SELECT t1.1e FROM t1;
                  ^
-- The correct way to use in dot operator scenarios is as follows:
m_db=# SELECT t1.`123` FROM t1;
 123
-----
   7
(1 row)

m_db=# SELECT t1.`1e3` FROM t1;
 1e3
-----
   8
(1 row)

m_db=# SELECT t1.`1e` FROM t1;
 1e
----
  9
(1 row)

m_db=# DROP TABLE t1;
DROP TABLE
```

- In GaussDB, the partition name is case-sensitive when it is enclosed in double quotation marks (**SQL_MODE** must be set to **ANSI_QUOTES**) or backquotes, but in MySQL the partition name is case-insensitive.
- The maximum length of a MySQL identifier is 64 characters, while that of a GaussDB identifier is 63 bytes. If the length of an identifier exceeds the limit, MySQL reports an error, while GaussDB truncates the identifier and generates an alarm.

3.2.7.3 DDL

Table 3-27 DDL syntax compatibility

Description	Syntax	Difference
Create primary keys, UNIQUE indexes, and foreign keys during table creation and modification.	ALTER TABLE and CREATE TABLE	• In GaussDB, when the table joined with the constraint is Ustore and USING BTREE is specified in the SQL statement, the underlying index is created as UB-tree. • GaussDB: Foreign keys can be used as partition keys. • The index name, constraint name, and key name are unique in a schema in GaussDB and unique in a table in MySQL.

Description	Syntax	Difference
Auto-increment columns	ALTER TABLE and CREATE TABLE	• It is recommended that an auto-increment column in GaussDB be the first column of an index. Otherwise, an alarm is generated during table creation. The auto-increment column in MySQL must be the first column of the index. Otherwise, an error is reported during table creation. In GaussDB, an error occurs when some operations (such as ALTER TABLE EXCHANGE PARTITION) are performed on a table that contains auto-increment columns. • In GaussDB, **value** in the AUTO_INCREMENT = value syntax must be a positive number less than 2^127. It can be **0** in MySQL but cannot be **0** in GaussDB. • In GaussDB, an error occurs if the auto-increment continues after an auto-increment value reaches the maximum value of a column data type. In MySQL, errors or warnings may be generated during auto-increment, and sometimes auto-increment continues until the maximum value is reached. • GaussDB does not support the *innodb_autoinc_lock_mode* system variable, but when its GUC parameter **auto_increment_cache** is set to **0**, the behavior of inserting auto-increment columns in batches is

Description	Syntax	Difference
		similar to that when the MySQL system variable *innodb_autoinc_lock_mode* is set to **1**. • In GaussDB, when 0s, NULLs, and definite values are imported or batch inserted into auto-increment columns, the auto-increment values inserted after an error occurs in GaussDB may not be the same as those in MySQL. You can use the GUC parameter **auto_increment_cache** to control the number of reserved auto-increment values. • In GaussDB, when auto-increment is triggered by parallel import or insertion of auto-increment columns, the cache value reserved for each parallel thread is used only in the thread. If the cache value is not used up, the values of auto-increment columns in the table are discontinuous. The auto-increment value generated by parallel insertion cannot be guaranteed to be the same as that generated in MySQL. • In GaussDB, when auto-increment columns are batch inserted into a local temporary table, no auto-increment value is reserved. In normal scenarios, auto-increment values are not discontinuous. In MySQL, the auto-increment result of an auto-increment column in a temporary

Description	Syntax	Difference
		table is the same as that in an ordinary table. • The SERIAL data type of GaussDB is an original auto-increment column, which is different from the **AUTO_INCREMENT** column. The SERIAL data type of MySQL is the **AUTO_INCREMENT** column. • GaussDB does not allow the value of **auto_increment_offset** to be greater than that of **auto_increment_increment**. Otherwise, an error occurs. MySQL allows it and states that **auto_increment_offset** will be ignored. • If a table has a primary key or index, the sequence in which the **ALTER TABLE** command rewrites table data may be different from that in MySQL. GaussDB rewrites table data based on the table data storage sequence, while MySQL rewrites table data based on the primary key or index sequence. As a result, the auto-increment sequence may be different. • When the **ALTER TABLE** command in GaussDB is used to add or modify auto-increment columns, the number of auto-increment values reserved for the first time is the number of rows in the table statistics. The number of rows in the statistics may not be the same as that in MySQL.

Description	Syntax	Difference
		• The return value of the last_insert_id function in GaussDB is a 128-bit integer. • When GaussDB performs auto-increment in a trigger or user-defined function, the return value of last_insert_id is updated. MySQL does not update it. • If the values of the GUC parameters **auto_increment_offset** and **auto_increment_increment** in GaussDB are out of range, an error occurs. MySQL automatically changes the value to a boundary value.
Prefix indexes	CREATE INDEX, ALTER TABLE, and CREATE TABLE	• In GaussDB, the prefix length cannot exceed 2676. 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, an error may be reported when the index line length exceeds the threshold. • In GaussDB, the primary key index does not support prefix keys. The prefix length cannot be specified when a primary key is created or added.

Description	Syntax	Difference
Specify character sets and collation rules.	ALTER SCHEMA, ALTER TABLE, CREATE SCHEMA, and CREATE TABLE	• When you specify a database-level character set, except binary character sets, the character set of a new database or schema cannot be different from that specified by **server_encoding** of the database. • When you specify a table-level or column-level character set and collation, MySQL allows you to specify a character set and collation that are different from the database-level character set and collation. In GaussDB, the table-level and column-level character sets and collations support only the binary character sets and collations or can be the same as the database-level character sets and collations.
Add columns before the first column of a table or after a specified column during table modification.	ALTER TABLE	-
Alter the column name/definition.	ALTER TABLE	Currently, the DROP INDEX, DROP KEY, or ORDER BY is not supported.

Description	Syntax	Difference
Create a partitioned table.	CREATE TABLE PARTITION	• MySQL supports expressions but does not support multiple partition keys in the following scenarios: – The LIST/RANGE partitioning policy is used and the COLUMNS keyword is not specified. – The hash partitioning policy is used. • MySQL does not support expressions but supports multiple partition keys in the following scenarios: – The LIST/RANGE partitioning policy is used and the COLUMNS keyword is specified. – The KEY partitioning policy is used. • In GaussDB, expressions cannot be used as partition keys, and partitions cannot be specified. • GaussDB supports multiple partition keys only when the LIST or RANGE partitioning policy is used. • When LIST partitioning is used in GaussDB, ensure that **list_value** is set to a valid value of the corresponding partition key type. Otherwise, the creation fails. However, the creation can be successful even if an invalid value is used. • In GaussDB partitioned tables, generated columns cannot be used as partition keys.

Description	Syntax	Difference
Specify table-level and column-level comments during table creation and modification.	CREATE TABLE and ALTER TABLE	-
Specify index-level comments during index creation.	CREATE INDEX	-

Description	Syntax	Difference
Exchange the partition data of an ordinary table and a partitioned table.	ALTER TABLE PARTITION	Differences in ALTER TABLE EXCHANGE PARTITION: ● After **ALTER TABLE EXCHANGE PARTITION** is executed, the auto-increment columns are reset in MySQL, but in GaussDB, they are not reset and continue the auto-increment based on their old values. ● If MySQL tables or partitions use tablespaces, data in partitions and ordinary tables cannot be exchanged. If GaussDB tables or partitions use different tablespaces, data in partitions and ordinary tables can still be exchanged. ● MySQL does not verify the default values of columns. Therefore, data in partitions and ordinary tables can be exchanged even if the default values are different. GaussDB verifies the default values. If they are different, data in partitions and ordinary tables cannot be exchanged. ● After the DROP COLUMN operation is performed on a partitioned table or an ordinary table in MySQL, if the table structure is still consistent, data can be exchanged between partitions and ordinary tables. In GaussDB, data can be exchanged between partitions and ordinary tables only when the deleted columns of ordinary tables and partitioned tables are strictly aligned.

Description	Syntax	Difference
		• MySQL and GaussDB use different hash algorithms. Therefore, data stored in the same hash partition may be inconsistent. As a result, the exchanged data may also be inconsistent. • MySQL partitioned tables do not support foreign keys. If an ordinary table contains foreign keys or other tables reference foreign keys of an ordinary table, data in partitions and ordinary tables cannot be exchanged. GaussDB partitioned tables support foreign keys. If the FOREIGN KEY constraints of two tables are the same, data in partitions and ordinary tables can be exchanged. If a GaussDB partitioned table does not contain foreign keys, an ordinary table is referenced by other tables, and the partitioned table is the same as the ordinary table, data in the partitioned table can be exchanged with that in the ordinary table.
Modify the partition key information of a partitioned table.	ALTER TABLE	MySQL allows you to modify the partition key information of a partitioned table, but GaussDB does not.

Description	Syntax	Difference
CREATE TABLE... LIKE syntax	CREATE TABLE ... LIKE	• In versions earlier than MySQL 8.0.16, CHECK constraints are parsed but their functions are ignored. In this case, CHECK constraints are not replicated. GaussDB supports replication of CHECK constraints. • When a table is created, all PRIMARY KEY constraint names in MySQL are fixed to **PRIMARY KEY**. GaussDB does not support replication of PRIMARY KEY constraint names. • When a table is created, MySQL supports replication of UNIQUE KEY constraint names, but GaussDB does not. • When a table is created, MySQL versions earlier than 8.0.16 do not have CHECK constraint information, but GaussDB supports replication of CHECK constraint names. • When a table is created, MySQL supports replication of index names, but GaussDB does not. • When a table is created across sql_mode, MySQL is controlled by the loose mode and strict mode. The strict mode may become invalid in GaussDB. For example, if the source table has the default value **"0000-00-00"**, a table with the default value **"0000-00-00"** can be created in "no_zero_date" strict mode in GaussDB, which means that the

Description	Syntax	Difference
		strict mode is invalid. In MySQL, the table creation will fail because it is controlled by the strict mode.
Truncate a partition.	ALTER TABLE [IF EXISTS] table_name truncate_clause;	For truncate_clause, the supported subitems are different: • M-compatible mode: TRUNCATE PARTITION { { ALL | partition_name [, ...] } | FOR (partition_value [, ...]) } [UPDATE GLOBAL INDEX] • MySQL: TRUNCATE PARTITION {partition_names | ALL}
Index name of a primary key	CREATE TABLE table_name (col_definitine ,PRIMARY KEY [index_name] [USING method] ({ column_name | (expression) }[ASC | DESC] } [, ...]) index_parameters [USING method| COMMENT 'string'])	The index name created after being specified by a primary key in GaussDB is the index name specified by a user. In MySQL, the index name is **PRIMARY**.
Delete dependent objects.	DROP drop_type name CASCADE;	In GaussDB, **CASCADE** needs to be added to delete dependent objects. In MySQL, **CASCADE** is not required.
The NOT NULL constraint does not allow NULL values to be inserted.	CREATE TABLE t1(id int NOT NULL DEFAULT 8); INSERT INTO t1 VALUES(NULL); INSERT INTO t1 VALUES(1), (NULL),(2);	In MySQL loose mode, NULL is converted and data is successfully inserted. In MySQL strict mode, NULL values cannot be inserted. GaussDB does not support this feature. NULL values cannot be inserted in loose or strict mode.

Description	Syntax	Difference
The CHECK constraint takes effect.	CREATE TABLE	• CREATE TABLE that contains the CHECK constraint takes effect in MySQL 8.0. MySQL 5.7 parses the syntax but the syntax does not take effect. GaussDB synchronizes this function of MySQL 8.0, and the GaussDB CHECK constraint can reference other columns, but MySQL cannot. • A maximum of 32767 CHECK constraints can be added to a table in GaussDB.
The **algorithm** and **lock** options of an index do not take effect.	CREATE INDEX ... DROP INDEX ...	Currently, the index options **algorithm_option** and **lock_option** in the CREATE/DROP INDEX statement in M-compatible mode are supported only in syntax. No error is reported during creation, but they do not take effect.
The storage of hash partitions and subpartitions in CREATE TABLE in GaussDB is different from that in MySQL.	CREATE TABLE	In GaussDB, the hash functions used by hash partitioned tables and subpartitioned tables in the CREATE TABLE statement are different from those used in MySQL. Therefore, the storage of hash partitioned tables and subpartitioned tables is different from that in MySQL.

Description	Syntax	Difference
Partitioned table indexes	CREATE INDEX	• GaussDB partitioned table indexes are classified into local and global indexes. A local index is bound to a specific partition, and a global index corresponds to the entire partitioned table. • For details about how to create local and global indexes and the default rules, see "SQL Syntax > SQL Statement > C > CREATE INDEX " in *Developer Guide*. For example, if a unique index is created on a non-partition key, a global index is created by default. • MySQL does not have global indexes. In GaussDB, if the partitioned table index is a global index, the global index is not updated by default when operations such as DROP, TRUNCATE, and EXCHANGE are performed on table partitions. As a result, the global index becomes invalid and cannot be selected in subsequent statements. To avoid this problem, you are advised to explicitly specify the UPDATE GLOBAL INDEX clause at the end of the partition syntax or set the global GUC parameter **enable_gpi_auto_update** to **true** (recommended) so that global indexes can be automatically updated during partition operations.

Description	Syntax	Difference
If the table is partitioned by key in the CREATE/ALTER TABLE statement, algorithms cannot be specified. Input parameters of some partition definition do not support expressions.	CREATE TABLE and ALTER TABLE	In GaussDB, if the table is partitioned by key in the CREATE/ALTER TABLE statement, algorithms cannot be specified. The syntaxes that do not support expressions as input parameters are as follows: • PARTITION BY HASH() • PARTITION BY KEY() • VALUES LESS THAN()
Partitioned tables do not support LINEAR/KEY hash.	CREATE TABLE ... PARTITION ...	GaussDB: Partitioned tables do not support LINEAR/KEY hash.
The CHECK and AUTO_INCREMENT syntaxes cannot be used in the same column.	CREATE TABLE	The column using CHECK does not take effect in MySQL 5.7. When both CHECK and AUTO_INCREMENT are used on the same column, only AUTO_INCREMENT takes effect. However, GaussDB reports an error.
Delete dependent tables.	DROP TABLE	In GaussDB, CASCADE must be added to delete dependent tables. In MySQL, CASCADE is not required.

Description	Syntax	Difference
Options related to table definition.	CREATE TABLE ... and ALTER TABLE ...	• GaussDB does not support the following options: **AVG_ROW_LENGTH**, **CHECKSUM**, **COMPRESSION**, **CONNECTION**, **DATA DIRECTORY**, **INDEX DIRECTORY**, **DELAY_KEY_WRITE**, **ENCRYPTION**, **INSERT_METHOD**, **KEY_BLOCK_SIZE**, **MAX_ROWS**, **MIN_ROWS**, **PACK_KEYS**, **PASSWORD**, **STATS_AUTO_RECALC**, **STATS_PERSISTENT**, and **STATS_SAMPLE_PAGES**. • The following options do not report errors in GaussDB and do not take effect: **ENGINE** and **ROW_FORMAT**.
Encrypt the CMKs of CEKs in round robin (RR) mode and encrypt the plaintext of CEKs.	ALTER COLUMN ENCRYPTION KEY	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
The encrypted equality query feature adopts a multi-level encryption model. The master key encrypts the column key, and the column key encrypts data. This syntax is used to create a master key object.	CREATE CLIENT MASTER KEY	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Create a CEK that can be used to encrypt a specified column in a table.	CREATE COLUMN ENCRYPTION KEY	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.

Description	Syntax	Difference
Send keys to the server for caching. This function is used only when the memory decryption emergency channel is enabled. This is a fully-encrypted function.	\send_token	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Send keys to the server for caching. This function is used only when the memory decryption emergency channel is enabled. This is a fully-encrypted function.	\st	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Destroy the keys cached on the server. This function is used only when the memory decryption emergency channel is enabled. This is a fully-encrypted function.	\clear_token	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Destroy the keys cached on the server. This function is used only when the memory decryption emergency channel is enabled. This is a fully-encrypted function.	\ct	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Set the parameters for accessing the external key manager in the fully-encrypted database features.	\key_info KEY_INFO	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
Enable third-party dynamic libraries and set related parameters. This is a fully-encrypted function.	\crypto_module_info MODULE_INFO	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.

Description	Syntax	Difference
Enable third-party dynamic libraries and set related parameters. This is a fully-encrypted function.	\cmi MODULE_INFO	The M-compatible mode does not support the full encryption. Therefore, this syntax is not supported.
The GENERATED ALWAYS AS statement cannot reference columns generated by GENERATED ALWAYS AS.	Generated Always AS	In GaussDB, the GENERATED ALWAYS AS statement cannot reference columns generated by GENERATED ALWAYS AS, but it can in MySQL.
Alter table names.	ALTER TABLE tbl_name RENAME [TO | AS | =] new_tbl_name;	• The ALTER RENAME syntax in GaussDB supports only the function of changing the table name and cannot be coupled with other function operations. • In GaussDB, only the old table name column supports the schema.table_name format, and the new and old table names belong to the same schema. • GaussDB does not support renaming of old and new tables across schemas. However, if you have the permission, you can modify the names of tables in other schemas in the current schema.
Disable the GUC parameter **enable_expr_fusion**.	SET enable_expr_fusion= ON	In M-compatible mode, the GUC parameter **enable_expr_fusion** cannot be enabled.

Description	Syntax	Difference
CREATE VIEW AS SELECT syntax	CREATE VIEW table_name AS query;	• For the following types, the query using the CREATE VIEW view_name AS query syntax cannot contain calculation operations (such as function call and calculation using operators): – 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 • In the simple query scenario, an error message is displayed for the

Description	Syntax	Difference

		preceding calculation operations in M-compatible mode. For example:
		m_db=# CREATE TABLE TEST (salary int(10));
		CREATE TABLE
		
		m_db=# INSERT INTO TEST VALUES(8000);
		INSERT 0 1
		
		m_db=# CREATE VIEW view1 AS SELECT salary/10 as te FROM TEST;
		ERROR: Unsupported type numeric used with expression in CREATE VIEW statement.
		
		m_db=# CREATE VIEW view2 AS SELECT sec_to_time(salary) as te FROM TEST;
		ERROR: Unsupported type time used with expression in CREATE VIEW statement.
		• In non-simple query scenarios such as composite query and subquery, the calculation operations of the preceding types in M-compatible mode are different from those in MySQL. In M-compatible mode, the data type column precision attribute of the created table is not retained.

Range of index names that can be duplicated	CREATE TABLE, CREATE INDEX	In MySQL, an index name is unique in a table. Different tables can have the same index name. In M-compatible mode, the index name must be unique in the same schema. In M-compatible mode, the same rules apply to constraints and keys that automatically create indexes.

Description	Syntax	Difference
View dependency differences	CREATE VIEW and ALTER TABLE	In MySQL, view storage records only the table name, column name, and database name of the target table, but does not record the unique identifier of the target table. GaussDB parses the SQL statement used for creating a view and stores the unique identifier of the target table. Therefore, the differences are as follows: • In MySQL, you can modify the data type of a column on which a view depends because the view is unaware of the modification of the target table. In GaussDB, such modification is forbidden and the attempt will fail. • In MySQL, you can rename a column on which a view depends because the view is unaware of the modification of the target table, but the view cannot be queried after the operation. In GaussDB, each column precisely stores the unique identifier of the corresponding table and column. Therefore, the column name in the table can be modified successfully without changing the column name in the view. In addition, the view can be queried after the operation.

Description	Syntax	Difference
Foreign key differences	CREATE TABLE	• In GaussDB, FOREIGN KEY constraints are insensitive to types. If the data types of the columns in the main and child tables are implicitly converted, foreign keys can be created. In MySQL, FOREIGN KEY constraints are sensitive to types. If the column types of the two tables are different, foreign keys cannot be created. • MySQL does not allow you to modify the data type or name of a table column where the foreign key of the column is located by running **MODIFY COLUMN** or **CHANGE COLUMN**, but GaussDB supports such operation.
Differences in index ascending and descending orders	CREATE INDEX	In MySQL 5.7, **ASC | DESC** is parsed but ignored, and the default behavior is **ASC**. In MySQL 8.0 and GaussDB, **ASC | DESC** is parsed and takes effect.

Description	Syntax	Difference

Setting default values of columns	CREATE TABLE and ALTER TABLE	• For MySQL 5.7, only the default value without parentheses is supported. MySQL 8.0 and GaussDB support default values in parentheses.
		-- GaussDB
		m_db=# DROP TABLE IF EXISTS t1, t2;
		DROP TABLE
		m_db=# CREATE TABLE t1(a DATETIME DEFAULT NOW());
		CREATE TABLE
		m_db=# CREATE TABLE t2(a DATETIME DEFAULT (NOW()));
		CREATE TABLE
		
		-- MySQL 5.7
		mysql> DROP TABLE IF EXISTS t1, t2;
		Query OK, 0 rows affected (0.04 sec)
		
		mysql> CREATE TABLE t1(a DATETIME DEFAULT NOW());
		Query OK, 0 rows affected (0.04 sec)
		
		mysql> CREATE TABLE t2(a DATETIME DEFAULT (NOW()));
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(NOW()))' at line 1
		
		-- MySQL 8.0
		mysql> DROP TABLE IF EXISTS t1, t2;
		Query OK, 0 rows affected (0.17 sec)
		
		mysql> CREATE TABLE t1(a DATETIME DEFAULT NOW());
		Query OK, 0 rows affected (0.19 sec)
		
		mysql> CREATE TABLE t2(a DATETIME DEFAULT (NOW()));
		Query OK, 0 rows affected (0.20 sec)
		• In MySQL, when specifying default values for BLOB, TEXT, and JSON data types, you must add parentheses to the default values. In GaussDB, you do not need to add parentheses when specifying default values

Description	Syntax	Difference

		for the preceding data types.
		• When the default value is specified, GaussDB does not check whether the default value overflows. When the default value without parentheses is specified in MySQL, MySQL checks whether the default value overflows. When the default value with parentheses is specified, MySQL does not check whether the default value overflows.
		• In GaussDB, time constants starting with DATE, TIME, or TIMESTAMP can be used to specify default values for columns. In MySQL, when time constants starting with DATE, TIME, or TIMESTAMP are used to specify default values for columns, parentheses must be added to the default values.
		-- GaussDB
		m_db=# DROP TABLE IF EXISTS t1, t2;
		DROP TABLE
		m_db=# CREATE TABLE t1(a TIMESTAMP DEFAULT TIMESTAMP '2000-01-01 00:00:00');
		CREATE TABLE
		m_db=# CREATE TABLE t2(a TIMESTAMP DEFAULT (TIMESTAMP '2000-01-01 00:00:00'));
		CREATE TABLE
		
		-- MySQL 5.7
		mysql> DROP TABLE IF EXISTS t1, t2;
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> CREATE TABLE t1(a TIMESTAMP DEFAULT TIMESTAMP '2000-01-01 00:00:00');
		ERROR 1067 (42000): Invalid default value for 'a'
		mysql> CREATE TABLE t2(a

Description	Syntax	Difference

		TIMESTAMP DEFAULT (TIMESTAMP '2000-01-01 00:00:00'));
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(TIMESTAMP '2000-01-01 00:00:00'))' at line 1
		
		-- MySQL 8.0
		mysql> DROP TABLE IF EXISTS t1, t2;
		Query OK, 0 rows affected (0.14 sec)
		
		mysql> CREATE TABLE t1(a TIMESTAMP DEFAULT TIMESTAMP '2000-01-01 00:00:00');
		ERROR 1067 (42000): Invalid default value for 'a'
		mysql> CREATE TABLE t2(a TIMESTAMP DEFAULT (TIMESTAMP '2000-01-01 00:00:00'));
		Query OK, 0 rows affected (0.19 sec)

3.2.7.4 DML

Table 3-28 DML syntax compatibility

Description	Syntax	Difference
DELETE supports deleting data from a specified partition (or subpartition).	DELETE	-
UPDATE supports ORDER BY and LIMIT.	UPDATE	-
SELECT INTO syntax	SELECT	• In GaussDB, you can use SELECT INTO to create a table based on the query result. MySQL does not support this function. • In GaussDB, the SELECT INTO syntax does not support the query result that is obtained after the set operation of multiple queries is performed.

Description	Syntax	Difference

REPLACE INTO syntax	REPLACE	Difference between the initial values of the time type. For example:
		• MySQL is not affected by the strict or loose mode. You can insert time 0 into a table.
		mysql> CREATE TABLE test(f1 TIMESTAMP NOT NULL, f2 DATETIME NOT NULL, f3 DATE NOT NULL);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> REPLACE INTO test VALUES(f1, f2, f3);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM test;
		+---------------------
		+---------------------+------------+
		| f1 | f2 | f3 |
		+---------------------
		+---------------------+------------+
		| 0000-00-00 00:00:00 | 0000-00-00 00:00:00 | 0000-00-00 |
		+---------------------
		+---------------------+------------+
		1 row in set (0.00 sec)
		• The time 0 can be successfully inserted only when GaussDB is in loose mode.
		gaussdb=# SET sql_mode = '';
		SET
		gaussdb=# CREATE TABLE test(f1 TIMESTAMP NOT NULL, f2 DATETIME NOT NULL, f3 DATE NOT NULL);
		CREATE TABLE
		gaussdb=# REPLACE INTO test VALUES(f1, f2, f3);
		REPLACE 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2 | f3
		--------------------- 	
		+---------------------+------------
		0000-00-00 00:00:00 | 0000-00-00 00:00:00 | 0000-00-00
		(1 row)
		In strict mode, the error "Incorrect Date/Time/Datetime/Timestamp/Year value" is reported.

SELECT supports multi-partition query.	SELECT	-

Description	Syntax	Difference
UPDATE supports multi-partition update.	UPDATE	-

Description	Syntax	Difference
Import data by using LOAD DATA.	LOAD DATA	When LOAD DATA is used to import data, GaussDB differs from MySQL in the following aspects: • The execution result of the LOAD DATA syntax is the same as that in M* strict mode. The loose mode is not adapted currently. • The **IGNORE** and **LOCAL** parameters are used only to ignore the conflicting rows when the imported data conflicts with the data in the table and to automatically fill default values for other columns when the number of columns in the file is less than that in the table. Other functions are not supported currently. • The **[(col_name_or_user_var [, col_name_or_user_var]...)]** parameter cannot be used to specify a column repeatedly. • The newline character specified by **[FIELDS TERMINATED BY 'string']** cannot be the same as the separator specified by **[LINES TERMINATED BY'string']**. • If the data written to a table by running **LOAD DATA** cannot be converted to the data type of the table, an error is reported. • The LOAD DATA SET expression does not support the calculation of a specified column name. • LOAD DATA applies only to tables but not views.

Description	Syntax	Difference
		• The default newline character of the file in Windows is different from that in Linux. LOAD DATA cannot identify this scenario and reports an error. You are advised to check the newline character at the end of lines in the file to be imported. • In GaussDB, when the GUC parameter **m_format_behavior_compat_options** is not set, data can be imported only from the server using LOAD DATA, regardless of whether the **LOCAL** parameter is specified. In MySQL, if the **LOCAL** parameter is specified, data can be imported from the client; otherwise, it is imported from the server. After you specify the value of this GUC parameter that includes **enable_load_data_remote_transmission** in GaussDB, the **LOCAL** parameter behavior of LOAD DATA becomes consistent with that in MySQL.
INSERT supports the VALUES reference column syntax.	INSERT INTO tabname VALUES(1,2,3) ON DUPLICATE KEY UPDATE b = VALUES(column_name)	The format of *table-name.column-name* is not supported by VALUES() in the ON DUPLICATE KEY UPDATE clause in GaussDB, but is supported in MySQL.

Description	Syntax	Difference
LIMIT differences	DELETE, SELECT, and UPDATE	The LIMIT clauses of each statement in GaussDB are different from those in MySQL. The maximum parameter value of LIMIT (of the BIG INT type) in GaussDB is **9223372036854775807**. If the actual value exceeds the number, an error is reported. In MySQL, the maximum value of LIMIT (of the unsigned LONGLONG type) is **18446744073709551615**. If the actual value exceeds the number, an error is reported. You can set a small value in LIMIT, which is rounded off during execution. The value cannot be a decimal in MySQL.

Description	Syntax	Difference
Difference in using backslashes (\)	INSERT	The usage of backslashes (\) can be determined by parameters in GaussDB and MySQL, but their default usages are different. In MySQL, the **NO_BACKSLASH_ESCAPES** parameter is used to determine whether backslashes (\) in character strings and identifiers are parsed as common characters or escape characters. By default, backslashes (\) are parsed as escape characters in character strings and identifiers. If **SET sql_mode='NO_BACKSLASH_ESCAPES';** is used, the backslashes (\) cannot be parsed as escape characters in strings and identifiers. In GaussDB, the **standard_conforming_strings** parameter is used to determine whether backslashes (\) in character strings and identifiers are parsed as common characters or escape characters. The default value is **on**, indicating that backslashes (\) are parsed as common text in common character string texts according to the SQL standard. If **SET standard_conforming_strings=off;** is used, backslashes (\) can be parsed as escape characters in character strings and identifiers.

Description	Syntax	Difference
If the inserted value is less than the number of columns, MySQL reports an error while GaussDB supplements null values.	INSERT	In GaussDB, if the column list is not specified and the inserted value is less than the number of columns, values are assigned based on the column sequence when the table is created by default. If a column has a NOT NULL constraint, an error is reported. If no NOT NULL constraint exists and a default value is specified, the default value is added to the column. If no default value is specified, **null** is added.
The columns sorted in ORDER BY must be included in the columns of the result set.	SELECT	In GaussDB, when used with the GROUP BY clause, the columns to be sorted in ORDER BY must be included in the columns of the result set retrieved by the SELECT statement. 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.
Modify constraint columns by using INSERT ON DUPLICATE KEY UPDATE.	INSERT	In GaussDB, constraint columns cannot be modified by using ON DUPLICATE KEY UPDATE, but this operation is allowed in MySQL.
Duplicate column names are allowed in the SELECT result.	SELECT	-
NATURAL JOIN in GaussDB is different from that in MySQL.	SELECT	In GaussDB, NATURAL [[LEFT | RIGHT] OUTER] JOIN allows you not to specify **LEFT | RIGHT**. If **LEFT | RIGHT** is not specified, **NATURAL OUTER JOIN** is **NATURAL JOIN**. You can use JOIN consecutively.

Description	Syntax	Difference
If the foreign key data type is timestamp or datetime, an error is reported for attempts to perform UPDATE or DELETE on a foreign table.	UPDATE/DELETE	If the foreign key data type is timestamp or datetime, an error is reported for attempts to perform UPDATE or DELETE on a foreign table, but such operations are allowed in MySQL.

Description	Syntax	Difference

Compatibility in terms of nature join and using	SELECT	• In GaussDB, join sequence is strictly from left to right. MySQL may adjust the sequence.
		• In GaussDB and MySQL, columns involving **join** in the left or right table cannot be ambiguous during natural join or using. (Generally, ambiguity is caused by duplicate names of columns in the left or right temporary table.) The join sequence differs in two databases, which may lead to different behaviors.
		– Behavior in GaussDB:
		m_regression=# CREATE TABLE t1(a int,b int);
		CREATE TABLE
		m_regression=# CREATE TABLE t2(a int,b int);
		CREATE TABLE
		m_regression=# CREATE TABLE t3(a int,b int);
		CREATE TABLE
		m_regression=# SELECT * FROM t1 JOIN t2;
		a | b | a | b
		---+---+---+---
		(0 rows)
		m_regression=# SELECT * FROM t1 JOIN t2 natural join t3; -- Failed. Duplicate contents exist in columns **a** and **b** of the temporary table obtained by **t1 join t2**. Therefore, there is ambiguity in nature join.
		ERROR: common column name "a" appears more than once in left table
		– Behavior in MySQL:
		mysql> SELECT * FROM t1 JOIN t2 NATURAL JOIN t3;
		Empty set (0.00 sec)
		mysql> SELECT * FROM (t1 join t2) NATURAL JOIN t3;
		ERROR 1052 (23000): Column 'a' in from clause is ambiguous

The WITH clause is compatible with MySQL 8.0.	SELECT, INSERT, UPDATE, and DELETE	-

Description	Syntax	Difference
Compatibility in terms of join	SELECT	Commas (,) cannot be used as a way of join in GaussDB, but can be used in MySQL. GaussDB does not support use index for join.
If the column expression in the SELECT statement is a function expression or arithmetic expression, the column name in the query result is **?column?**.	SELECT	In GaussDB, if the column expression in the SELECT statement is a function expression or arithmetic expression, the column name in the query result is **?column?**. In MySQL, the name is the corresponding expression.
SELECT export file (into outfile)	SELECT ... INTO OUFILE ...	In the file exported by using the SELECT INTO OUTFILE syntax, the display precision of values of the FLOAT, DOUBLE, and REAL types in GaussDB is different from that in MySQL. The syntax does not affect the import using COPY the values after import.

Description	Syntax	Difference

Specify schema names and table names by using SELECT/UPDATE/ INSERT/REPLACE.	SELECT/UPDATE/ INSERT/REPLACE	• When the SELECT statement is used to the projection column, MySQL supports the three-segment format of *schema name.table alias.column name*, but GaussDB does not.
		m_db=# CREATE SCHEMA test;
		CREATE SCHEMA
		m_db=# CREATE TABLE test.t1(a int);
		CREATE TABLE
		m_db=# SELECT test.alias1.a FROM t1 alias1;
		ERROR: invalid reference to FROM-clause entry for table "alias1"
		LINE 1: SELECT test.alias1.a FROM t1 alias1;
		^
		HINT: There is an entry for table "alias1", but it cannot be referenced from this part of the query.
		CONTEXT: referenced column: a
		• The three-segment format for UPDATE/REPLACE SET is *database.table.column* in MySQL, and is *table.column.field* in GaussDB, where *field* indicates the attribute in the specified composite type.
		• For INSERT ... SET, MySQL supports *column*, *table.column*, and *database.table.column*. GaussDB supports only *column* and does not support *table.column* and *database.table.column*.
		• For INSERT... SET, you can reference column names and expressions that contain column names on the right of the SET clause in MySQL. GaussDB does not support this operation.
		– Behavior in GaussDB:
		m_db=# CREATE TABLE t2 (a int default 3, b int default 5);

Description	Syntax	Difference

		CREATE TABLE
		
		m_db=# INSERT INTO t2 SET a = b + 1;
		ERROR: Column "b" does not exist.
		LINE 1: INSERT INTO t2 SET a = b + 1;
		^
		HINT: There is a column named "b" in table "t2", but it cannot be referenced from this part of the query.
		
		m_db=# INSERT INTO t2 SET a = b + 1, b = 0;
		ERROR: Column "b" does not exist.
		LINE 1: INSERT INTO t2 SET a = b + 1, b = 0;
		^
		HINT: There is a column named "b" in table "t2", but it cannot be referenced from this part of the query.
		
		m_db=# INSERT INTO t2 SET b = 0, a = b + 1;
		ERROR: Column "b" does not exist.
		LINE 1: INSERT INTO t2 SET b = 0, a = b + 1;
		^
		HINT: There is a column named "b" in table "t2", but it cannot be referenced from this part of the query.
		
		m_db=# INSERT INTO t2 SET a = a + 1;
		ERROR: Column "a" does not exist.
		LINE 1: INSERT INTO t2 SET a = a + 1;
		^
		HINT: There is a column named "a" in table "t2", but it cannot be referenced from this part of the query.
		
		m_db=# DROP TABLE t2;
		DROP TABLE
		– Behavior in MySQL:
		mysql> CREATE TABLE t2 (a int default 3, b int default 5);
		Query OK, 0 rows affected (0.07 sec)
		
		mysql> INSERT INTO t2 SET a = b + 1;
		Query OK, 1 row affected (0.02 sec)
		
		mysql> SELECT * FROM t2;

Description	Syntax	Difference

		+------+------+
		| a | b |
		+------+------+
		| 6 | 5 |
		+------+------+
		1 row in set (0.00 sec)
		
		mysql> INSERT INTO t2 SET a = b + 1, b = 0;
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM t2;
		+------+------+
		| a | b |
		+------+------+
		| 6 | 5 |
		| 6 | 0 |
		+------+------+
		2 rows in set (0.00 sec)
		
		mysql> INSERT INTO t2 SET b = 0, a = b + 1;
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM t2;
		+------+------+
		| a | b |
		+------+------+
		| 6 | 5 |
		| 6 | 0 |
		| 1 | 0 |
		+------+------+
		3 rows in set (0.00 sec)
		
		mysql> INSERT INTO t2 SET a = a + 1;
		Query OK, 1 row affected (0.02 sec)
		
		mysql> SELECT * FROM t2;
		+------+------+
		| a | b |
		+------+------+
		| 6 | 5 |
		| 6 | 0 |
		| 1 | 0 |
		| 4 | 5 |
		+------+------+
		4 rows in set (0.00 sec)
		
		mysql> DROP TABLE t2;
		Query OK, 4 rows affected (0.40 sec)

Description	Syntax	Difference
The execution sequence of UPDATE SET is different from that of MySQL.	UPDATE ... SET	In MySQL, UPDATE SET is performed in sequence. The results of UPDATE at the front affect subsequent results of UPDATE, and the same column can be updated for multiple times. In GaussDB, all related data is obtained first, and then UPDATE is performed on the data at a time. The same column cannot be updated for multiple times.
IGNORE feature	UPDATE/DELETE/ INSERT	The execution process in MySQL is different from that in GaussDB. Therefore, the number and information of generated warnings may be different.

Description	Syntax	Difference
SHOW COLUMNS syntax	SHOW	• User permission verification is different from that of MySQL. – In GaussDB, you need the USAGE permission on the schema of a specified table and table-level or column-level permissions on the specified table. Only information about columns with the SELECT, INSERT, UPDATE, REFERENCES, and COMMENT permissions is displayed. – In MySQL, you need table-level or column-level permissions on a specified table. Only information about columns with the SELECT, INSERT, UPDATE, REFERENCES, and COMMENT permissions is displayed. • When the LIKE and WHERE clauses involve string comparison operations, the collation is different from that in MySQL. – **utf8_general_ci** is used in MySQL. – The **collation_connection** of the current client is used as the collation in GaussDB. In GaussDB, you are advised not to select columns other than the returned fields in the WHERE clause. Otherwise, unexpected errors may occur.

Description	Syntax	Difference
		-- Expected error m_db=# SHOW FULL COLUMNS FROM t02 WHERE `b`='pri'; ERROR: Column "b" does not exist. LINE 1: SHOW FULL COLUMNS FROM t02 WHERE `b`='pri'; ^ -- Unexpected error m_db=# SHOW FULL COLUMNS FROM t02 WHERE `c`='pri'; ERROR: input of anonymous composite types is not implemented LINE 1: SHOW FULL COLUMNS FROM t02 WHERE `c`='pri'; ^
SHOW CREATE DATABASE syntax	SHOW	User permission verification is different from that of MySQL. • In GaussDB, you need the USAGE permission on a specified schema. • In MySQL, you need database-level permissions (except GRANT OPTION and USAGE), table-level permissions (except GRANT OPTION), or column-level permissions.

Description	Syntax	Difference
SHOW CREATE TABLE syntax	SHOW	• User permission verification is different from that of MySQL. – In GaussDB, you need the USAGE permission on the schema where a specified table is located and table-level permissions on the specified table. – Table-level permissions (except GRANT OPTION) of the specified table are required in MySQL. • The returned statements for table creation are different from those in MySQL. – In GaussDB, indexes are returned as CREATE INDEX statements. In MySQL, indexes are returned as CREATE TABLE statements. In GaussDB, the range of optional parameters supported by the CREATE INDEX syntax is different from that supported by the CREATE TABLE syntax. Therefore, some indexes cannot be created in CREATE TABLE statements. – In GaussDB, the **ENGINE** and **ROW_FORMAT** options of CREATE TABLE are adapted only for the syntax but do not take effect. Therefore, they are not displayed in the returned statements for table creation. • These statements are compatible with MySQL

Description	Syntax	Difference
		only after the compatibility parameter **m_format_dev_version** is set to **'s2'**. The compatibility parameter takes effect by changing the positions of column comments, table comments, **ON COMMIT** option for global temporary tables, PRIMARY KEY and UNIQUE constraints (where the **USING INDEX TABLESPACE** option is no longer displayed), and index comments.

Description	Syntax	Difference
SHOW CREATE VIEW syntax	SHOW	• User permission verification is different from that of MySQL. – In GaussDB, you need the USAGE permission on the schema where a specified view is located and table-level permissions on the specified view. – In MySQL, you need the table-level SELECT and table-level SHOW VIEW permissions on the specified view. • The returned statements for view creation are different from those in MySQL. If a view is created in the format of **SELECT * FROM** *tbl_name*, * is not expanded in GaussDB but expanded in MySQL. • The *character_set_client* and *collation_connection* columns in the returned result are different from those in MySQL. – The session values of system variables *character_set_client* and *collation_connection* are displayed during view creation in MySQL – Related metadata is not recorded in GaussDB and **NULL** is displayed.

Description	Syntax	Difference
SHOW PROCESSLIST syntax	SHOW	In GaussDB, the column content and case in the query result of this command are the same as those in the information_schema.processlist view. In MySQL, the column content and case may be different. • In GaussDB, common users can access only their own thread information. Users with the SYSADMIN permission can access thread information of all users. • In MySQL, common users can access only their own thread information. Users with the PROCESS permission can access thread information of all users.
SHOW [STORAGE] ENGINES	SHOW	In GaussDB, the column content and case of the query result of this command are the same as those in the information_schema.engines view. In MySQL, they may be different from those in the view. The query results of this command are different in MySQL and GaussDB because the databases have different storage engines.
SHOW [SESSION] STATUS	SHOW	In GaussDB, the column content and case of the query result of this command are the same as those in the information_schema.session_status view. In MySQL, they may be different from those in the view. Currently, GaussDB supports only **Threads_connected** and **Uptime**.

Description	Syntax	Difference
SHOW [GLOBAL] STATUS	SHOW	In GaussDB, the column content and case of the query result of this command are the same as those in the information_schema.global_status view. In MySQL, they may be different from those in the view. Currently, GaussDB supports only **Threads_connected** and **Uptime**.
SHOW INDEX	SHOW	• User permission verification is different from that of MySQL. – In GaussDB, you need the USAGE permission on a specified schema and table-level or column-level permissions on a specified table. – In MySQL, you need table-level (except GRANT OPTION) or column-level permission on the specified table. • Temporary tables in GaussDB are stored in independent temporary schemas. When using the FROM or IN db_name condition to display the index information of a specified temporary table, you must specify **db_name** as the schema where the temporary table is located. Otherwise, the system displays a message indicating that the temporary table does not exist. This is different from MySQL in some cases.

Description	Syntax	Difference
SHOW SESSION VARIABLES	SHOW	In GaussDB, the column content and case of the query result are the same as those in the information_schema.session_variables view. In MySQL, they may be different from those in the view.
SHOW GLOBAL VARIABLES	SHOW	In GaussDB, the column content and case of the query result are the same as those in the information_schema.global_variables view. In MySQL, they may be different from those in the view.
SHOW CHARACTER SET	SHOW	In GaussDB, the column content and case of the query result are the same as those in the information_schema.character_sets view. In MySQL, they may be different from those in the view.
SHOW COLLATION	SHOW	In GaussDB, the column content and case of the query result are the same as those in the information_schema.collations view. In MySQL, they may be different from those in the view.
EXCEPT Syntax	SELECT	-
SELECT supports the STRAIGHT_JOIN syntax.	SELECT	The execution plans generated in the multi-table JOIN scenarios in GaussDB may be different from those in MySQL.

Description	Syntax	Difference
SHOW TABLES	SHOW	• The LIKE behavior is different. For details, see "LIKE" in **Operators**. • The WHERE expression behavior is different. For details, see "WHERE" in GaussDB. • In GaussDB, permissions on tables and databases must be assigned to users separately. The database to be queried must be available to users on the SHOW SCHEMAS. Users must have permissions on both tables and databases. MySQL can be accessed as long as you have table permissions. • In GaussDB, the verification logic preferentially checks whether a schema exists and then checks whether the current user has the permission on the schema, which is different from that in MySQL.

Description	Syntax	Difference
SHOW TABLE STATUS	SHOW	• In GaussDB, the syntax displays data depending on the tables view under information_schema. In MySQL, the tables view specifies tables. • In GaussDB, permissions on tables and databases must be assigned to users separately. The database to be queried must be available to users on the SHOW SCHEMAS. Users must have permissions on both tables and databases. MySQL can be accessed as long as you have table permissions. • In GaussDB, the verification logic preferentially checks whether a schema exists and then checks whether the current user has the permission on the schema, which is different from that in MySQL.
HAVING syntax	SELECT	In GaussDB, HAVING can only reference columns in the GROUP BY clause or columns used in aggregate functions. MySQL supports more: it allows HAVING to reference SELECT columns in the list and columns in external subqueries.

Description	Syntax	Difference

SELECT followed by a row expression	SELECT	In MySQL, SELECT cannot be followed by a row expression, but in GaussDB, SELECT can be followed by a row expression.
		Behavior in MySQL:
		`mysql> SELECT row(1,2);`
		`ERROR 1241 (21000): Operand should contain 1 column(s)`
		Behavior in GaussDB:
		`m_db=# SELECT row(1,2);`
		` row(1,2)`
		`----------`
		` (1,2)`
		`(1 row)`

Description	Syntax	Difference

SELECT FOR SHARE/FOR UPDATE/ LOCK IN SHRAE MODE	SELECT	• The FOR SHARE/FOR UPDATE/LOCK IN SHARE MODE and UNION/ EXCEPT/DISTINCT/GROUP BY/HAVING clauses cannot be used together in GaussDB. They can be used together in MySQL 5.7 (except in the FOR SHARE/EXCEPT syntax) and MySQL 8.0.
		• When a lock clause is used together with the LEFT/ RIGHT [OUTER] JOIN clause, the LEFT JOIN cannot be used to lock the right table, and the RIGHT JOIN clause cannot be used to lock the left table. In MySQL, tables on both sides of JOIN can be locked at the same time.
		• In MySQL, multiple lock clauses cannot be specified for the same table, while GaussDB supports this operation and the strongest lock will take effect.
		-- GaussDB
		m_db=# DROP TABLE IF EXISTS t1;
		DROP TABLE
		
		m_db=# CREATE TABLE t1(a INT, b INT);
		CREATE TABLE
		
		m_db=# INSERT INTO t1 VALUES(1,2);
		INSERT 0 1
		
		m_db=# SELECT * FROM t1 FOR UPDATE OF t1 LOCK IN SHARE MODE;
		a | b
		---+---
		1 | 2
		(1 row)
		
		m_db=# DROP TABLE t1;
		DROP TABLE
		
		-- MySQL
		mysql> DROP TABLE IF EXISTS t1;
		Query OK, 0 rows affected (0.05 sec)

Description	Syntax	Difference

		mysql> CREATE TABLE t1(a INT, b INT);
		Query OK, 0 rows affected (0.09 sec)
		
		mysql> INSERT INTO t1 VALUES(1,2);
		Query OK, 1 row affected (0.01 sec)
		
		mysql> SELECT * FROM t1 FOR UPDATE OF t1 LOCK IN SHARE MODE;
		ERROR 3569 (HY000): Table t1 appears in multiple locking clauses.
		
		mysql> DROP TABLE t1;
		Query OK, 0 rows affected (0.05 sec)

Description	Syntax	Difference

SELECT syntax	SELECT	• In GaussDB, the HAVING clause must be used together with the GROUP BY clause or aggregate functions. In MySQL, specifying only the HAVING clause in a query statement is allowed.
		• In GaussDB, HAVING can only reference columns in the GROUP BY clause or columns used in aggregate functions. MySQL supports more: it allows HAVING to reference SELECT columns in the list and columns in external subqueries.
		• In GaussDB and MySQL 5.7, GROUP BY WITH ROLLUP cannot be used together with the DISTINCT and ORDER BY clauses. However, MySQL 8.0 supports this operation.
		• In GaussDB, when an empty table is queried by using the specified WITH ROLLUP statement, the query result is an empty row. In contrast, the query result in MySQL is empty.
		• In GaussDB, a table alias with the column name can be specified by using the FROM clause. In MySQL 5.7, a table alias with the column name cannot be specified. In MySQL 8.0, it is allowed only in a subquery.
		-- GaussDB
		m_db=# DROP TABLE IF EXISTS t1;
		DROP TABLE
		m_db=# CREATE TABLE t1(a INT, b INT);
		CREATE TABLE
		m_db=# INSERT INTO t1 VALUES(1,2);
		INSERT 0 1

Description	Syntax	Difference

		m_db=# SELECT * FROM t1 t2(a, b);
		 a | b
		---+---
		 1 | 2
		(1 row)
		
		m_db=# SELECT * FROM (SELECT * FROM t1) t2(a, b);
		 a | b
		---+---
		 1 | 2
		(1 row)
		
		-- MySQL 5.7
		mysql> DROP TABLE IF EXISTS t1;
		Query OK, 0 rows affected, 1 warning (0.00 sec)
		
		mysql> CREATE TABLE t1(a INT, b INT);
		Query OK, 0 rows affected (0.03 sec)
		
		mysql> INSERT INTO t1 VALUES(1,2);
		Query OK, 1 row affected (0.01 sec)
		
		mysql> SELECT * FROM t1 t2(a, b);
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(a, b)' at line 1
		mysql> SELECT * FROM (SELECT * FROM t1) t2(a, b);
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(a, b)' at line 1
		
		-- MySQL 8.0
		mysql> DROP TABLE IF EXISTS t1;
		Query OK, 0 rows affected (0.10 sec)
		
		mysql> CREATE TABLE t1(a INT, b INT);
		Query OK, 0 rows affected (0.18 sec)
		
		mysql> INSERT INTO t1 VALUES(1,2);
		Query OK, 1 row affected (0.03 sec)
		
		mysql> SELECT * FROM t1 t2(a, b);
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(a, b)' at

Description	Syntax	Difference

		line 1
		mysql> SELECT * FROM (SELECT * FROM t1) t2(a, b);
		+------+------+
		| a | b |
		+------+------+
		| 1 | 2 |
		+------+------+
		1 row in set (0.00 sec)
		• If a query statement does not contain the FROM clause, GaussDB supports the WHERE clause, which is the same as that in MySQL 8.0. MySQL 5.7 does not support the WHERE clause.
		-- GaussDB
		m_db=# SELECT 1 WHERE true;
		1
		--- 	
		1
		(1 row)
		
		-- MySQL 5.7
		mysql> SELECT 1 WHERE true;
		ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'where true' at line 1
		
		-- MySQL 8.0
		mysql> SELECT 1 WHERE true;
		+---+
		| 1 |
		+---+
		| 1 |
		+---+
		1 row in set (0.00 sec)

Description	Syntax	Difference
INSERT ... ON DUPLICATE KEY UPDATE syntax	INSERT	• The format of *table-name.column-name* is not supported by VALUES() in the ON DUPLICATE KEY UPDATE clause in GaussDB, but is supported in MySQL. • In GaussDB, constraint columns cannot be modified by using ON DUPLICATE KEY UPDATE, but this operation is allowed in MySQL. • In the INSERT... query ON DUPLICATE KEY UPDATE statement, if query is a subquery, the ON DUPLICATE KEY UPDATE clause cannot reference column names in the subquery in GaussDB, while MySQL supports this behavior. • In MySQL, when you use the ON DUPLICATE KEY UPDATE clause to update multiple columns, the result of the previous UPDATE statement affects the subsequent results. In addition, you can update the same column for multiple times. In GaussDB, the result of the previous UPDATE operation does not affect the subsequent results. In addition, the same column cannot be updated for multiple times. • When the UPDATE operation is performed on inserted data that violates the UNIQUE constraint, the number of affected rows returned by GaussDB is different from that returned by MySQL. When a data record is updated,

Description	Syntax	Difference
		GaussDB returns **1** and MySQL returns **2**. If such update does not change the value of an existing row, GaussDB returns **1** and MySQL returns **0**.

Description	Syntax	Difference

If the value of a GROUP BY HAVING column is an alias of the query projection column and overlaps with the table column name, the syntax may be ambiguous, and the result may be different from that of MySQL.	SELECT	• If the alias of a query expression is the same as a table column name, in GaussDB, the name in the HAVING condition indicates the column name no matter whether the GROUP BY clause is followed by a number or a name.
		• In MySQL, if the GROUP BY clause is followed by a number, the name in the HAVING condition indicates the expression alias in the projection column. If the GROUP BY clause is followed by a name, the name in the HAVING condition indicates the column name.
		m_db=# CREATE TABLE t1(col_int int);
		CREATE TABLE
		m_db=# INSERT INTO t1 VALUES(1),(2);
		INSERT 0 2
		m_db=# SELECT abs(col_int) + 2 AS col_int FROM t1 GROUP BY col_int HAVING col_int > 2;
		col_int
		--------- 	
		(0 rows)
		
		m_db=# SET sql_mode = '';
		SET
		m_db=# SELECT abs(col_int) + 2 AS col_int FROM t1 GROUP BY 1 HAVING col_int > 2;
		col_int
		--------- 	
		(0 rows)
		
		mysql> CREATE TABLE t1(col_int int);
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> INSERT INTO t1 VALUES(1),(2);
		Query OK, 2 rows affected (0.00 sec)
		Records: 2 Duplicates: 0 Warnings: 0
		
		mysql> SELECT abs(col_int) + 2 AS col_int FROM t1 GROUP BY col_int HAVING col_int > 2;
		Empty set, 2 warnings (0.00 sec)
		
		mysql> SELECT abs(col_int) + 2 AS col_int FROM t1 GROUP BY 1 HAVING col_int > 2;

Description	Syntax	Difference
		+---------+ | col_int | +---------+ | 3 | | 4 | +---------+ 2 rows in set (0.00 sec)

3.2.7.5 DCL

Table 3-29 DCL syntax compatibility

Description	Syntax	Difference
Set names with COLLATE specified.	SET [SESSION | LOCAL] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};	GaussDB does not allow **charset_name** to be different from the database character set. For details, see "SQL Reference > SQL Syntax > SQL Statements > S > SET" in *M Compatibility Developer Guide*.

Switch the current mode with USE.	USE schema_name	If the USE statement is used to specify a schema and the user does not have USAGE permissions on the schema, MySQL reports an error while GaussDB specifies the current schema as null.
		-- MySQL
		mysql> USE test;
		ERROR 1044 (42000): Access denied for user 'u1'@'%' to database 'test'
		
		-- GaussDB
		m_db=> USE test;
		SET
		m_db=> SELECT database();
		ERROR: function returned NULL
		CONTEXT: referenced column: database

3.2.7.6 Other Statements

Table 3-30 Compatibility of other syntaxes

Description	Syntax	Difference
Transaction-related syntax	Default database isolation level	The default isolation level of an M-compatible database is READ COMMITTED, and that of MySQL is REPEATABLE READ. Only the READ COMMITTED and REPEATABLE READ isolation levels take effect in M-compatible databases.
Transaction-related syntax	Transaction nesting	In M-compatible mode, nested transactions are not automatically committed, but in MySQL, they are automatically committed.
Transaction-related syntax	Autocommit	In M-compatible mode, GaussDB is used for storage and the GaussDB transaction mechanism is inherited. If DDL or DCL is executed in a transaction, the transaction is not automatically committed. In MySQL, if DDL, DCL, management-related, or lock-related statements are executed, the transaction is automatically committed.
Transaction-related syntax	Rollback is required after an error is reported.	If an error is reported for a transaction in an M-compatible database, rollback needs to be performed. There is no such restriction in MySQL.
Transaction-related syntax	Lock mechanism	The M-compatible lock mechanism can be used only in transaction blocks. There is no such restriction in MySQL.

Description	Syntax	Difference
Lock mechanism	Lock mechanism	• After the read lock is obtained, write operations cannot be performed on the current session in MySQL, but write operations can be performed on the current session in an M-compatible database. • After MySQL locks a table, an error is reported when other tables are read. M-compatible does not have such restriction. • In MySQL, if the lock of the same table is obtained in the same session, the previous lock is automatically released and the transaction is committed. M-compatible databases do not have this mechanism. • M-compatible databases allow LOCK TABLE to be used only inside a transaction block, and have no UNLOCK TABLE command. Locks are always released at the end of transactions.

Description	Syntax	Difference
PBE	PBE	• In an M-compatible database, if a PREPARE statement with the same name is repeatedly created, an error is reported, indicating that the statement already exists. You need to delete the existing statement first. In MySQL, the old statement will be overwritten. • M-compatible databases and MySQL report errors in different phases, such as parsing and execution, during SQL statement execution. PREPARE statements process prepared statements till the parsing phrase. Therefore, in abnormal scenarios in PBE, an M-compatible database may be different from that in MySQL in terms of whether the error is reported in the PREPARE or EXECUTE phase.
Single-line comment syntax	Single-line comment syntax	The single-line comment syntax is consistent with MySQL only when the **m_format_behavior_compat_options** parameter is set to **'forbid_none_space_comment'**.

3.2.7.7 Users and Permissions

Overview

In M-compatible mode, the behaviors and syntaxes related to user and permission control inherit the GaussDB mechanism but are not synchronized with those in MySQL.

User and permission behaviors are the same as those in GaussDB. For details, see "Database Security Management > Managing Users and Their Permissions" in *Developer Guide*.

Some syntaxes for users and permissions are tailored in GaussDB. For details about the syntaxes, see "SQL Reference > SQL Syntax > SQL Statements" in *M Compatibility Developer Guide*. For details about the syntax differences between an M-compatible database and GaussDB, see **Table 3-31**.

Table 3-31 Syntax differences between an M-compatible database and GaussDB

Syntax	Overview	Difference
CREATE ROLE	Creates a role.	In M-compatible mode, options involving the following keywords cannot be specified: **ENCRYPTED**, **UNENCRYPTED**, **RESOURCE POOL**, **PERM SPACE**, **TEMP SPACE**, and **SPILL SPACE**. When a user is created, a schema with the same name as the user is automatically created in an M-compatible database, but it is not created in MySQL.
CREATE USER	Creates a user.	
CREATE GROUP	Creates a user group. CREATE GROUP is the alias of CREATE ROLE and is not recommended.	
ALTER ROLE	Modifies role attributes.	
ALTER USER	Modifies user attributes.	
ALTER GROUP	Modifies the attributes of a user group.	-
DROP ROLE	Deletes a role.	-
DROP USER	Deletes a user.	-
DROP GROUP	Deletes a user group.	-
DROP OWNED	Deletes the database objects owned by a database role.	-
REASSIGN OWNED	Changes the owner of a database object.	This syntax is not supported in an M-compatible database.
GRANT	Grants permissions to roles and users.	In an M-compatible database, permissions on objects such as functions, stored procedures, tablespaces, and database links cannot be granted or revoked.
REVOKE	Revokes permissions from one or more roles.	
ALTER DEFAULT PRIVILEGES	Sets the permissions that will be granted to objects created in the future. (It does not affect permissions granted to existing objects.)	This syntax is not supported in an M-compatible database.

Differences

- Syntax format differences

  For details about the M-compatible permission granting syntaxes, see "SQL Reference > SQL Syntax > G > GRANT" in *M Compatibility Developer Guide*. The permission granting syntax in MySQL is as follows:

```
-- Global, database-level, table-level, and stored procedure–level permission granting syntax
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user [auth_option] [, user [auth_option]] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH {GRANT OPTION | resource_option} ...]

-- Syntax for granting permissions to a user proxy
GRANT PROXY ON user
    TO user [, user] ...
    [WITH GRANT OPTION]

object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user:
    'user_name'@'host_name'

auth_option: {
    IDENTIFIED BY 'auth_string'
  | IDENTIFIED WITH auth_plugin
  | IDENTIFIED WITH auth_plugin BY 'auth_string'
  | IDENTIFIED WITH auth_plugin AS 'auth_string'
  | IDENTIFIED BY PASSWORD 'auth_string'
}

tls_option: {
    SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
}

resource_option: {
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
}
```

- Differences in types of permissions granted

  In MySQL, the following types of permissions can be granted.

Table 3-32 Types of permissions that can be granted in MySQL

Permission Type	Definition and Permission Level
ALL [PRIVILEGES]	Grants all permissions of a specified access level, except **GRANT OPTION** and **PROXY**.
ALTER	Enables **ALTER TABLE**. Level: global, database, and table.
ALTER ROUTINE	Allows you to modify or delete stored procedures. Level: global, database, and routine.
CREATE	Enables database and table creation. Level: global, database, and table.
CREATE ROUTINE	Enables stored procedure creation. Level: global and database.
CREATE TABLESPACE	Allows you to create, modify, or delete tablespaces or log file groups. Level: global.
CREATE TEMPORARY TABLES	Enables **CREATE TEMPORARY TABLE**. Level: global and database.
CREATE USER	Enable **CREATE USER**, **DROP USER**, **RENAME USER**, and **REVOKE ALL PRIVILEGES**. Level: global.
CREATE VIEW	Allows you to create or modify views. Level: global, database, and table.
DELETE	Enable **DELETE**. Level: global, database, and table.
DROP	Allows you to delete databases, tables, or views. Level: global, database, and table.
EVENT	Enable scheduled tasks. Level: global and database.
EXECUTE	Allows you to execute stored procedures. Level: global, database, and stored procedure.
FILE	Allows you to enable the server to read or write files. Level: global.
GRANT OPTION	Allows you to grant permissions to or remove permissions from other accounts. Level: global, database, table, stored procedure, and proxy.
INDEX	Allows you to create or delete indexes. Level: global, database, and table.
INSERT	Enables **INSERT**. Level: global, database, table, and column.

Permission Type	Definition and Permission Level
LOCK TABLES	LOCK TABLES is enabled on tables with the SELECT permission. Level: global and database.
PROCESS	Allows you to view all running threads through **SHOW PROCESSLIST**. Level: global.
PROXY	Enables a user proxy. Level: from user to user.
REFERENCES	Enables foreign key creation. Level: global, database, table, and column.
RELOAD	Enables **FLUSH**. Level: global.
REPLICATION CLIENT	Allows you to query the location of the source server or replica server. Level: global.
REPLICATION SLAVE	Allows replicas to read binary logs from the source. Level: global.
SELECT	Enables **SELECT**. Level: global, database, table, and column.
SHOW DATABASES	Enables **SHOW DATABASES** to display all databases. Level: global.
SHOW VIEW	Enables **SHOW CREATE VIEW**. Level: global, database, and table.
SHUTDOWN	Enables **mysqladmin shutdown**. Level: global.
SUPER	Enables other management operations, such as the **CHANGE MASTER TO**, **KILL**, **PURGE BINARY LOGS**, **SET GLOBAL**, and **mysqladmin debug** commands. Level: global.
TRIGGER	Enables **TRIGGER**. Level: global, database, and table.
UPDATE	Enables **UPDATE**. Level: global, database, table, and column.
USAGE	Equivalent to "no privilege".

M-compatible databases support the following permissions by level:

Table 3-33 Types of permissions that can be granted in M-compatible databases

Object	Permissions That Can Be Granted
Database	CREATE, CONNECT, TEMPORARY, TEMP, ALTER, DROP, and COMMENT

Object	Permissions That Can Be Granted
Schema	CREATE, USAGE, ALTER, DROP, and COMMENT
Table and view	SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, ALTER, DROP, COMMENT, INDEX, and VACUUM
Column	SELECT, INSERT, UPDATE, REFERENCES, and COMMENT
Sequence	SELECT, USAGE, UPDATE, ALTER, DROP, and COMMENT

- In MySQL, *'*.*'* represents a global-level authorization object; in GaussDB, *'{DATABASE} db_name'* represents a database-level authorization object. The database level of GaussDB corresponds to the global level of MySQL.
- In MySQL, *'schema_name.*'* represents a database/schema-level authorization object; in GaussDB, *'{SCHEMA} schema_name'* represents a schema-level authorization object. The schema level of GaussDB corresponds to the database/schema level of MySQL.
- In MySQL, a username consists of two parts: *username*@*hostname*, but a username is only itself in M-compatible databases.
- MySQL allows you to modify user authentication, secure connection, and resource parameter attributes (including **auth_option**, **tls_option**, and **resource option**) with the GRANT syntax. In M-compatible databases, permission granting syntax does not support this function, and you need to use CREATE USER and ALTER USER to set user attributes.
- MySQL supports permission granting with a user proxy. GRANT PROXY ON is used to manage permissions of users in a unified manner. MySQL 5.7 does not provide the role mechanism, but MySQL 8.0 and M-compatible databases provide the role mechanism. If a role can manage and control the permissions of users in a unified manner, it can replace GRANT PROXY ON.
- M-compatible databases have a concept called public. All users have public permissions and they can query some system catalogs and system views. Users can grant or revoke public permissions. In MySQL, newly created users have only the global usage permission, which is almost low to none. They have only the permission to connect to the database and query the information_schema database.
- In M-compatible databases, the owner of an object has all permissions on the object by default. For security purposes, the owner can discard some permissions. However, ALTER, DROP, COMMENT, INDEX, VACUUM, and re-grantable permissions on the object are implicitly inherent permissions of the owner: MySQL does not have a concept called owner. Even if a user creates a table, the user cannot perform operations such as IUD on the table without being granted the corresponding permissions.
- In MySQL, All users have the USAGE permission, which indicates no permission. When **REVOKE** or **GRANT USAGE** is executed, no modification is performed. In M-compatible databases, the USAGE permission has the following meanings:

  - For schemas, USAGE allows access to objects contained in the schema. Without this permission, it is still possible to see the object names.
  - For sequences, USAGE allows use of the nextval function.
- In M-compatible databases, administrator roles can be set for users, including system administrator (SYSADMIN), security administrator (CREATEROLE), audit administrator (AUDITADMIN), monitor administrator (MONADMIN), O&M administrator (OPRADMIN), and security policy administrator (POLADMIN). By default, system administrators with the SYSADMIN attribute have the highest permission in the system. After separation of duties is enabled, a system administrator does not have the CREATEROLE or AUDITADMIN attribute. That is, the system administrator can neither create roles or users, nor view or maintain database audit logs. In MySQL, administrator roles cannot be set for users, and there is no design for separation of duties.
- In M-compatible databases, the ANY permission can be granted to a user, indicating that the user can have the corresponding permission in non-system mode, including CREATE ANY TABLE, SELECT ANY TABLE, and CREATE ANY INDEX. In MySQL, ANY permission cannot be granted.
- MySQL provides **SHOW GRANTS** to query user permissions. In M-compatible databases, you can run a gsql client meta-command **'\l+'**, **'\dn+'**, or **'\dp'** to query permission information, or query related columns in system catalogs such as pg_namespace, pg_class, and pg_attribute for permission information.
- When a database, table, or column is deleted from MySQL, the related permission granting information is still retained in the system catalog. If an object with the same name is created again, the user still has the original permissions. In M-compatible databases, when a database, table, or column is deleted, related permission granting information is deleted. If an object with the same name is created again, permissions need to be granted again.
- When granting database-level permissions, MySQL supports fuzzy match of database names using underscores (_) and percent signs (%). However, M-compatible databases do not support fuzzy match of object names using special characters such as underscores (_) or percent signs (%), which are identified as common characters.
- In MySQL, if a user specified in the GRANT statement does not exist, a user account is created by default (this feature has been removed from MySQL 8.0). In M-compatible databases, permissions cannot be granted to users who are not created.

3.2.7.8 System Catalogs and System Views

Table 3-34 Differences between M-compatible databases and GaussDB in terms of system catalogs or views

System Catalog or System View	Column	Difference
information_schema.columns	generation_expression	The output of this column varies due to different string concatenation logics of expressions in M-compatible mode and MySQL.

System Catalog or System View	Column	Difference
information_schema.columns	data_type	The output result of this column in M-compatible mode, having not been modified due to the data type format_type involved, is different from that in MySQL.
information_schema.columns	column_type	The output result of this column in M-compatible mode, having not been modified due to the data type format_type involved, is different from that in MySQL.
information_schema.tables	engine	In M-compatible mode: • ENGINE is aligned with data of information_schema.engines. • In some system catalogs, ENGINE is left empty. • If the default table is an ASTORE table and **STORAGE_TYPE** is not specified, **ENGINE** is empty.
information_schema.tables	version	This column is not supported in M-compatible mode.
information_schema.tables	row_format	This column is not supported in M-compatible mode.
information_schema.tables	avg_row_length	In M-compatible mode, the result of dividing the size of the data files by the number of all tuples (including live tuples and dead tuples) is used. If there is no tuple in the table, the value is **null**.
information_schema.tables	max_data_length	This column is not supported in M-compatible mode.
information_schema.tables	data_free	In M-compatible mode, it indicates the result of (Number of dead tuples/Total number of tuples) x Data file size. If there is no tuple in the table, the value is **null**.
information_schema.tables	check_time	This column is not supported in M-compatible mode.
information_schema.tables	create_time	The behavior of this column in M-compatible mode is different from that in MySQL. When a view is created in MySQL, this column is set to **null**. In M-compatible mode, the actual table creation time is displayed. The value is **null** if it is a table or view provided by the database.
information_schema.tables	update_time	The value is **null** if it is a table or view provided by the M-compatible database.

System Catalog or System View	Column	Difference
information_schema.statistics	collation	The value can only be **A** or **D** but not NULL in M-compatible mode.
information_schema.statistics	packed	This column is not supported in M-compatible mode.
information_schema.statistics	sub_part	This column is not supported in M-compatible mode.
information_schema.statistics	comment	This column is not supported in M-compatible mode.
information_schema.partitions	subpartition_name	In M-compatible mode, if the partition is not a level-2 partition, the value is **null**.
information_schema.partitions	subpartition_ordinal_position	In M-compatible mode, if the partition is not a level-2 partition, the value is **null**.
information_schema.partitions	partition_method	In M-compatible mode: Partitioning policy. If a partition is not specified, the value is **null**. ● **'r'**: range partition. ● **'i'**: interval partition. ● **'l'**: list partition. ● **'h'**: hash partition.
information_schema.partitions	subpartition_method	In M-compatible mode: Subpartitioning policy. If a subpartition is not specified, the value is **null**. ● **'r'**: range partition. ● **'i'**: interval partition. ● **'l'**: list partition. ● **'h'**: hash partition.
information_schema.partitions	partition_description	In M-compatible mode, there are partitions and subpartitions.
information_schema.partitions	partition_expression	This column is not supported in M-compatible mode.

System Catalog or System View	Column	Difference
information_schema.partitions	subpartition_expression	This column is not supported in M-compatible mode.
information_schema.partitions	data_length	This column is not supported in M-compatible mode.
information_schema.partitions	max_data_length	This column is not supported in M-compatible mode.
information_schema.partitions	index_length	This column is not supported in M-compatible mode.
information_schema.partitions	data_free	This column is not supported in M-compatible mode.
information_schema.partitions	create_time	This column is not supported in M-compatible mode.
information_schema.partitions	update_time	This column is not supported in M-compatible mode.
information_schema.partitions	check_time	This column is not supported in M-compatible mode.
information_schema.partitions	checksum	This column is not supported in M-compatible mode.
information_schema.partitions	partition_comment	This column is not supported in M-compatible mode.
information_schema.partitions	nodegroup	This column is not supported in M-compatible mode.

NOTE

- The precision range cannot be specified for the command output of the integer type in a view. For example, the bigint(1) type in MySQL corresponds to the bigint type in M-compatible mode, and the bigint(21) unsigned type in MySQL corresponds to the bigint unsigned type in M-compatible mode.
- The int type in MySQL corresponds to the integer type in M-compatible mode.
- M-compatible mode does not support columns of the set and enum types that are supported in MySQL. This version does not support or display **Column_priv** column in the m_schema.columns_priv view, **Table_priv,Column_priv** column in the m_schema.tables_priv view, **Routine_type,Proc_priv** column in the m_schema.procs_priv view, the **type,language,sql_data_access,is_deterministic,security_type,sql_mode** column in the m_schema.proc view, or the **type** column in the m_schema.func view.
- **table_rows**, **avg_row_length**, **data_length**, **data_free**, **index_length**, and **cardinality** in information_schema.tables and **cardinality** in information_schema.statistics are obtained based on statistics. Therefore, run **ANALYZE** to update statistics before viewing them. (If data is updated in the database, you are advised to delay running **ANALYZE**.)
- The index columns contained in information_schema.statistics must be complete table columns in the created indexes. If the index columns are expressions, they are not in this view.
- **table_row** in information_schema.partitions is obtained based on statistics. Before viewing the value, run **ANALYZE** to update the statistics. (If data is updated in the database, you are advised to delay running **ANALYZE**.)
- The format of the **grantee** column supported in MySQL is '*user_name*'**@**'*host_name*'. In an M-compatible database, it is the name of the user or role to which the permission is granted.
- For the **host** column supported in the M-compatible database, the **hostname** of the current node is returned.
- In MySQL, you need the permission before viewing m_schema.tables_priv, information_schema.user_privileges, information_schema.schema_privileges, information_schema.table_privileges, information_schema.column_privileges, m_schema.columns_priv, m_schema.func, and m_schema.procs_priv. In an M-compatible database, you can view them with the default permission. For example, for table **t1**, you need the corresponding permission in MySQL so that you can view the corresponding permission information in the permission view. In an M-compatible database, you can view the permission information related to table **t1** in the view.
- A system view in m_schema is a system catalog in MySQL.
- The collations of **VIEW_DEFINITION** in information_schema.views and **ROUTINE_DEFINITION** in information_schema.routines are not controlled.
- For the view columns of the character type listed in "Schemas" in *M Compatibility Developer Guide*, the character set is utf8mb4, and the collation is utf8mb4_bin or utf8mb4_general_ci, and the collation priority is the priority of columns of data types that support collation described in "SQL Reference > Character Set and Collations > Rules for Combining Character Sets and Collations" in *M Compatibility Developer Guide*. These features are different from those in MySQL.

3.2.8 Drivers

3.2.8.1 ODBC

3.2.8.1.1 ODBC API Reference

Obtaining Parameter Description

SQLDescribeParam is a function in the ODBC API. It is used to obtain the description of parameters related to prepared SQL statements (for example, calling SQLPrepare). It can return metadata such as the type, size, and whether **NULL** values are allowed for parameters, which is useful for dynamically building SQL statements and binding parameters.

Prototype

```
SQLRETURN SQLDescribeParam(
    SQLHSTMT        StatementHandle,
    SQLUSMALLINT    ParameterNumber,
    SQLSMALLINT    *DataTypePtr,
    SQLULEN        *ParameterSizePtr,
    SQLSMALLINT    *DecimalDigitsPtr,
    SQLSMALLINT    *NullablePtr);
```

Table 3-35 Parameters of SQLDescribeParam

Parameter	Description	Difference
StatementHandle	Statement handle.	-
ParameterNumber	Parameter marker number, starting with 1 and increasing in ascending order.	-
DataTypePtr	Points to the data type of the returned parameter.	In MySQL, ODBC returns **SQL_VARCHAR** for any type. In GaussDB, ODBC returns the data type to an application based on that returned by the kernel.
ParameterSizePtr	Points to the size of the returned parameter.	If MySQL allows the ODBC driver to use a larger data packet for data transmission, **24M** is returned. Otherwise, **255** is returned. In GaussDB, ODBC returns the parameter size based on the actual type.
DecimalDigitsPtr	Points to the number of decimal digits of the returned parameter.	-

Parameter	Description	Difference
NullablePtr	Points to whether **NULL** values are allowed for the returned parameter.	In MySQL, ODBC directly returns **SQL_NULLABLE_UNKNOWN**. In GaussDB, ODBC directly returns **SQL_NULLABLE**.

3.2.8.2 JDBC

JDBC can convert the data types supported by the database to the standard data types of JDBC. For details about the supported data types, see the set and get APIs in "Application Development Guide > Development Based on JDBC > JDBC API Reference" in *Developer Guide*.

NOTE

If the time() type used in JDBC includes a precision, for example, time(6), the precision is precisely retained, which is the same as that in MySQL 8.0.

3.3 MySQL-compatible Mode

3.3.1 Data Types

3.3.1.1 Numeric Data Types

Integer types

Unless otherwise specified, the precision, scale, and number of digits cannot be set to the floating-point values in MySQL-compatible mode by default. You are advised to use valid integer values.

Differences in terms of the integer types:

- Input format:
  - MySQL

    For characters such as "asbd", "12dd", and "12 12", the system truncates them or returns 0 and reports a WARNING. Data fails to be inserted into a table in strict mode.
  - GaussDB
    - For integer types (TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, and BIGINT), if the invalid part of a character string is truncated, for example, "12@3", no message is displayed. Data is successfully inserted into a table.

    - If the whole integer is truncated (for example, "@123") or the character string is empty, 0 is returned and data is successfully inserted into a table.

- Operators:
  - +, -, and *

    GaussDB: When INT, INTEGER, SMALLINT, or BIGINT is used for calculation, a value of the original type is returned and is not changed to a larger type. If the return value exceeds the range, an error is reported.

    MySQL: The value can be changed to BIGINT for calculation.
  - |, &, ^, and ~

    GaussDB: The value is calculated in the bits occupied by the type. In GaussDB, ^ indicates the exponentiation operation. If the XOR operator is required, replace it with #.

    MYSQL: The value is changed to a larger type for calculation.
- Type conversion of negative numbers:

  GaussDB: The result is **0** in loose mode and an error is reported in strict mode.

  MySQL: The most significant bit is replaced with a numeric bit based on the corresponding binary value, for example, (-1)::uint4 = 4294967295.
- Other differences:

  The precision of INT[(M)] controls formatted output in MySQL. GaussDB supports only the syntax but does not support the function.
- Aggregate function:
  - variance: indicates the sample variance in GaussDB and the population variance in MySQL.
  - stddev: indicates the sample standard deviation in GaussDB and the overall standard deviation in MySQL.
- Display width:
  - If **ZEROFILL** is not specified when the width information is specified for an integer column, the width information is not displayed in the table structure description.
  - When the INSERT statement is used to insert a column of the character type, GaussDB pads 0s before inserting the column.
  - The JOIN USING statement involves type derivation. In MySQL, the first table column is used by default. In GaussDB, if the result is of the signed type, the width information is invalid. Otherwise, the width of the first table column is used.
  - For GREATEST/LEAST, IFNULL/IF, and CASE WHEN/DECODE, MySQL does not pad 0s. In GaussDB, 0s are padded when the type and width information is consistent.
  - MySQL supports this function when it is used as the input or output parameter or return value of a function or stored procedure. GaussDB neither reports syntax errors nor supports this function.

For details about the differences in GaussDB and MySQL in terms of the integer types, see **Table 3-36**.

Table 3-36 Integer types

MySQL	GaussDB	Difference
BOOL	Supported, with differences	MySQL: The BOOL/BOOLEAN type is actually mapped to the TINYINT type. GaussDB: BOOL is supported. • Valid literal values for the "true" state include: **TRUE**, **'t'**, **'true'**, **'y'**, **'yes'**, **'1'**, **'TRUE'**, **true**, **'on'**, and all non-zero values. • Valid literal values for the "false" state include: **FALSE**, **'f'**, **'false'**, **'n'**, **'no'**, **'0'**, **0**, **'FALSE'**, **false**, and **'off'**. **TRUE** and **FALSE** are standard expressions, compatible with SQL statements.
BOOLEAN	Supported, with differences	
TINYINT[(M)] [UNSIGNED]	Supported, with differences	For details, see **Differences in terms of the integer types**.
SMALLINT[(M)] [UNSIGNED]	Supported, with differences	For details, see **Differences in terms of the integer types**.
MEDIUMINT[(M)] [UNSIGNED]	Supported, with differences	MySQL requires 3 bytes to store MEDIUMINT data. • The signed range is –8388608 to +8388607. • The unsigned range is 0 to +16777215. GaussDB maps data to the INT type and requires 4 bytes for storage. • The signed range is –2147483648 to +2147483647. • The unsigned range is 0 to +4294967295. For details about other differences, see **Differences in terms of the integer types**.
INT[(M)] [UNSIGNED]	Supported, with differences	For details, see **Differences in terms of the integer types**.
INTEGER[(M)] [UNSIGNED]	Supported, with differences	For details, see **Differences in terms of the integer types**.
BIGINT[(M)] [UNSIGNED]	Supported, with differences	For details, see **Differences in terms of the integer types**.

Arbitrary precision types

Table 3-37 Arbitrary precision types

MySQL	GaussDB	Difference
DECIMAL[(M[,D])]	Supported, with differences	• Operator: In GaussDB, "^" indicates the exponentiation operation. If the XOR operator is required, replace it with "#". In MySQL, "^" indicates the XOR operation. • Value range: The precision **M** and scale **D** support only integers and do not support floating-point values. • Input format: No error is reported when all input parameters of a character string (for example, "@123") are truncated. An error is reported only when it is partially truncated, for example, "12@3".
NUMERIC[(M[,D])]	Supported, with differences	
DEC[(M[,D])]	Supported, with differences	
FIXED[(M[,D])]	Not supported	-

Floating-point types

Table 3-38 Floating-point types

MySQL	GaussDB	Difference
FLOAT[(M,D)]	Supported, with differences	• Partitioned table: The FLOAT data type does not support partitioned tables with the key partitioning policy. • Operator: In GaussDB, "^" indicates the exponentiation operation. If the XOR operator is required, replace it with "#". In MySQL, "^" indicates the XOR operation. • Value range: The precision **M** and scale **D** support only integers and do not support floating-point values. • Output format: An ERROR message is reported for invalid input parameters. No WARNING is reported in loose mode (that is, **sql_mode** is set to **''**).

MySQL	GaussDB	Difference
FLOAT(p)	Supported, with differences	• Partitioned table: The FLOAT data type does not support partitioned tables with the key partitioning policy. • Operator: The ^ operator is used for the numeric types, which is different from that in MySQL. In GaussDB, the ^ operator is used for exponential calculation. • Value range: When the precision **p** is defined, only valid integer data types are supported. • Output format: An ERROR message is reported for invalid input parameters. No WARNING is reported in loose mode (that is, **sql_mode** is set to **''**).
DOUBLE[(M, D)]	Supported, with differences	• Partitioned table: The DOUBLE data type does not support partitioned tables with the key partitioning policy. • Operator: In GaussDB, "^" indicates the exponentiation operation. If the XOR operator is required, replace it with "#". In MySQL, "^" indicates the XOR operation. • Value range: The precision **M** and scale **D** support only integers and do not support floating-point values. • Output format: An ERROR message is reported for invalid input parameters. No WARNING is reported in loose mode (that is, **sql_mode** is set to **''**).
DOUBLE PRECISION[(M,D)]	Supported, with differences	• Operator: In GaussDB, "^" indicates the exponentiation operation. If the XOR operator is required, replace it with "#". In MySQL, "^" indicates the XOR operation. • Value range: The precision **M** and scale **D** support only integers and do not support floating-point values. • Output format: An ERROR message is reported for invalid input parameters. No WARNING is reported in loose mode (that is, **sql_mode** is set to **''**).

MySQL	GaussDB	Difference
REAL[(M,D)]	Supported, with differences	• Partitioned table: The REAL data type does not support partitioned tables with the key partitioning policy. • Operator: In GaussDB, "^" indicates the exponentiation operation. If the XOR operator is required, replace it with "#". In MySQL, "^" indicates the XOR operation. • Value range: The precision **M** and scale **D** support only integers and do not support floating-point values. • Output format: An ERROR message is reported for invalid input parameters. No WARNING is reported in loose mode (that is, **sql_mode** is set to **''**).

Sequential Integers

Table 3-39 Sequential integers

MySQL	GaussDB	Difference
SERIAL	Supported, with differences	For details about SERIAL in GaussDB, see "SQL Reference > Data Types > Value Types" in *Developer Guide*. The differences in specifications are as follows: `CREATE TABLE test(f1 serial, f2 CHAR(20));` • Difference in type definition: MySQL maps serial to BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE, and GaussDB maps serial to INTEGER NOT NULL DEFAULT nextval('test_f1_seq'::regclass). For example: `-- Definition of MySQL SERIAL:` `mysql> SHOW CREATE TABLE test\G` `*************************** 1. row ***************************` `Table: test` `Create Table: CREATE TABLE `test` (` `` `f1` bigint(20) unsigned NOT NULL AUTO_INCREMENT, `` `` `f2` char(20) DEFAULT NULL, `` `` UNIQUE KEY `f1` (`f1`) `` `) ENGINE=InnoDB DEFAULT CHARSET=utf8` `1 row in set (0.00 sec)` `-- Definition of GaussDB SERIAL` `gaussdb=# \d+ test` `Table "public.test"` `Column	Type	Modifiers	Storage	Stats target	Description` `--------+---------- +----------+-------------+-------------` `f1	integer	not null default nextval('test_f1_seq'::regclass)	plain	|` `f2	character(20)		extended	|` `Has OIDs: no` `Options: orientation=row, compression=no, storage_type=USTORE` • Differences in using INSERT to insert default values of the SERIAL type. For example: `-- Inserting default values of the SERIAL type in MySQL` `mysql> INSERT INTO test VALUES(DEFAULT, 'aaaa');` `Query OK, 1 row affected (0.00 sec)`  `mysql> INSERT INTO test VALUES(10, 'aaaa');` `Query OK, 1 row affected (0.00 sec)`  `mysql> INSERT INTO test VALUES(DEFAULT, 'aaaa');` `Query OK, 1 row affected (0.00 sec)`  `mysql> SELECT * FROM test;` `+----+------+` `| f1	f2 |` `+----+------+` `| 1	aaaa |` `| 10	aaaa |` `| 11	aaaa |`

MySQL	GaussDB	Difference

		+----+------+
		3 rows in set (0.00 sec)
		
		-- Inserting default values of the SERIAL type in GaussDB
		gaussdb=# INSERT INTO test VALUES(DEFAULT, 'aaaa');
		INSERT 0 1
		gaussdb=# INSERT INTO test VALUES(10, 'aaaa');
		INSERT 0 1
		gaussdb=# INSERT INTO test VALUES(DEFAULT, 'aaaa');
		INSERT 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2
		----+----------------------
		1 | aaaa
		2 | aaaa
		10 | aaaa
		(3 rows)
		• Differences in performing REPLACE on referencing columns of the SERIAL type. For details about GaussDB referencing columns, see "SQL Reference > SQL Syntax > R > REPLACE" in *Developer Guide*. For example:
		-- Inserting values of the referencing columns of the SERIAL type in MySQL
		mysql> REPLACE INTO test VALUES(f1, 'aaaa');
		Query OK, 1 row affected (0.00 sec)
		
		mysql> REPLACE INTO test VALUES(f1, 'bbbb');
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM test;
		+----+------+
		| f1 | f2 |
		+----+------+
		| 1 | aaaa |
		| 2 | bbbb |
		+----+------+
		2 rows in set (0.00 sec)
		
		-- Inserting values of the referencing columns of the SERIAL type in GaussDB
		gaussdb=# REPLACE INTO test VALUES(f1, 'aaaa');
		REPLACE 0 1
		gaussdb=# REPLACE INTO test VALUES(f1, 'bbbb');
		REPLACE 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2
		----+----------------------
		0 | aaaa
		0 | bbbb
		(2 rows)

3.3.1.2 Date and Time Data Types

Table 3-40 Date and Time Data Types

MySQL	GaussDB	Difference
DATE	Supported, with differences	GaussDB supports the date data type. Compared with MySQL, GaussDB has the following differences in specifications: ● Input format – GaussDB supports only the character type and does not support the numeric type. For example, the format can be '2020-01-01' or '20200101', but cannot be 20200101. MySQL supports conversion from numeric input to the date type. – Separator: GaussDB does not support the plus sign (+) or colon (:) as the separator between the year, month, and day. Other symbols are supported. MySQL supports all symbols as separators. Sometimes, the mixed use of separators is not supported, which is different from MySQL, such as '2020-01>01' and '2020/01+01'. You are advised to use hyphens (-) or slashes (/) as separators. – No separator: You are advised to use the complete format, for example, 'YYYYMMDD' or 'YYMMDD'. The parsing rules of incomplete formats (including the ultra-long format) are different from those of MySQL. An error may be reported or the parsing result may be inconsistent with that of MySQL. Therefore, the incomplete format is not recommended. ● Output format If the **sql_mode** parameter of GaussDB does not contain **'strict_trans_tables'** (the strict mode is used unless otherwise defined as the loose mode), the year, month, and day can be set to **0**. However, the value is converted to a valid value in the sequence of year, month, and day. For example, date '0000-00-10' is converted to 0002-12-10 BC. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00** is returned. MySQL outputs the date value as it is, even if the year, month, and day are set to **0**. ● Value range

MySQL	GaussDB	Difference

		The value range of GaussDB is 4713-01-01 BC to 5874897-12-31 AD. BC dates are supported. In loose mode, if the value exceeds the range, **0000-00-00** is returned. In strict mode, an error is reported. In MySQL, the value range is 0000-00-00 to 9999-12-31. In loose mode, if the value exceeds the range, the performance varies in different scenarios. An error may be reported (for example, in the SELECT statement) or the value **0000-00-00** may be returned (for example, in the INSERT statement). As a result, when the date type is used as the input parameter of the function, the results returned by the function are different.
		● Operator
		– GaussDB supports only the comparison operators =, !=, <, <=, >, and >= between date types and returns **true** or **false**. For the addition operation between the date and interval types, the return result is of the date type. For the subtraction operation between the date and interval types, the return result is of the date type. For the subtraction operation between date types, the return result is of the interval type.
		– When the MySQL date type and other numeric types are calculated, the date type is converted to the numeric type, and then the calculation is performed based on the numeric type. The result is also of the numeric type. It is different from GaussDB. For example:
		`-- MySQL: date+numeric. Convert the date type to 20200101 and add it to 1. The result is 20200102.`
		`mysql> SELECT date'2020-01-01' + 1;`
		`+----------------------+`
		`| date'2020-01-01' + 1 |`
		`+----------------------+`
		`| 20200102 |`
		`+----------------------+`
		`1 row in set (0.00 sec)`
		
		`-- GaussDB: date+numeric. Convert the numeric type to the interval type (1 day), and then add them up to obtain a new date.`
		`gaussdb=# SELECT date'2020-01-01' + 1;`
		`?column?`
		`------------`
		`2020-01-02`
		`(1 row)`
		● Type conversion

MySQL	GaussDB	Difference
		Compared with MySQL, GaussDB supports conversion between the date type and char(n), nchar(n), datetime, or timestamp type, but does not support conversion between the date type and binary, decimal, JSON, integer, unsigned integer, or time type. The principles for determining common types in scenarios such as collections and complex expressions are different from those in MySQL. For details, see **Data Type Conversion**.

MySQL	GaussDB	Difference
DATETIME[(fsp)]	Supported, with differences.	GaussDB supports the datetime data type. Compared with MySQL, GaussDB has the following differences in specifications: ● Input format – GaussDB supports only the character type and does not support the numeric type. For example, '2020-01-01 10:20:30.123456' or '20200101102030.123456' is supported, but 20200101102030.123456 is not supported. MySQL supports conversion from numeric input to the datetime type. – Separator: GaussDB does not support the plus sign (+) or colon (:) as the separator between the year, month, and day. Other symbols are supported. Only colons (:) can be used as separators between hours, minutes, and seconds. Sometimes, the mixed use of separators is not supported, which is different from MySQL. Therefore, it is not recommended. MySQL supports all symbols as separators. – No separator: In GaussDB, the complete format 'YYYYMMDDhhmiss.ffffff' is recommended. The parsing rules of incomplete formats (including the ultra-long format) may be different from those of MySQL. An error may be reported or the parsing result may be inconsistent with that of MySQL. Therefore, the incomplete format is not recommended. ● Output format: – The format is 'YYYY-MM-DD hh:mi:ss.ffffff', which is the same as that of MySQL and is not affected by the **DateStyle** parameter. However, for the precision part, if the last several digits are 0, they are not displayed in GaussDB but displayed in MySQL. – If the **sql_mode** parameter of GaussDB does not contain **'strict_trans_tables'** (the strict mode is used unless otherwise defined as the loose mode), the year, month, and day can be set to **0**. However, the value is converted to a valid value in the sequence of year, month, and day. For example, datetime '0000-00-10 00:00:00'

MySQL	GaussDB	Difference
		is converted to 0002-12-10 00:00:00 BC. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00 00:00:00** is returned. MySQL outputs the datetime value as it is, even if the year, month, and day are set to **0**. ● Value range 4713-11-24 00:00:00.000000 BC to 294277-01-09 04:00:54.775806 AD. If the value is **294277-01-09 04:00:54.775807 AD**, **infinity** is returned. If the value exceeds the range, GaussDB reports an error in strict mode. Whether MySQL reports an error depends on the application scenario. Generally, no error is reported in the query scenario. However, an error is reported when the DML or SQL statement is executed to change the value of a table attribute. In loose mode, GaussDB returns **0000-00-00 00:00:00**. MySQL may report an error, return **0000-00-00 00:00:00**, or return **null** based on the application scenario. As a result, the execution result of the function that uses the datetime type as the input parameter is different from that of MySQL. ● Precision The value ranges from 0 to 6. For a table column, the default value is **0**, which is the same as that in MySQL. In the datetime[(p)]'str' expression, GaussDB parses **(p)** as the precision. The default value is **6**, indicating that **'str'** is formatted to the datetime type based on the precision specified by **p**. MySQL does not support the datetime[(p)]'str' expression. ● Operator – GaussDB supports only the comparison operators =, !=, <, <=, >, and >= between datetime types and returns **true** or **false**. For the addition operation between the datetime and interval types, the return result is of the datetime type. For the subtraction operation between the datetime and interval types, the return result is of the datetime type. For the subtraction operation between datetime types, the return result is of the interval type.

MySQL	GaussDB	Difference

		– When the MySQL datetime type and other numeric types are calculated, the datetime type is converted to the numeric type, and then the calculation is performed based on the numeric type. The result is also of the numeric type. It is different from GaussDB. For example:
		-- MySQL: datetime+numeric. Convert the datetime type to 20201010123456 and add it to 1. The result is 20201010123457.
		mysql> SELECT cast('2020-10-10 12:34:56.123456' AS datetime) + 1;
		+--+
		| cast('2020-10-10 12:34:56.123456' as datetime) + 1 |
		+--+
		| 20201010123457 |
		+--+
		1 row in set (0.00 sec)
		
		-- GaussDB: datetime+numeric. Convert the numeric type to the interval type (1 day), and then add them up to obtain the new datetime.
		gaussdb=# SELECT cast('2020-10-10 12:34:56.123456' AS datetime) + 1;
		?column?
		--------------------- 	
		2020-10-11 12:34:56
		(1 row)
		If the calculation result of the datetime type and numeric type is used as the input parameter of a function, the result of the function may be different from that of MySQL.
		• Type conversion
		Compared with MySQL, GaussDB supports only the conversion between the datetime type and the char(n), varchar(n), or timestamp type, and the conversion from datetime to date or time (only value assignment and explicit conversion). Conversion between binary, decimal, json, integer, and unsigned integer types is not supported. The principles for determining common types in scenarios such as collections and complex expressions are different from those in MySQL. For details, see **Data Type Conversion**.
		• Time zone
		In GaussDB, the datetime value can carry the time zone information (time zone offset or time zone name), for example, '2020-01-01 12:34:56.123456 +01:00' or '2020-01-01 2:34:56.123456 CST'. GaussDB converts the time to the time of the current server time

MySQL	GaussDB	Difference
		zone. MySQL 5.7 does not support this function. MySQL 8.0 and later versions support this function. • The table columns of the datetime data type in GaussDB are actually converted to the timestamp(p) without time zone. When you query the table information or use a tool to export the table structure, the data type of columns is timestamp(p) without time zone instead of datetime. For MySQL, datetime(p) is displayed.

MySQL	GaussDB	Difference
TIMESTAMP[(fsp)]	Supported, with differences	GaussDB supports the timestamp data type and differs from MySQL in terms of the following specifications: ● Input format: – It supports only the character type and does not support the numeric type. For example, '2020-01-01 10:20:30.123456' or '20200101102030.123456' is supported, but 20200101102030.123456 is not supported. MySQL supports conversion from numeric input to the timestamp type. – Separator: It does not support the plus sign (+) or colon (:) as the separator between the year, month, and day. Other symbols are supported. Only colons (:) can be used as separators between hours, minutes, and seconds. Sometimes, the mixed use of separators is not supported, which is different from MySQL. Therefore, it is not recommended. MySQL supports all symbols as separators. – No separator: The complete format 'YYYYMMDDhhmiss.ffffff' is recommended. The parsing rules of incomplete formats (including the ultra-long format) may be different from those of MySQL. An error may be reported or the parsing result may be inconsistent with that of MySQL. Therefore, the incomplete format is not recommended. ● Output format: – The format is 'YYYY-MM-DD hh:mi:ss.ffffff', which is the same as that of MySQL and is not affected by the **DateStyle** parameter. However, for the precision part, if the last several digits are 0, they are not displayed in GaussDB but displayed in MySQL. – If the **sql_mode** parameter of GaussDB does not contain **'strict_trans_tables'** (the strict mode is used unless otherwise defined as the loose mode), the year, month, and day can be set to **0**. However, the value is converted to a valid value in the sequence of year, month, and day. For example, timestamp '0000-00-10

MySQL	GaussDB	Difference
		00:00:00' is converted to 0002-12-10 00:00:00 BC. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00 00:00:00** is returned. MySQL outputs the timestamp value as it is, even if the year, month, and day are set to **0**. ● Value range: 4713-11-24 00:00:00.000000 BC to 294277-01-09 04:00:54.775806 AD. If the value is **294277-01-09 04:00:54.775807 AD**, **infinity** is returned. If the value exceeds the range, GaussDB reports an error in strict mode. Whether MySQL reports an error depends on the application scenario. Generally, no error is reported in the query scenario. However, an error is reported when the DML or SQL statement is executed to change the value of a table attribute. In loose mode, GaussDB returns **0000-00-00 00:00:00**. MySQL may report an error, return **0000-00-00 00:00:00**, or return **null** based on the application scenario. As a result, the execution result of the function that uses the timestamp type as the input parameter is different from that of MySQL. ● Precision: The value ranges from 0 to 6. For a table column, the default value is **0**, which is the same as that in MySQL. In the timestamp[(p)] 'str' expression: – GaussDB parses **(p)** as the precision. The default value is **6**, indicating that **'str'** is formatted to the timestamp type based on the precision specified by **p**. – The meaning of timestamp 'str' in MySQL is the same as that in GaussDB. The default precision is **6**. However, timestamp(p) 'str' is parsed as a function call. **p** is used as the input parameter of the timestamp function. The result returns a value of the timestamp type, and **'str'** is used as the alias of the projection column. ● Operators: – GaussDB supports only the comparison operators =, !=, <, <=, >, and >= between timestamp types and returns **true** or **false**. For the addition operation between

MySQL	GaussDB	Difference

		the timestamp and interval types, the return result is of the timestamp type. For the subtraction operation between the timestamp and interval types, the return result is of the timestamp type. For the subtraction operation between timestamp types, the return result is of the interval type.
		– When the MySQL timestamp type and other numeric types are calculated, the timestamp type is converted to the numeric type, and then the calculation is performed based on the numeric type. The result is also of the numeric type. It is different from GaussDB. For example:
		`-- MySQL: timestamp+numeric. Convert the timestamp type to 20201010123456.123456 and add it to 1. The result is 20201010123457.123456.`
		`mysql> SELECT timestamp '2020-10-10 12:34:56.123456' + 1;`
		`+-------------------------------------------+`
		`| timestamp '2020-10-10 12:34:56.123456' + 1 |`
		`+-------------------------------------------+`
		`| 20201010123457.123456 |`
		`+-------------------------------------------+`
		`1 row in set (0.00 sec)`
		
		`-- GaussDB: timestamp+numeric. Convert the numeric type to the interval type (1 day), and then add them up to obtain a new timestamp.`
		`gaussdb=# SELECT timestamp '2020-10-10 12:34:56.123456' + 1;`
		`?column?`
		`---------------------------`
		`2020-10-11 12:34:56.123456`
		`(1 row)`
		If the calculation result of the timestamp type and numeric type is used as the input parameter of a function, the result of the function may be different from that of MySQL.
		• Type conversion:
		Compared with MySQL, GaussDB supports only the conversion between the timestamp type and the char(n), varchar(n), or datetime type, and the conversion from timestamp to date or time (only value assignment and explicit conversion). Conversion between binary, decimal, json, integer, and unsigned integer types is not supported. The principles for determining common types in scenarios such as collections and complex expressions are different from those in MySQL. For details, see **Data Type Conversion**.

MySQL	GaussDB	Difference
		• Time zone In GaussDB, the timestamp value can carry the time zone information (time zone offset or time zone name), for example, '2020-01-01 12:34:56.123456 +01:00' or '2020-01-01 2:34:56.123456 CST'. GaussDB converts the time to the time of the current server time zone. If the time zone of the server is changed, the timestamp value is converted to the timestamp of the new time zone. MySQL 5.7 does not support this function. MySQL 8.0 and later versions support this function. • The table columns of the timestamp data type in GaussDB are actually converted to the timestamp(p) with time zone. When you query the table information or use a tool to export the table structure, the data type of columns is timestamp(p) with time zone instead of timestamp. For MySQL, timestamp(p) is displayed.

MySQL	GaussDB	Difference
TIME[(fsp)]	Supported, with differences	GaussDB supports the time data type and differs from MySQL in terms of the following specifications: ● Input format: – It supports only the character type and does not support the numeric type. For example, '1 10:20:30' or '102030' is supported, but 102030 is not supported. MySQL supports conversion from numeric input to the time type. – Separator: GaussDB supports only colons (:) as separators between hours, minutes, and seconds. MySQL supports all symbols as separators. – No separator: The complete format 'hhmiss.ffffff' is recommended. The parsing rules of incomplete formats (including the ultra-long format) may be different from those of MySQL. An error may be reported or the parsing result may be inconsistent with that of MySQL. Therefore, the incomplete format is not recommended. – When a negative value is entered for minute, second, or precision, GaussDB may ignore the first part of the negative value, which is parsed as 0. For example, '00:00:-10' is parsed as '00:00:00'. An error may also be reported. For example, if '00:00:-10000' is parsed, an error will be reported. The result depends on the range of the input value. However, MySQL reports an error in both cases. ● Output format: The format is hh:mi:ss.ffffff, which is the same as that of MySQL. However, for the precision part, if the last several digits are 0, they are not displayed in GaussDB but displayed in MySQL. ● Value range: –838:59:59.000000 to 838:59:59.000000, which is the same as that of MySQL. For values that exceed the range, when GaussDB performs DML operations such as SELECT, INSERT, and UPDATE in loose mode, it returns the nearest boundary values such as **-838:59:59** or **838:59:59**. In MySQL, an error is reported during query, or the nearest

MySQL	GaussDB	Difference

		boundary value is returned after a DML operation. As a result, when the time type is used as the input parameter of the function, the results returned by the function are different.
		● Precision:
		The value ranges from 0 to 6. For a table column, the default value is **0**, which is the same as that in MySQL. In the time(p) 'str' expression, GaussDB parses **(p)** as the precision. The default value is **6**, indicating that **'str'** is formatted to the time type based on the precision specified by **p**. MySQL parses it as a time function, **p** is an input parameter, and **'str'** is the alias of the projection column.
		● Operators:
		– GaussDB supports only the comparison operators =, !=, <, <=, >, and >= between time types and returns **true** or **false**. For the addition operation between the time and interval types, the return result is of the time type. For the subtraction operation between the time and interval types, the return result is of the time type. For the subtraction operation between time types, the return result is of the interval type.
		– When the MySQL time type and other numeric types are calculated, the time type is converted to the numeric type, and then the calculation is performed based on the numeric type. The result is also of the numeric type. It is different from GaussDB. For example:
		`-- MySQL: time+numeric. Convert the time type to 123456 and add it to 1. The result is 123457.`
		`mysql> SELECT time '12:34:56' + 1;`
		`+----------------------+`
		`| time '12:34:56' + 1 |`
		`+----------------------+`
		`| 123457 |`
		`+----------------------+`
		`1 row in set (0.00 sec)`
		
		`-- GaussDB: time+numeric. Convert the numeric type to the interval type (1 day), and then add them up to obtain the new time. Because 24 hours are added, the obtained time is still 12:34:56.`
		`gaussdb=# SELECT time '12:34:56' + 1;`
		` ?column?`
		`----------`

MySQL	GaussDB	Difference
		12:34:56 (1 row) If the calculation result of the time type and numeric type is used as the input parameter of a function, the result of the function may be different from that of MySQL. • Type conversion: Compared with MySQL, GaussDB supports only the conversion between the time type and the char(n) or nchar(n) type, and the conversion from datetime or timestamp to the time type. The conversion between the binary, decimal, date, json, integer, and unsigned integer types is not supported. The principles for determining common types in scenarios such as collections and complex expressions are different from those in MySQL. For details, see **Data Type Conversion**.
YEAR[(4)]	Supported, with differences	GaussDB supports the year data type and differs from MySQL in terms of the following specifications: • Operators: – GaussDB supports only the comparison operators =, !=, <, <=, >, and >= between year types and returns **true** or **false**. – GaussDB supports only the arithmetic operators + and - between the year and int4 types and returns integer values. MySQL returns unsigned integer values. • Type conversion: Compared with MySQL, GaussDB supports only the conversion between the year type and int4 type, and supports only the conversion from the int4, varchar, numeric, date, time, timestamp, or timestamptz type to the year type. The principles for determining common types in scenarios such as collections and complex expressions are different from those in MySQL. For details, see **Data Type Conversion**.

MySQL	GaussDB	Difference
INTERVAL	Supported, with differences	GaussDB supports the INTERVAL data type, but INTERVAL is an expression in MySQL. The differences are as follows: • The date input of the character string type cannot be used as an operation, for example, SELECT '2023-01-01' + interval 1 day. • In the INTERVAL expr unit syntax, **expr** cannot be a negative integer or floating-point number, for example, SELECT date'2023-01-01' + interval -1 day. • In the INTERVAL expr unit syntax, **expr** cannot be the input of an operation expression, for example, SELECT date'2023-01-01' + interval 4/2 day. • When the INTERVAL expression is used for calculation, the return value is of the datetime type. For MySQL, the return value is of the datetime or date type. The calculation logic is the same as that of GaussDB but different from that of MySQL. • In the INTERVAL expr unit syntax, the value range of **expr** varies with the unit. The maximum value range is [–2147483648, 2147483647]. If the value exceeds the range, an error is reported in strict mode, and a warning is reported in loose mode and **0** is returned. • In the INTERVAL expr unit syntax, if the number of columns specified by **expr** is greater than the expected number of columns in unit, an error is reported in strict mode, and a warning is reported in loose mode and **0** is returned. For example, if the value of **unit** is **DAY_HOUR**, the expected number of columns is 2. If the value of **expr** is **'1-2-3'**, the expected number of columns is 3.

3.3.1.3 String Data Types

Table 3-41 String data types

MySQL	GaussDB	Difference
CHAR[(M)]	Supported, with differences.	• Input format – The length of parameters and return values of GaussDB user-defined functions cannot be verified. The length of stored procedure parameters cannot be verified. In addition, correct spaces cannot be supplemented when **PAD_CHAR_TO_FULL_LENGTH** is enabled. However, MySQL supports these functions. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Syntax The CAST(expr as char) syntax of GaussDB cannot convert the input string to the corresponding type based on the string length. It can only be converted to the varchar type. CAST('' as char) and CAST('' as char(0)) cannot convert an empty string to the char(0) type. MySQL supports conversion to the corresponding type by length. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.

MySQL	GaussDB	Difference
VARCHAR(M)	Supported, with differences.	• Input format – The length of parameters and return values of GaussDB user-defined functions cannot be verified. The length of stored procedure parameters cannot be verified. However, MySQL supports these functions. – The length of temporary variables in GaussDB user-defined functions and stored procedures can be verified, and an error or truncation alarm is reported in strict or loose mode. However, MySQL does not support these functions. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.

MySQL	GaussDB	Difference
TINYTEXT	Supported, with differences.	• Input format – In GaussDB, the length of this type cannot exceed 1 GB. If the length exceeds 1 GB, an error is reported. In MySQL, the length of this type cannot exceed 255 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.

MySQL	GaussDB	Difference
TEXT	Supported, with differences.	• Input format – In GaussDB, the length of this type cannot exceed 1 GB. If the length exceeds 1 GB, an error is reported. In MySQL, the length of this type cannot exceed 65535 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.

MySQL	GaussDB	Difference
MEDIUMTEXT	Supported, with differences.	• Input format – In GaussDB, the length of this type cannot exceed 1 GB. If the length exceeds 1 GB, an error is reported. In MySQL, the length of this type cannot exceed 16777215 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.

MySQL	GaussDB	Difference
LONGTEXT	Supported, with differences.	• Input format – GaussDB supports a maximum of 1 GB, and MySQL supports a maximum of 4 GB minus 1 byte. – GaussDB does not support escape characters or double quotation marks (""). MySQL supports these inputs. • Operator – After performing addition, subtraction, multiplication, division, or modulo operations on a string (that can be converted to a floating-point value) and an integer value, GaussDB returns an integer, while MySQL returns a floating-point value. – If a value is divided by 0, GaussDB reports an error, and MySQL returns **null**. – "~": returns a negative number in GaussDB and an 8-byte unsigned integer in MySQL. – "^": indicates a power in GaussDB and a bitwise XOR in MySQL.
ENUM('value1','value2',...)	Not supported	-
SET('value1','value2',...)	Not supported	-

3.3.1.4 Binary Data Types

Table 3-42 Binary data types

MySQL	GaussDB	Difference
BINARY[(M)]	Not supported	-
VARBINARY(M)	Not supported	-

MySQL	GaussDB	Difference
TINYBLOB	Supported, with differences.	• Value range: In GaussDB, this type is mapped from the BYTEA type. Its length cannot exceed 1 GB. Otherwise, an error is reported. In MySQL, the length of this type cannot exceed 255 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. • Input format: Escape characters and double quotation marks ("") are not supported. • Output format: For the '\0' character, the query result is displayed as "\000". If the getBytes API of the JDBC driver is used, the result is the '\0' character. • Operator: Arithmetic operators (+ - * / %) are not supported. Common logical operators OR, AND, NOT (|| && !) are not supported. Common bitwise operators (~ & | ^) are not supported.
BLOB	Supported, with differences.	• Value range: In GaussDB, this type is mapped from the BYTEA type. Its length cannot exceed 1 GB. Otherwise, an error is reported. In MySQL, the length of this type cannot exceed 65535 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. • Input format: Escape characters and double quotation marks ("") are not supported. • Output format: For the '\0' character, the query result is displayed as "\000". If the getBytes API of the JDBC driver is used, the result is the '\0' character. • Operator: Arithmetic operators (+ - * / %) are not supported. Common logical operators OR, AND, NOT (|| && !) are not supported. Common bitwise operators (~ & | ^) are not supported.

MySQL	GaussDB	Difference
MEDIUMBLOB	Supported, with differences.	• Value range: In GaussDB, this type is mapped from the BYTEA type. Its length cannot exceed 1 GB. Otherwise, an error is reported. In MySQL, the length of this type cannot exceed 16777215 bytes. Otherwise, an error is reported in strict mode, and data is truncated and an alarm is generated in loose mode. • Input format: Escape characters and double quotation marks ("") are not supported. • Output format: For the '\0' character, the query result is displayed as "\000". If the getBytes API of the JDBC driver is used, the result is the '\0' character. • Operator: Arithmetic operators (+ - * / %) are not supported. Common logical operators OR, AND, NOT (|| && !) are not supported. Common bitwise operators (~ & | ^) are not supported.
LONGBLOB	Supported, with differences.	• Value range: In GaussDB, this type is mapped from the BYTEA type. Its length cannot exceed 1 GB. For details, see the centralized and distributed specifications of the BYTEA data type. • Input format: Escape characters and double quotation marks ("") are not supported. • Output format: For the '\0' character, the query result is displayed as "\000". If the getBytes API of the JDBC driver is used, the result is the '\0' character. • Operator: Arithmetic operators (+ - * / %) are not supported. Common logical operators OR, AND, NOT (|| && !) are not supported. Common bitwise operators (~ & | ^) are not supported.
BIT[(M)]	Not supported	-

3.3.1.5 JSON Data Type

Table 3-43 JSON Data Type

MySQL	GaussDB	Difference
JSON	Supported, with differences.	• The JSON types in GaussDB in MySQL-compatible mode are the same as the native JSON type of GaussDB but greatly different from that of MySQL. Therefore, the JSON types are not listed one by one. • For details about the JSON types in GaussDB in MySQL-compatible mode, see "SQL Reference > Data Types > JSON/JSONB Types" in *Developer Guide*.

3.3.1.6 Attributes Supported by Data Types

Table 3-44 Attributes supported by data types

MySQL	GaussDB
NULL	Supported.
NOT NULL	Supported.
DEFAULT	Supported.
ON UPDATE	Supported.
PRIMARY KEY	Supported.
CHARACTER SET name	Supported.
COLLATE name	Supported.

3.3.1.7 Data Type Conversion

Conversion between different data types is supported. Data type conversion is involved in the following scenarios:

- The data types of operands of operators (such as comparison and arithmetic operators) are inconsistent. It is commonly used for comparison operations in query or join conditions.
- The data types of arguments and parameters are inconsistent when a function is called.
- The data types of target columns to be updated by DML statements (including INSERT, UPDATE, MERGE, and REPLACE) and the defined column types are inconsistent.
- Using CAST(expr AS datatype) can explicitly convert an expression to a data type.

- After the target data type of the final projection column is determined by set operations (UNION, MINUS, EXCEPT, and INTERSECT), the type of the projection column in each SELECT statement is inconsistent with the target data type.
- In other expression calculation scenarios, the target data type used for comparison or final result is determined based on the data type of different expressions.
  - DECODE
  - CASE WHEN
  - lexpr [NOT] IN (expr_list)
  - BETWEEN AND
  - JOIN USING(a,b)
  - GREATEST and LEAST
  - NVL and COALESCE

GaussDB and MySQL have different rules for data type conversion and target data types. The following examples show the differences between the two processing modes:

```
-- MySQL: The execution result of IN is 0, indicating false. According to the rule, '1970-01-01' is compared with the expressions in the list in sequence. The results are all 0s. Therefore, the final result is 0.
mysql> SELECT '1970-01-01' IN ('1970-01-02', 1, '1970-01-02');
+-------------------------------------------------+
| '1970-01-01' in ('1970-01-02', 1, '1970-01-02') |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+

-- GaussDB: The execution result of IN is true, which is opposite to the MySQL result. The common type selected based on the rule is int. Therefore, the left expression '1970-01-01' is converted to the int type and compared with the value after the expression in the list is converted to the int type.
-- When '1970-01-01' and '1970-01-02' are converted to the int type, the values are 1970. (In MySQL-compatible mode, invalid characters and the following content are ignored during conversion, and the previous part is converted to the int type.) The comparison result is equal. Therefore, the returned result is true.
gaussdb=# SELECT '1970-01-01' IN ('1970-01-02', 1::int, '1970-01-02') AS result;
 result
--------
 t
(1 row)
```

Differences in data type conversion rules:

- The GaussDB clearly defines the conversion rules between different data types.
  - Whether to support conversion: Conversion is supported only when the conversion path of two types is defined in the pg_cast system catalog.
  - Conversion scenarios: conversion in any scenario, conversion only in CAST expressions, and conversion only during value assignment. In scenarios that are not supported, data type conversion cannot be performed even if the conversion path is defined.
- MySQL supports conversion between any two data types.

Due to the preceding differences, when MySQL-based applications are migrated to GaussDB, an error may be reported because the SQL statement does not support the conversion between different data types. In the scenario where conversion is

supported, different conversion rules result in different execution results of SQL statements.

You are advised to use the same data type in SQL statements for comparison or value assignment to avoid unexpected results or performance loss caused by data type conversion.

Differences in target data type selection rules:

In some scenarios, the data type to be compared or returned can be determined only after the types of multiple expressions are considered. For example, in the UNION operation, projection columns at the same position in different SELECT statements are of different data types. The final data type of the query result needs to be determined based on the data type of the projection columns in each SELECT statement.

GaussDB and MySQL have different rules for determining the target data types.

- GaussDB rules:
  - If the operand types of operators are inconsistent, the operand types are not converted to the target type before calculation. Instead, operators of two data types are directly registered, and two types of processing rules are defined during operator processing. In this mode, implicit type conversion does not exist, but the customized processing rule implies the conversion operation.
  - Rules for determining the target data type in the set operation and expression scenarios:
    - If all types are the same, it is the target type.
    - If the two data types are different, check whether the data types are of the same type, such as the numeric type, character type, and date and time type. If they do not belong to the same type, the target type cannot be determined. In this case, an error is reported during SQL statement execution.
    - For data types with the same **category** attribute (defined in the pg_type system catalog), the data type with the **preferred** attribute (defined in the pg_type system catalog) is selected as the target type. If operand 1 can be converted to operand 2 (no conversion path), but operand 2 cannot be converted to operand 1 or the priority of the numeric type is lower than that of operand 2, then operand 2 is selected as the target type.
    - If three or more data types are involved, the rule for determining the target type is as follows: common_type(type1,type2,type3) = common_type(common_type(type1,type2),type3). Perform iterative processing in sequence to obtain the final result.
    - For IN and NOT IN expressions, if the target type cannot be determined based on the preceding rules, each expression in **lexpr** and **expr_list** is compared one by one based on the equivalent operator (=).
    - Precision determination: The precision of the finally selected expression is used as the final result.

- MySQL rules:
  - If the operand types of operators are inconsistent, determine the target type based on the following rules. Then, convert the inconsistent operand types to the target type and then process the operands.
    - If both parameters are of the string type, they are compared based on the string type.
    - If both parameters are of the integer type, they are compared based on the integer type.
    - If a hexadecimal value is not compared with a numeric value, they are compared based on the binary string.
    - If one parameter is of the datetime/timestamp type, and the other parameter is a constant, the constant is converted to the timestamp type for comparison.
    - If one parameter is of the decimal type, the data type used for comparison depends on the other parameter. If the other type is decimal or integer, the decimal type is used. If the other type is not decimal, the real type is used.
    - In other scenarios, the data type is converted to the real type for comparison.
  - Rules for determining the target data type in the set operation and expression scenarios:
    - Establish a target type matrix between any two types. Given two types, the target type can be determined by using the matrix.
    - If three or more data types are involved, the rule for determining the target type is as follows: common_type(type1,type2,type3) = common_type(common_type(type1,type2),type3). Perform iterative processing in sequence to obtain the final result.
    - If the target type is integer and each expression type contains signed and unsigned integers, the type is promoted to an integer type with higher precision. The result is unsigned only when all expressions are unsigned. Otherwise, the result is signed.
    - The highest precision in the expression is used as the final result.

According to the preceding rules, GaussDB and MySQL differ greatly in data type conversion rules and types cannot be directly compared. In the preceding scenario, the execution result of SQL statements may be different from that in MySQL. In the current version, you are advised to use the same type for all expressions or use CAST to convert the type to the required type in advance to avoid differences.

3.3.2 System Functions

3.3.2.1 Flow Control Functions

Table 3-45 Flow control functions

MySQL	GaussDB	Difference
IF()	Supported, with differences.	• The **expr1** input parameter supports only the Boolean type. If an input parameter of the non-BOOL type cannot be converted to the BOOL type, an error is reported. • If the types of **expr2** and **expr3** are different and no implicit conversion function exists between the two types, an error is reported. • If the two input parameters are of the same type, the input parameter type is returned. • If the **expr2** and **expr3** input parameters are of the NUMERIC, STRING, or TIME type respectively, GaussDB outputs the TEXT type, while MySQL outputs the VARCHAR type.
IFNULL()	Supported, with differences.	• If the types of **expr1** and **expr2** are different and no implicit conversion function exists between the two types, an error is reported. • If the two input parameters are of the same type, the input parameter type is returned. • If the **expr1** and **expr2** input parameters are of the NUMERIC, STRING, or TIME type respectively, GaussDB outputs the TEXT type, while MySQL outputs the VARCHAR type. • If one input parameter is of the FLOAT4 type and the other is of any type in the numeric category, GaussDB returns the DOUBLE type. In MySQL, if one input parameter is of FLOAT4 type and the other is of the TINYINT, UNSIGNED TINYINT, SMALLINT, UNSIGNED SMALLINT, MEDIUMINT, UNSIGNED MEDIUMINT, or BOOL type, the FLOAT4 type is returned. If the first is of FLOAT4 type and the second is of BIGINT or UNSIGNED BIGINT type, the FLOAT type is returned.

MySQL	GaussDB	Difference

NULLIF()	Supported, with differences.	• The NULLIF() type derivation in GaussDB complies with the following logic:
		– If the data types of two parameters are different and the two input parameter types have an equality comparison operator, the left value type corresponding to the equality comparison operator is returned. Otherwise, the two input parameter types are forcibly compatible.
		– If an equality comparison operator exists after forcible type compatibility, the left value type of the equality comparison operator after forcible type compatibility is returned.
		– If the corresponding equality operator cannot be found after forcible type compatibility, an error is reported.
		-- The two input parameter types have an equality comparison operator.
		gaussdb=# SELECT pg_typeof(nullif(1::int2, 2::int8));
		pg_typeof
		---------- 	
		smallint
		(1 row)
		-- The two input parameter types do not have the equality comparison operator, but the equality comparison operator can be found after forcible type compatibility.
		gaussdb=# SELECT pg_typeof(nullif(1::int1, 2::int2));
		pg_typeof
		---------- 	
		bigint
		(1 row)
		
		-- The two input parameter types do not have the equality comparison operator, and the equality comparison operator does not exist after forcible type compatibility.
		gaussdb=# SELECT nullif(1::bit, '1'::MONEY);
		ERROR: operator does not exist: bit = money
		LINE 1: SELECT nullif(1::bit, '1'::MONEY);
		^
		HINT: No operator matches the given name and argument type(s). You might need to add explicit type casts.
		CONTEXT: referenced column: nullif
		• The MySQL output type is related only to the type of the first input parameter.
		– If the type of the first input parameter is TINYINT, SMALLINT, MEDIUMINT, INT, or BOOL, the output is of the INT type.
		– If the type of the first input parameter is BIGINT, the output is of the BIGINT type.

MySQL	GaussDB	Difference
		– When the type of the first input parameter is UNSIGNED TINYINT, UNSIGNED SMALLINT, UNSIGNED MEDIUMINT, UNSIGNED INT, or BIT, the output is of the UNSIGNED INT type. – If the type of the first input parameter is UNSIGNED BIGINT, the output is of the UNSIGNED BIGINT type. – If the type of the first input parameter is of the FLOAT, DOUBLE, or REAL type, the output is of the DOUBLE type. – If the type of the first input parameter DECIMAL or NUMERIC, the output is of the DECIMAL type. – If the type of the first input parameter is DATE, TIME, DATE, DATETIME, TIMESTAMP, CHAR, VARCHAR, TINYTEXT, ENUM, or SET, the output is of the VARCHAR type. – If the type of the first input parameter is TEXT, MEDIUMTEXT, or LONGTEXT, the output is of the LONGTEXT type. – If the type of the first input parameter is TINYBLOB, the output is of the VARBINARY type. – If the type of the first input parameter is MEDIUMBLOB or LONGBLOB, the output is of the LONGBLOB type. – If the type of the first input parameter is BLOB, the output is of the BLOB type.
ISNULL()	Supported, with differences.	In GaussDB, the return value is **t** or **f** of the BOOLEAN type. In MySQL, the return value is **1** or **0** of the INT type.

3.3.2.2 Date and Time Functions

The date and time functions in GaussDB in MySQL-compatible mode, with the same behavior as MySQL, are described as follows:

- Functions may use time expressions as their input parameters.

  Time expressions mainly include text, datetime, date, and time. Besides, all types that can be implicitly converted to time expressions can be input parameters. For example, a number can be implicitly converted to text and then used as a time expression.

  However, different functions take effect in different ways. For example, the datediff function calculates only the difference between dates. Therefore, time

expressions are parsed as the date type. The timestampdiff function parses time expressions as date, time, or datetime based on the **unit** parameter before calculating the time difference.

- The input parameters of functions may contain an invalid date.

  Generally, the supported date and datetime ranges are the same as those of MySQL. The value of date ranges from '0000-01-01' to '9999-12-31', and the value of datetime ranges from '0000-01-01 00:00:00' to '9999-12-31 23:59:59'. Although GaussDB supports larger date and datetime ranges than MySQL, dates out of range are still considered invalid.

  In most cases, time functions report an alarm and return NULL if the input date is invalid, unless the invalid date can be converted by CAST.

- Separators for input parameters of functions:

  For a time function, all non-digit characters are regarded as separators when input parameters are processed. The standard format is recommended: Use hyphens (-) to separate year, month, and day, use colons (:) to separate hour, minute, and second, and use a period (.) before milliseconds.

  Error-prone scenario: When **SELECT timestampdiff(hour, '2020-03-01 00:00:00', '2020-02-28 00:00:00+08');** is executed in a MySQL-compatible database, the time function does not automatically calculate the time zone. Therefore, +08 is not identified as the time zone. Instead, + is used as the separator for calculation as seconds.

Most function scenarios of GaussDB date and time functions are the same as those of MySQL, but there are still differences. Some differences are as follows:

- If an input parameter of a function is **NULL**, the function returns **NULL**, and no WARNING or ERROR is reported. These functions include:

  from_days, date_format, str_to_date, datediff, timestampdiff, date_add, subtime, month, time_to_sec, to_days, to_seconds, dayname, monthname, convert_tz, sec_to_time, addtime, adddate, date_sub, timediff, last_day, weekday, from_unixtime, unix_timestamp, subdate, day, year, weekofyear, dayofmonth, dayofyear, week, yearweek, dayofweek, time_format, hour, minute, second, microsecond, quarter, get_format, extract, makedate, period_add, timestampadd, period_diff, utc_time, utc_timestamp, maketime, and curtime.

  Example:

  ```
  gaussdb=# SELECT day(null);
   day
  -----

  (1 row)
  ```

- Some functions with pure numeric input parameters are different from those of MySQL. Numeric input parameters without quotation marks are converted into text input parameters for processing.

  Example:

  ```
  gaussdb=# SELECT day(19231221.123141);
  WARNING:  Incorrect datetime value: "19231221.123141"
  CONTEXT:  referenced column: day
   day
  -----

  (1 row)
  ```


- Time and date calculation functions are adddate, subdate, date_add, and date_sub. If the calculation result is a date, the supported range is [0000-01-01,9999-12-31]. If the calculation result is a date and time, the supported range is [0000-01-01 00:00:00.000000,9999-12-31 23:59:59.999999]. If the calculation result exceeds the supported range, an ERROR is reported in strict mode, or a WARNING is reported in loose mode. If the date result after calculation is within the range [0000-01-01,0001-01-01], GaussDB returns the result normally. MySQL returns '0000-00-00'.

  Example:

  ```
  gaussdb=# SELECT subdate('0000-01-01', interval 1 hour);
  ERROR:  Datetime function: datetime field overflow
  CONTEXT:  referenced column: subdate

  gaussdb=# SELECT subdate('0001-01-01', interval 1 day);
    subdate
  -------------
   0000-12-31

  (1 row)
  ```

- If the input parameter of the date or datetime type of the date and time function contains month 0 or day 0, the value is invalid. In strict mode, an ERROR is reported. In loose mode, if the input is a character string or number, a WARNING is reported. If the input is of the date or datetime type, the system processes the input as December of the previous year or the last day of the previous month.

  If the type of the CAST function is converted to date or datetime, an ERROR is reported in strict mode. In loose mode, no WARNING is reported. Instead, the system processes the input as December of the previous year or the last day of the previous month. Pay attention to this difference. MySQL outputs the value as it is, even if the year, month, and day are set to **0**.

  Example:

  ```
  gaussdb=# SELECT adddate('2023-01-00', 1);-- Strict mode
  ERROR:  Incorrect datetime value: "2023-01-00"
  CONTEXT:  referenced column: adddate

  gaussdb=# SELECT adddate('2023-01-00', 1); -- Loose mode
  WARNING:  Incorrect datetime value: "2023-01-00"
  CONTEXT:  referenced column: adddate
   adddate
  ---------

  (1 row)

  gaussdb=# SELECT adddate(date'2023-00-00', 1); -- Loose mode
    adddate
  ------------
   2022-12-01
  (1 row)

  gaussdb=# SELECT cast('2023/00/00' as date); -- Loose mode
      date
  ------------
   2022-11-30
  (1 row)

  gaussdb=# SELECT cast('0000-00-00' as datetime); -- Loose mode
        timestamp
  ---------------------
   0000-00-00 00:00:00
  (1 row)
  ```


- If the input parameter of the function is of the numeric data type, no error is reported in the case of invalid input, and the input parameter is processed as 0.

  Example:

```
gaussdb=# SELECT from_unixtime('aa');
    from_unixtime
---------------------
 1970-01-01 08:00:00
(1 row)
```

- A maximum of six decimal places are allowed. Decimal places with all 0s are not allowed.

  Example:

```
gaussdb=# SELECT from_unixtime('1234567899.00000');
    from_unixtime
---------------------
 2009-02-14 07:31:39
(1 row)
```

- If the time function parameter is a character string, the result is correct only when the year, month, and day are separated by a hyphen (-) and the hour, minute, and second are separated by a colon (:).

  Example:

```
gaussdb=# SELECT adddate('20-12-12',interval 1 day);
  adddate
------------
 2020-12-13
(1 row)
```

- If the return value of a function is of the varchar type in MySQL, the return value of the function is of the text type in GaussDB.

```
-- Return value of a function in GaussDB.
gaussdb=# SELECT pg_typeof(adddate('2023-01-01', 1));
 pg_typeof
-----------
 text
(1 row)

-- Return value of a function in MySQL.
mysql> CREATE VIEW v1 AS SELECT adddate('2023-01-01', 1);
Query OK, 0 rows affected (0.00 sec)

mysql> DESC v1;
+--------------------------+-------------+------+-----+---------+-------+
| Field                    | Type        | Null | Key | Default | Extra |
+--------------------------+-------------+------+-----+---------+-------+
| adddate('2023-01-01', 1) | varchar(29) | YES  |     | NULL    |       |
+--------------------------+-------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

Table 3-46 Date and time functions

MySQL	GaussDB	Difference
ADDDATE()	Supported, with differences.	The performance of this function is different from that of MySQL due to interval expression differences. For details, see **INTERVAL**.

MySQL	GaussDB	Difference
ADDTIME()	Supported, with differences.	• MySQL returns NULL if the second input parameter is a string in the DATETIME format. GaussDB can calculate the value. • The value range of an input parameter is ['0001-01-01 00:00:00', 9999-12-31 23:59:59.999999]. • If the first parameter of the ADDTIME function in MySQL is a dynamic parameter (for example, in a prepared statement), the return type is TIME. Otherwise, the parse type of the function is derived from the parse type of the first parameter. The return value rules of the ADDTIME function in GaussDB are as follows: – The first input parameter is of the date type, the second input parameter is of the date type, and the return value is of the time type. – The first input parameter is of the date type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the date type, the second input parameter is of the datetime type, and the return value is of the time type. – The first input parameter is of the date type, the second input parameter is of the time type, and the return value is of the time type. – The first input parameter is of the text type, the second input parameter is of the date type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the datetime type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the time type, and the return value is of the text type.

MySQL	GaussDB	Difference
		– The first input parameter is of the datetime type, the second input parameter is of the date type, and the return value is of the datetime type. – The first input parameter is of the datetime type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the datetime type, the second input parameter is of the datetime type, and the return value is of the datetime type. – The first input parameter is of the datetime type, the second input parameter is of the time type, and the return value is of the datetime type. – The first input parameter is of the time type, the second input parameter is of the date type, and the return value is of the time type. – The first input parameter is of the time type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the time type, the second input parameter is of the datetime type, and the return value is of the time type. – The first input parameter is of the time type, the second input parameter is of the time type, and the return value is of the time type.
CONVERT_TZ()	Supported.	-
CURDATE()	Supported.	-
CURRENT_DATE(), CURRENT_DATE	Supported.	-

MySQL	GaussDB	Difference
CURRENT_TIME(), CURRENT_TIME	Supported, with differences.	The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. For other values, an error is reported. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Supported, with differences.	The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. If the input integer value is greater than 6, an alarm is generated and the time value is output based on the precision 6. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.
CURTIME()	Supported, with differences.	In GaussDB, if a character string or a non-integer value is entered, the value is implicitly converted into an integer and then the precision is verified. If the value is beyond the [0,6] range, an error is reported. If the value is within the range, the time value is output normally. In MySQL, an error is reported. The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. For other values, an error is reported. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.

MySQL	GaussDB	Difference
YEARWEEK()	Supported.	-
DATE_ADD()	Supported, with differences.	The performance of this function is different from that of MySQL due to interval expression differences. For details, see **INTERVAL**.
DATE_FORMAT()	Supported.	-
DATE_SUB()	Supported, with differences.	The performance of this function is different from that of MySQL due to interval expression differences. For details, see **INTERVAL**.
DATEDIFF()	Supported.	-
DAY()	Supported.	-
DAYNAME()	Supported.	-
DAYOFMONTH()	Supported.	-
DAYOFWEEK()	Supported.	-
DAYOFYEAR()	Supported.	-
EXTRACT()	Supported.	-
FROM_DAYS()	Supported.	-
FROM_UNIXTIME()	Supported.	-
GET_FORMAT()	Supported.	-
HOUR()	Supported.	-
LAST_DAY	Supported.	-

MySQL	GaussDB	Difference
LOCALTIME(), LOCALTIME	Supported, with differences.	The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. For other integer values, an error is reported. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.
LOCALTIMESTAMP, LOCALTIMESTAMP()	Supported, with differences.	The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. If the input integer value is greater than 6, an alarm is generated and the time value is output based on the precision 6. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.
MAKEDATE()	Supported.	-
MAKETIME()	Supported, with differences.	When the input parameter is NULL, GaussDB does not support self-nesting of the maketime function, but MySQL supports.
MICROSECOND()	Supported.	-
MINUTE()	Supported.	-
MONTH()	Supported.	-
MONTHNAME()	Supported.	-

MySQL	GaussDB	Difference
NOW()	Supported, with differences.	The time value (after the decimal point) output by precision is rounded off in GaussDB and directly truncated in MySQL. The trailing 0s of the time value (after the decimal point) output by precision are not displayed in GaussDB but displayed in MySQL. GaussDB supports only an integer value within the range of [0,6] as the precision of the returned time. If the input integer value is greater than 6, an alarm is generated and the time value is output based on the precision 6. In MySQL, a precision value within [0,6] is valid, but an input integer value is divided by 256 to get a remainder. For example, if the input integer value is **257**, the time value with precision 1 is returned.
PERIOD_ADD()	Supported, with differences.	If the input parameter **period** or result is less than 0, GaussDB reports an error by referring to the performance in MySQL 8.0.x. Integer wrapping occurs in MySQL 5.7. As a result, the calculation result is abnormal.
PERIOD_DIFF()	Supported, with differences.	If the input parameter or result is less than 0, GaussDB reports an error by referring to the performance in MySQL 8.0.x. Integer wrapping occurs in MySQL 5.7. As a result, the calculation result is abnormal.
QUARTER()	Supported.	-
SEC_TO_TIME()	Supported.	-
SECOND()	Supported.	-
STR_TO_DATE()	Supported, with differences.	GaussDB returns the text type, while MySQL returns the datetime or date type.
SUBDATE()	Supported, with differences.	The performance of this function is different from that of MySQL due to interval expression differences. For details, see **INTERVAL**.

MySQL	GaussDB	Difference
SUBTIME()	Supported, with differences.	• MySQL returns NULL if the second input parameter is a string in the DATETIME format. GaussDB can calculate the value. • The value range of an input parameter is ['0001-01-01 00:00:00', 9999-12-31 23:59:59.999999]. • If the first parameter of the SUBTIME function in MySQL is a dynamic parameter (for example, in a prepared statement), the return type is TIME. Otherwise, the parse type of the function is derived from the parse type of the first parameter. The return value rules of the SUBTIME function in GaussDB are as follows: – The first input parameter is of the date type, the second input parameter is of the date type, and the return value is of the time type. – The first input parameter is of the date type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the date type, the second input parameter is of the datetime type, and the return value is of the time type. – The first input parameter is of the date type, the second input parameter is of the time type, and the return value is of the time type. – The first input parameter is of the text type, the second input parameter is of the date type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the datetime type, and the return value is of the text type. – The first input parameter is of the text type, the second input parameter is of the time type, and the return value is of the text type.

MySQL	GaussDB	Difference
		– The first input parameter is of the datetime type, the second input parameter is of the date type, and the return value is of the datetime type. – The first input parameter is of the datetime type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the datetime type, the second input parameter is of the datetime type, and the return value is of the datetime type. – The first input parameter is of the datetime type, the second input parameter is of the time type, and the return value is of the datetime type. – The first input parameter is of the time type, the second input parameter is of the date type, and the return value is of the time type. – The first input parameter is of the time type, the second input parameter is of the text type, and the return value is of the text type. – The first input parameter is of the time type, the second input parameter is of the datetime type, and the return value is of the time type. – The first input parameter is of the time type, the second input parameter is of the time type, and the return value is of the time type.
SYSDATE()	Supported, with differences.	In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value), while GaussDB does not.
YEAR()	Supported.	-
TIME_FORMAT()	Supported.	-
TIME_TO_SEC()	Supported.	-
TIMEDIFF()	Supported.	-
WEEKOFYEAR()	Supported.	-

MySQL	GaussDB	Difference
TIMESTAMPADD()	Supported.	-
TIMESTAMPDIFF()	Supported.	-
TO_DAYS()	Supported.	-
TO_SECONDS()	Supported.	-
UNIX_TIMESTAMP()	Supported, with differences.	GaussDB returns the numeric type, while MySQL returns the int type.
UTC_DATE()	Supported, with differences.	• MySQL supports calling without parentheses, but GaussDB does not. In MySQL, an integer input value is wrapped when it reaches **255** (maximum value of a one-byte integer value). • MySQL input parameters support only integers ranging from 0 to 6. GaussDB supports input parameters that can be implicitly converted to integers ranging from 0 to 6.
UTC_TIME()	Supported, with differences.	
UTC_TIMESTAMP()	Supported, with differences.	
WEEK()	Supported.	-
WEEKDAY()	Supported.	-

3.3.2.3 String Functions

Table 3-47 String functions

MySQL	GaussDB	Difference
BIN()	Supported, with differences.	In GaussDB, the types supported by function input parameters are as follows: • Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec. • Boolean type: bool.
CONCAT()	Supported, with differences.	The data type of the return value of CONCAT is always text regardless of the data type of the parameter. However, in MySQL, if CONCAT contains binary parameters, the return value is binary.
CONCAT_WS()	Supported, with differences.	The data type of the return value of CONCAT_WS is always text regardless of the data type of the parameter. However, in MySQL, if CONCAT_WS contains binary parameters, the return value is binary. In other cases, the return value is a string.

MySQL	GaussDB	Difference
ELT()	Supported, with differences.	• In GaussDB, the types supported by function input parameter 1 are as follows: – Integer types: tinyint, smallint, mediumint, int, and bigint. – Unsigned integer types: tinyint unsigned, smallint unsigned, and int unsigned. – Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. – Floating-point types: float, real, and double. – Fixed-point types: numeric, decimal, and dec. – Boolean type: bool. • In GaussDB, the types supported by function input parameter 2 are as follows: – Integer types: tinyint, smallint, mediumint, int, and bigint. – Unsigned integer types: tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. – Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. – Floating-point types: float, real, and double. – Fixed-point types: numeric, decimal, and dec. – Boolean type: bool. – Large object types: tinyblob, blob, mediumblob, and longblob. – Date types: datetime, timestamp, date, and time.
FIELD()	Supported, with differences.	• When function input parameters range from the maximum bigint value to the maximum bigint unsigned value, incompatibility occurs. • When function input parameters are of the float(m, d), double(m, d), or real(m, d) type, the precision is higher and incompatibility occurs.

MySQL	GaussDB	Difference
FIND_IN_SET()	Supported, with differences.	When the database encoding is set to **'SQL_ASCII'**, the default case sensitivity rule is not supported. That is, if no character set rule is specified, uppercase and lowercase letters are treated as distinct.
INSERT()	Supported, with differences.	• The range of input parameters of the Int64 type is from –9223372036854775808 to +9223372036854775807. If a value is out of range, an error is reported. MySQL does not limit the range of input parameters of the numeric type. If an exception occurs, an alarm is generated, indicating that the value is set to the upper or lower limit. • The maximum length of the input parameter of the text type is 2^30 – 5 bytes, and the maximum length of the input parameter of the bytea type is 2^30 – 512 bytes. • If any of the **s1** and **s2** parameters is of the bytea type and the result contains invalid characters, the displayed result may be different from that of MySQL, but the character encoding is the same as that of MySQL.
LOCATE()	Supported, with differences.	When input parameter 1 is of the bytea type and input parameter 2 is of the text type, the behavior of GaussDB is different from that of MySQL.

MySQL	GaussDB	Difference
MAKE_SET()	Supported, with differences.	• When the **bits** parameter is an integer, the maximum range is int128, which is smaller than the MySQL range. • When the **bits** parameter is of the date type (datetime, timestamp, date, or time), it is not supported because the conversion from the date type to the integer type is different from that in MySQL. • GaussDB and MySQL are inherently different in the bit and Boolean types, causing different returned results. When the **bits** input parameter is of the Boolean type, and the **str** input parameter is of the bit or Boolean type, they are not supported. • When the **bits** input parameter is of the character string or text type, only the pure integer format is supported. In addition, the value range of pure integers is limited to bigint. • The integer value of the **str** input parameter exceeds the range from 81 negative nines to 81 positive nines. The return value is different from that of MySQL. • When the **str** input parameter is expressed in scientific notation, trailing zeros are displayed in GaussDB but not displayed in MySQL. This is an inherent difference.

MySQL	GaussDB	Difference
QUOTE()	Supported, with differences.	• If the **str** character string contains "\Z", "\r", "\%", or "_", GaussDB does not escape it, which is different from MySQL. The slash followed by digits may also cause differences, for example, "\563". This function difference is the escape character difference between GaussDB and MySQL. • The output format of "\b" in the **str** character string is different from that in MySQL. This is an inherent difference between GaussDB and MySQL. • If the **str** character string contains "\0", GaussDB cannot identify the character because the UTF-8 character set cannot identify the character. As a result, the input fails. This is an inherent difference between GaussDB and MySQL. • If **str** is of the bit or Boolean type, this type is not supported because it is different in GaussDB and MySQL. • GaussDB supports a maximum of 1 GB data transfer. The maximum length of the **str** input parameter is 536870908 bytes, and the maximum size of the result string returned by the function is 1 GB. • The integer value of the **str** input parameter exceeds the range from 81 negative nines to 81 positive nines. The return value is different from that of MySQL. • When the **str** input parameter is expressed in scientific notation, trailing zeros are displayed in GaussDB but not displayed in MySQL. This is an inherent difference.

MySQL	GaussDB	Difference
SPACE()	Supported, with differences.	• GaussDB allows an input parameter of no more than 1073741818 bytes. If the length exceeds the limit, an empty string is returned. By default, MySQL allows an input parameter of no more than 4194304 bytes. If the length exceeds the limit, an alarm is generated. • In GaussDB, the types supported by function input parameters are as follows: – Integer types: tinyint, smallint, mediumint, int, and bigint. – Unsigned integer types: tinyint unsigned, smallint unsigned, and int unsigned. – Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. – Floating-point types: float, real, and double. – Fixed-point types: numeric, decimal, and dec. – Boolean type: bool.
SUBSTR()	Supported.	-
SUBSTRING()	Supported.	-
SUBSTRING_INDEX()	Supported.	-
STRCMP()	Supported, with differences.	• In GaussDB, the types supported by function input parameters are as follows: – Character types: char, varchar, nvarchar2, and text – Binary type: BYTEA – Value type: tinying [unsigned], smallint [unsigned], integer [unsigned], bigint [unsigned], float4, float8, and numeric – Date and time type: date, time without time zone, datetime, and timestamptz • For the floating-point type in the numeric type, the precision may be different from that in MySQL due to different connection parameter settings. Therefore, this scenario is not recommended, or the NUMERIC type is used instead.

MySQL	GaussDB	Difference
SHA()/ SHA1()	Supported.	-
SHA2()	Supported.	-

3.3.2.4 Forced Conversion Functions

Table 3-48 Forced conversion functions

MySQL	GaussDB	Difference
CAST()	Supported, with differences.	The data type conversion rules and supported conversion types are subject to the conversion scope and rules supported by GaussDB.
CONVERT()	Supported, with differences.	The data type conversion rules and supported conversion types are subject to the conversion scope and rules supported by GaussDB.

3.3.2.5 Encryption Functions

Table 3-49 Encryption functions

MySQL	GaussDB	Difference
AES_DECRYPT()	Supported.	-
AES_ENCRYPT()	Supported.	-

3.3.2.6 JSON Functions

JSON function differences:

- If you add escape characters as input parameters to JSON functions and other functions that allow character inputs, the processing is different from that in MySQL by default. To be compatible with MySQL, set the GUC parameter **standard_conforming_strings** to **off**. In this case, the processing of escape characters is compatible with MySQL, but a warning is generated for non-standard character input. The escape characters \t and \u and escape digits are different from those in MySQL. The JSON_UNQUOTE() function is compatible with MySQL. Even if the GUC parameter is not set, no alarm is generated.
- When processing an ultra-long number (the number contains more than 64 characters), the JSON function of GaussDB parses the number as a DOUBLE

and uses scientific notation for counting. The input parameters of the non-JSON type are the same as those of MySQL. However, when input parameters of the JSON type are used, the JSON type is not completely compatible with MySQL. As a result, differences occur in this scenario. MySQL displays complete numbers. (When the number length exceeds 82, MySQL displays an incorrect result.) GaussDB still parses an ultra-long number into a double-precision value. Long numbers are stored using floating-point numbers. During calculation, precision loss occurs in both GaussDB and MySQL. Therefore, you are advised to use character strings to store long numbers.

```
gaussdb=# SELECT json_insert('[1, 4,
99999999999999999999999999999999999999999999999999999999999999999999999999]','$[6]',json
_insert('[1,4]','$[5]',99999999999999999999999999999999999999999999999999999999999999999999999999
999999));
        json_insert
------------------------------
 [1, 4, 1e+74, [1, 4, 1e+74]]
(1 row)
```

Table 3-50 JSON functions

MySQL	GaussDB	Difference
JSON_APPEND()	Supported.	-
JSON_ARRAY()	Supported.	-
JSON_ARRAY_APPEND()	Supported.	-
JSON_ARRAY_INSERT()	Supported.	-
JSON_CONTAINS()	Supported.	-
JSON_CONTAINS_PATH()	Supported.	-
JSON_DEPTH()	Supported, with differences.	GaussDB returns values of the int type, while MySQL returns values of the bigint type.
JSON_EXTRACT()	Supported.	-
JSON_INSERT()	Supported.	-
JSON_KEYS()	Supported.	-
JSON_LENGTH()	Supported, with differences.	GaussDB returns values of the int type, while MySQL returns values of the bigint type.
JSON_MERGE()	Supported.	-

MySQL	GaussDB	Difference
JSON_OBJECT()	Supported.	-
JSON_QUOTE()	Supported, with differences.	Return value difference: In GaussDB, JSON is returned. In MySQL, varchar or text is returned.
JSON_REMOVE()	Supported.	-
JSON_REPLACE()	Supported.	-
JSON_SEARCH()	Supported, with differences.	GaussDB returns the text type, while MySQL returns the JSON type.
JSON_SET()	Supported.	-
JSON_TYPE()	Supported, with differences.	JSON values of the numeric type are identified as number, which is different from MySQL.
JSON_UNQUOTE()	Supported.	-
JSON_VALID()	Supported.	-

3.3.2.7 Aggregate Functions

Table 3-51 Aggregate functions

MySQL	GaussDB	Difference
GROUP_CONCAT()	Supported, with differences.	• If the **group_concat** parameter contains both the DISTINCT and ORDER BY syntaxes, all expressions following ORDER BY must be in the DISTINCT expression. • **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. • The data type of the return value of **group_concat** is always text regardless of the data type of the parameter. For MySQL, if **group_concat** contains binary parameters, the return value is binary. In other cases, the return value is a character string. If the return value length is greater than 512 bytes, the data type is a character large object or binary large object. • The value of **group_concat_max_len** ranges from 0 to 1073741823. The maximum value is smaller than that of MySQL.

MySQL	GaussDB	Difference
DEFAULT()	Supported, with differences.	• The default value of a column is an array. GaussDB returns an array. MySQL does not support the array type. • GaussDB columns are hidden columns (such as **xmin** and **cmin**). The default function returns a null value. • GaussDB supports default values of partitioned tables, temporary tables, and multi-table join query. • GaussDB supports the query of nodes whose column names contain character string values (indicating names) and A_Star nodes (indicating asterisks [*]), for example, **default(tt.t4.id)** and **default(tt.t4.*)**. For invalid query column names and A_Star nodes, the error information reported by GaussDB is different from that reported by MySQL. • When the default value of a column is created in GaussDB, the range of the column type is not verified. As a result, an error may be reported when the default function is used. • If the default value of a column is a function expression, the default function in GaussDB returns the calculated value of the default expression of the column during table creation. The default function in MySQL returns NULL.

3.3.2.8 Numeric Operation Functions

Table 3-52 Numeric operation functions

MySQL	GaussDB	Difference
log2()	Supported, with differences.	• The display of decimal places is different from that in MySQL. Due to the limitation of the GaussDB floating-point data type, the **extra_float_digits** parameter is used to control the number of decimal places to be displayed. • Due to the internal processing difference of the input precision, the calculation results of GaussDB and MySQL are different. • The following data types are supported: – Integer types: bigint, int16, int, smallint, and tinyint. – Unsigned integer types: bigint unsigned, integer unsigned, smallint unsigned, and tinyint unsigned. – Floating-point number type: numeric and real. – Character string type: character, character varying, clob, and text. Only numeric integer strings are supported. – SET type. – NULL type.

MySQL	GaussDB	Difference
log10()	Supported, with differences.	• The display of decimal places is different from that in MySQL. Due to the limitation of the GaussDB floating-point data type, the **extra_float_digits** parameter is used to control the number of decimal places to be displayed. • Due to the internal processing difference of the input precision, the calculation results of GaussDB and MySQL are different. • The following data types are supported: – Integer types: bigint, int16, int, smallint, and tinyint. – Unsigned integer types: bigint unsigned, integer unsigned, smallint unsigned, and tinyint unsigned. – Floating-point number type: numeric and real. – Character string type: character, character varying, clob, and text. Only numeric integer strings are supported. – SET type. – NULL type.
RAND([seed])	Supported, with differences.	Due to the random number generation algorithm used in the function, the random number returned by the function is different from that returned by MySQL.

3.3.2.9 Other Functions

Table 3-53 Other functions

MySQL	GaussDB	Difference
UUID()	Supported.	-
UUID_SHORT()	Supported.	-

3.3.3 Operators

GaussDB is compatible with most MySQL operators, but there are some differences. Unless otherwise specified, the operator behavior in MySQL-compatible mode is the native GaussDB behavior by default.

Table 3-54 Operators

MySQL	GaussDB	Difference
NULL-safe equal (<=>)	Supported.	-
[NOT] REGEXP	Supported, with differences.	• If the GUC parameter **b_format_dev_version** is set to **'s2'** and a pattern string with escape characters such as "\\a", "\\d", "\\e", "\\n", "\\Z", or "\\u" is matched with source character strings "\a", "\d", "\e", "\n", "\Z", or "\u", the behavior of GaussDB is different from that of MySQL 5.7 but the same as that of MySQL 8.0. • When the GUC parameter **b_format_dev_version** is set to **'s2'**, "\b" in GaussDB can match "\\b", but the matching will fail in MySQL. • If the input parameter of the pattern string is invalid with only the right parenthesis ()), GaussDB and MySQL 5.7 will report an error, but MySQL 8.0 will not. • In the rule of matching the de|abc sequence with de or abc, when there are empty values on the left and right of the pipe symbol (|), MySQL 5.7 will report an error, but GaussDB and MySQL 8.0 will not. • The regular expression of the tab character "\t" can match the character class [:blank:] in GaussDB and MySQL 8.0 but cannot in MySQL 5.7. • GaussDB supports non-greedy pattern matching. That is, the number of matching characters is as small as possible. A question mark (?) is added after some special characters, for example, ?? *? +? {n}? {n,}? {n,m}? MySQL 5.7 does not support non-greedy pattern matching, and the error message "Got error 'repetition-operator operand invalid' from regexp" is displayed. MySQL 8.0 already supports this function. • In the binary character set, the text and BLOB types will be converted to the bytea type. However, the REGEXP operator does not support the bytea type. Therefore, the matching will fail.
[NOT] RLIKE	Supported, with differences.	Same as [NOT] REGEXP.

3.3.4 Character Sets

GaussDB allows you to specify the following character sets for databases, schemas, tables, or columns.

Table 3-55 Character sets

MySQL	GaussDB
utf8mb4	Supported.
gbk	Supported.
gb18030	Supported.

NOTE

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. However, an error is reported during verification in MySQL.

3.3.5 Collation Rules

GaussDB allows you to specify the following collation rules for schemas, tables, or columns.

NOTE

Differences in collation rules:

- Currently, collation rules can only be specified for the character string type and some binary types. You can check the typcollation attribute of a type in the pg_type system catalog. If it is not **0**, the type supports the collation. The collation can be specified for all types in MySQL. However, collation rules are meaningless except those for character strings and binary types.
- The current collation rules can be specified only when the corresponding character set is the same as the database-level character set.
- The default collation of the utf8mb4 character set is utf8mb4_general_ci, which is the same as that in MySQL 5.7. utf8mb4_0900_ai_ci is not the default collation of utf8mb4.
- In GaussDB, utf8 and utf8mb4 are the same character set.

Table 3-56 Collation rules

MySQL	GaussDB
utf8mb4_general_ci	Supported.
utf8mb4_unicode_ci	Supported.
utf8mb4_bin	Supported.
gbk_chinese_ci	Supported.
gbk_bin	Supported.

MySQL	GaussDB
gb18030_chinese_ci	Supported.
gb18030_bin	Supported.
binary	Supported.
utf8mb4_0900_ai_ci	Supported.
utf8_general_ci	Supported.
utf8_bin	Supported.

3.3.6 SQL

3.3.6.1 DDL

Table 3-57 DDL syntax compatibility

Description	Syntax	Difference
Create primary keys and UNIQUE indexes during table creation and modification.	ALTER TABLE and CREATE TABLE	• GaussDB does not support the UNIQUE INDEX|KEY index_name syntax. An error will be reported when the UNIQUE INDEX|KEY index_name syntax is used. However, MySQL supports these functions. • When a constraint is created as a global secondary index and USING BTREE is specified in the SQL statement, the underlying index is created as UB-tree. • When the table joined with the constraint is Ustore and USING BTREE is specified in the SQL statement, the underlying index is created as UB-tree.

Description	Syntax	Difference
Support prefix indexes.	CREATE INDEX	• The prefix length cannot exceed 2676. 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, an error may be reported when the index line length exceeds the threshold. • In the CREATE INDEX syntax, the following keywords cannot be used as prefix keys for column names: COALESCE, EXTRACT, GREATEST, LEAST, LNNVL, NULLIF, NVL, NVL2, OVERLAY, POSITION, REGEXP_LIKE, SUBSTRING, TIMESTAMPDIFF, TREAT, TRIM, XMLCONCAT, XMLELEMENT, XMLEXISTS, XMLFOREST, XMLPARSE, XMLPI, XMLROOT, and XMLSERIALIZE. • Prefix keys are not supported in primary key and unique key indexes.
Specify character sets and collation rules.	ALTER SCHEMA, ALTER TABLE, CREATE SCHEMA, and CREATE TABLE	-
Create a partitioned table.	CREATE TABLE PARTITION	-
Specify table-level and column-level comments during table creation and modification.	CREATE TABLE and ALTER TABLE	-
Specify index-level comments during index creation.	CREATE INDEX	-

Description	Syntax	Difference
Exchange the partition data of an ordinary table and a partitioned table.	ALTER TABLE PARTITION	Differences in ALTER TABLE EXCHANGE PARTITION: • If MySQL tables or partitions use tablespaces, data in partitions and ordinary tables cannot be exchanged. If GaussDB tables or partitions use different tablespaces, data in partitions and ordinary tables can still be exchanged. • MySQL does not verify the default values of columns. Therefore, data in partitions and ordinary tables can be exchanged even if the default values are different. GaussDB verifies the default values. If the default values are different, data in partitions and ordinary tables cannot be exchanged. • After the DROP COLUMN operation is performed on a partitioned table or an ordinary table in MySQL, if the table structure is still consistent, data can be exchanged between partitions and ordinary tables. In GaussDB, data can be exchanged between partitions and ordinary tables only when the deleted columns of ordinary tables and partitioned tables are strictly aligned. • MySQL and GaussDB use different hash algorithms. Therefore, data stored in the same hash partition may be inconsistent. As a result, the exchanged data may also be inconsistent.

Description	Syntax	Difference
		• MySQL partitioned tables do not support foreign keys. If an ordinary table contains foreign keys or other tables reference foreign keys of an ordinary table, data in partitions and ordinary tables cannot be exchanged. GaussDB partitioned tables support foreign keys. If the FOREIGN KEY constraints of two tables are the same, data in partitions and ordinary tables can be exchanged. If a GaussDB partitioned table does not contain foreign keys, an ordinary table is referenced by other tables, and the partitioned table is the same as the ordinary table, data in the partitioned table can be exchanged with that in the ordinary table.

Description	Syntax	Difference
Support auto-increment columns.	ALTER TABLE and CREATE TABLE	• Currently, only local auto-increment columns of each DN are supported. • It is recommended that the auto-increment column be the first column of a non-global secondary index. Otherwise, an alarm is generated when a table is created, and errors may occur when some operations are performed on a table that contains auto-increment columns, for example, ALTER TABLE EXCHANGE PARTITION. The auto-increment column in MySQL must be the first column of the index. • In the syntax AUTO_INCREMENT = value, **value** must be a positive number less than 2^127. MySQL does not verify the value. • An error occurs if the auto-increment continues after an auto-increment value reaches the maximum value of a column data type. In MySQL, errors or warnings may be generated during auto-increment, and sometimes auto-increment continues until the maximum value is reached. • GaussDB does not support the *innodb_autoinc_lock_mode* system variable, but when its GUC parameter **auto_increment_cache** is set to **0**, the behavior of inserting auto-increment columns in batches is

Description	Syntax	Difference
		similar to that when the MySQL system variable *innodb_autoinc_lock_mode* is set to **1**. ● When 0s, NULLs, and definite values are imported or batch inserted into auto-increment columns, the auto-increment values inserted after an error occurs in GaussDB may not be the same as those in MySQL. The **auto_increment_cache** parameter is provided to control the number of reserved auto-increment values. ● In different execution plans, the auto-increment sequence and reserved auto-increment values may be different from those in MySQL. For example, "INSERT INTO table VALUES(...),(...),..." is distributed to different DNs. Therefore, in some execution plans, DNs cannot obtain the number of rows to be inserted. The **auto_increment_cache** parameter is provided to control the number of reserved auto-increment values. ● When auto-increment is triggered by parallel import or insertion of auto-increment columns, the cache value reserved for each parallel thread is used only in the thread. If the cache value is not used up, the values of auto-increment columns in the table are discontinuous. The auto-

Description	Syntax	Difference
		increment value generated by parallel insertion cannot be guaranteed to be the same as that generated in MySQL. • The SERIAL data type of GaussDB is an original auto-increment column, which is different from the **AUTO_INCREMENT** column. The SERIAL data type of MySQL is the **AUTO_INCREMENT** column. • The value of **auto_increment_offset** cannot be greater than that of **auto_increment_increment**. Otherwise, an error occurs. MySQL allows it and states that **auto_increment_offset** will be ignored. • If a table has a primary key or index, the sequence in which the **ALTER TABLE** command rewrites table data may be different from that in MySQL. GaussDB rewrites table data based on the table data storage sequence, while MySQL rewrites table data based on the primary key or index sequence. As a result, the auto-increment sequence may be different. • When the **ALTER TABLE** command is used to add or modify auto-increment columns, the number of auto-increment values reserved for the first time is the number of rows in the table statistics. The number of rows in the

Description	Syntax	Difference
		statistics may not be the same as that in MySQL. • When auto-increment is performed in a trigger or user-defined function, the return value of last_insert_id is updated. MySQL does not update it. • If the values of the GUC parameters **auto_increment_offset** and **auto_increment_increment** are out of range, an error occurs. MySQL automatically changes the value to a boundary value. • The last_insert_id function is not supported. • Currently, local temporary tables do not support auto-increment columns. • If **sql_mode** is set to **no_auto_value_on_zero**, the auto-increment columns of the table are not subject to NOT NULL constraints. In GaussDB and MySQL, when the value of an auto-increment column is not specified, **NULL** will be inserted into the auto-increment column, but auto-increment is triggered for the former and not triggered for the latter.
Delete the PRIMARY KEY constraints of a table.	ALTER TABLE	-

Description	Syntax	Difference
Support the CREATE TABLE... LIKE syntax.	CREATE TABLE ... LIKE	• In versions earlier than MySQL 8.0.16, CHECK constraints are parsed but their functions are ignored. In this case, CHECK constraints are not replicated. GaussDB supports replication of CHECK constraints. • For the set data type, MySQL supports replication while GaussDB does not during table creation. • When a table is created, all PRIMARY KEY constraint names in MySQL are fixed to **PRIMARY KEY**. GaussDB does not support replication of PRIMARY KEY constraint names. • When a table is created, MySQL supports replication of UNIQUE KEY constraint names, but GaussDB does not. • When a table is created, MySQL versions earlier than 8.0.16 do not have CHECK constraint information, but GaussDB supports replication of CHECK constraint names. • When a table is created, MySQL supports replication of index names, but GaussDB does not. • When a table is created across sql_mode, MySQL is controlled by the loose mode and strict mode. The strict mode may become invalid in GaussDB. For example, if the source table has the default value

Description	Syntax	Difference
		"0000-00-00", GaussDB can create a table that contains the default value **"0000-00-00"** in "no_zero_date" strict mode, which means that the strict mode is invalid. MySQL fails to create the table because it is controlled by the strict mode. • MySQL supports cross-database table creation, but GaussDB does not. • If the source table is a temporary table, you can create a non-temporary table in MySQL but not in GaussDB.

Description	Syntax	Difference
Compatible with syntax for changing table names.	ALTER TABLE[IF EXISTS] tbl_name RENAME [TO | AS | =] new_tbl_name; RENAME {TABLE | TABLES} tbl_name TO new_tbl_name [, tbl_name2 TO new_tbl_name2, ...];	• The ALTER RENAME syntax in GaussDB supports only the function of changing the table name and cannot be coupled with other function operations. • In GaussDB, only the old table name column supports the schema.table_name format, and the new and old table names belong to the same schema. • GaussDB does not support renaming of old and new tables across schemas. However, if you have the permission, you can modify the names of tables in other schemas in the current schema. • The syntax for renaming multiple groups of tables in GaussDB supports renaming of all local temporary tables, but does not support the combination of local temporary tables and non-local temporary tables.

Description	Syntax	Difference
Create a partition.	ALTER TABLE [IF EXISTS] { table_name [*] | ONLY table_name | ONLY (table_name)} action [, ...]; action: move_clause | exchange_clause | row_clause | merge_clause | modify_clause | split_clause | add_clause | drop_clause | ilm_clause add_clause: ADD {{partition_less_than_item | partition_start_end_item | partition_list_item} | PARTITION({partition_less_than_item | partition_start_end_item | partition_list_item})}	• The ALTER TABLE table_name ADD PARTITION (partition_definition1, partition_definition1,…); syntax cannot be used to add multiple partitions. • Only the original syntax for adding multiple partitions is supported: ALTER TABLE table_name ADD PARTITION (partition_definition1), ADD PARTITION (partition_definition2[y1]), …;.

3.3.6.2 DML

Table 3-58 DML syntax compatibility

Description	Syntax	Difference
DELETE supports ORDER BY and LIMIT.	DELETE	-
UPDATE supports ORDER BY and LIMIT.	UPDATE	-

Description	Syntax	Difference

Support the REPLACE INTO syntax.	REPLACE	• Difference between the initial values of the time type. For example:
		– MySQL is not affected by the strict or loose mode. You can insert time 0 into a table.
		mysql> CREATE TABLE test(f1 TIMESTAMP NOT NULL, f2 DATETIME NOT NULL, f3 DATE NOT NULL);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> REPLACE INTO test VALUES(f1, f2, f3);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM test;
		+---------------------
		+---------------------+------------
		+
		| f1 | f2
		| f3 |
		+---------------------
		+---------------------+------------
		+
		| 0000-00-00 00:00:00 |
		0000-00-00 00:00:00 |
		0000-00-00 |
		+---------------------
		+---------------------+------------
		+
		1 row in set (0.00 sec)
		– The time 0 can be successfully inserted only when GaussDB is in loose mode.
		gaussdb=# SET b_format_version = '5.7';
		SET
		gaussdb=# SET b_format_dev_version = 's1';
		SET
		gaussdb=# SET sql_mode = '';
		SET
		gaussdb=# CREATE TABLE test(f1 TIMESTAMP NOT NULL, f2 DATETIME NOT NULL, f3 DATE NOT NULL) DISTRIBUTE BY HASH(f1);
		CREATE TABLE
		gaussdb=# REPLACE INTO test VALUES(f1, f2, f3);
		REPLACE 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2 | f3
		--------------------- 	
		+---------------------+------------

Description	Syntax	Difference

		0000-00-00 00:00:00 |
		0000-00-00 00:00:00 |
		0000-00-00
		(1 row)
		In strict mode, the error is reported: date/time field value out of range: "0000-00-00 00:00:00".
		• Difference between the initial values of the BIT type when NOT NULL exists. For example:
		– The initial value of the BIT type is an empty string '' in MySQL, that is:
		mysql> CREATE TABLE test(f1 BIT(3) NOT NULL);
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> REPLACE INTO test VALUES(f1);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT f1, f1 IS NULL FROM test;
		+----+------------+
		| f1 | f1 is null |
		+----+------------+
		| | 0 |
		| | 0 |
		+----+------------+
		2 rows in set (0.00 sec)
		– If the initial value of the BIT type is **NULL** in GaussDB, an error is reported.
		gaussdb=# CREATE TABLE test(f1 int, f2 BIT(3) NOT NULL) DISTRIBUTE BY HASH(f1);
		CREATE TABLE
		gaussdb=# REPLACE INTO test VALUES(1, f2);
		ERROR: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (1, null).

SELECT supports multi-partition query.	SELECT	-

Description	Syntax	Difference
UPDATE supports multi-partition update.	UPDATE	-

Description	Syntax	Difference
Import data by using LOAD DATA.	LOAD DATA	• The execution result of the LOAD DATA syntax is the same as that in MySQL strict mode. The loose mode is not adapted currently. • The **IGNORE** and **LOCAL** parameters are used only to ignore the conflicting rows when the imported data conflicts with the data in the table and to automatically fill default values for other columns when the number of columns in the file is less than that in the table. Other functions are not supported currently. • If the keyword LOCAL is specified and the file path is a relative path, the file is searched from the binary directory. If the keyword LOCAL is not specified and the file path is a relative path, the file is searched from the data directory. • If single quotation marks are specified as separators, escape characters, and newline characters in the syntax, lexical parsing errors occur. • The **[(col_name_or_user_var [, col_name_or_user_var]...)]** parameter cannot be used to specify a column repeatedly. • The newline character specified by **[FIELDS TERMINATED BY 'string']** cannot be the same as the separator specified by

Description	Syntax	Difference
		[LINES TERMINATED BY'string']. ● If the data written to a table by running **LOAD DATA** cannot be converted to the data type of the table, an error is reported. ● Columns can only be specified by column name instead of user variables. ● The LOAD DATA SET expression does not support the calculation of a specified column name. ● If there is no implicit conversion function between the return value type of the SET expression and the corresponding column type, an error is reported. ● LOAD DATA does not support the INSERT or DELETE trigger. ● LOAD DATA applies only to tables but not views. ● The default newline character of the file in Windows is different from that in Linux. LOAD DATA cannot identify this scenario and reports an error. You are advised to check the newline character at the end of lines in the file to be imported.

Description	Syntax	Difference

Compatible with INSERT IGNORE.	INSERT IGNORE	• GaussDB displays the error information after the downgrade. MySQL records the error information after the downgrade to the error stack and runs the **show warnings;** command to view the error information. For example:
		• Time type difference. For example:
		– The default values of **date**, **datetime**, and **timestamp** in GaussDB are **0**.
		gaussdb=# CREATE TABLE test(f1 DATE NOT NULL, f2 DATETIME NOT NULL, f3 TIMESTAMP NOT NULL);
		CREATE TABLE
		gaussdb=# INSERT IGNORE INTO test VALUES(NULL, NULL, NULL);
		WARNING: null value in column "f1" violates not-null constraint
		DETAIL: Failing row contains (null, null, null, null).
		WARNING: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (null, null, null, null).
		WARNING: null value in column "f3" violates not-null constraint
		DETAIL: Failing row contains (null, null, null, null).
		INSERT 0 1
		gaussdb=#
		SELECT * FROM test;
		f1 | f2 | f3
		------------+---------------------+---------------------
		1970-01-01 | 1970-01-01 00:00:00 | 1970-01-01 00:00:00
		(1 row)
		– The default values of **date**, **datetime**, and **timestamp** in MySQL are **0**.
		mysql> CREATE TABLE test(f1 DATE NOT NULL, f2 DATETIME NOT NULL, f3 TIMESTAMP NOT NULL);

Description	Syntax	Difference

		Query OK, 0 rows affected (0.00 sec)
		
		mysql> INSERT IGNORE INTO test VALUES(NULL, NULL, NULL);
		Query OK, 1 row affected, 3 warnings (0.00 sec)
		
		mysql> show warnings;
		+---------+------
		+----------------------------+
		| Level | Code |
		Message |
		+---------+------
		+----------------------------+
		| Warning | 1048 | Column 'f1' cannot be null |
		| Warning | 1048 | Column 'f2' cannot be null |
		| Warning | 1048 | Column 'f3' cannot be null |
		+---------+------
		+----------------------------+
		3 rows in set (0.00 sec)
		
		mysql> SELECT * FROM test;
		+------------+---------------------
		+---------------------+
		| f1 | f2 |
		f3 |
		+------------+---------------------
		+---------------------+
		| 0000-00-00 | 0000-00-00 00:00:00 | 0000-00-00 00:00:00 |
		+------------+---------------------
		+---------------------+
		1 row in set (0.00 sec)
		• GaussDB does not support the MySQL bit type. Therefore, the INSERT IGNORE error downgrade is not supported when the NOT NULL constraint of the bit type is ignored and the length of the inserted bit type is different from that defined.
		– Bit type in GaussDB
		gaussdb=# CREATE TABLE test(f1 BIT(10) NOT NULL);
		CREATE TABLE
		gaussdb=# INSERT IGNORE INTO test VALUES(NULL);
		ERROR: Un-support feature
		DETAIL: ignore null for insert statement is not supported in column f1.
		gaussdb=# INSERT IGNORE INTO test VALUES('1010');

Description	Syntax	Difference

		ERROR: bit string length 4 does not match type bit(10)
		CONTEXT: referenced column: f1
		– Bit type in MySQL
		mysql> CREATE TABLE test(f1 BIT(10) NOT NULL);
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> INSERT IGNORE INTO test VALUES(NULL);
		Query OK, 1 row affected, 1 warning (0.00 sec)
		
		mysql> INSERT IGNORE INTO test VALUES('1010');
		Query OK, 1 row affected, 1 warning (0.01 sec)
		• If the precision is specified for the time type in MySQL, the precision is displayed when the zero value is inserted. It is not displayed in GaussDB. For example:
		– Time precision specified in GaussDB
		gaussdb=# CREATE TABLE test(f1 TIME(3) NOT NULL, f2 DATETIME(3) NOT NULL, f3 TIMESTAMP(3) NOT NULL);
		CREATE TABLE
		gaussdb=# INSERT IGNORE INTO test VALUES(NULL,NULL,NULL);
		WARNING: null value in column "f1" violates not-null constraint
		DETAIL: Failing row contains (null, null, null).
		WARNING: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (null, null, null).
		WARNING: null value in column "f3" violates not-null constraint
		DETAIL: Failing row contains (null, null, null).
		INSERT 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2 | f3
		----------+----------+----------
		00:00:00 | 1970-01-01 00:00:00 | 1970-01-01 00:00:00
		(1 row)

Description	Syntax	Difference

		– Time precision specified in MySQL
		mysql> CREATE TABLE test(f1 TIME(3) NOT NULL, f2 DATETIME(3) NOT NULL, f3 TIMESTAMP(3) NOT NULL);
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> INSERT IGNORE INTO test VALUES(NULL,NULL,NULL);
		Query OK, 1 row affected, 3 warnings (0.00 sec)
		
		mysql> SELECT * FROM test;
		+--------------
		+-------------------------
		+-------------------------+
		| f1 | f2 | f3 |
		+--------------
		+-------------------------
		+-------------------------+
		| 00:00:00.000 | 0000-00-00 00:00:00.000 | 0000-00-00 00:00:00.000 |
		+--------------
		+-------------------------
		+-------------------------+
		1 row in set (0.00 sec)
		• The execution process in MySQL is different from that in GaussDB. Therefore, the number of generated warnings may be different. For example:
		– Number of warnings generated in GaussDB
		gaussdb=# CREATE TABLE test(f1 INT, f2 INT not null);
		CREATE TABLE
		gaussdb=# INSERT INTO test VALUES(1,0),(3,0),(5,0);
		INSERT 0 3
		gaussdb=# INSERT IGNORE INTO test SELECT f1+1, f1/f2 FROM test;
		WARNING: division by zero
		CONTEXT: referenced column: f2
		WARNING: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (2, null).
		WARNING: division by zero
		CONTEXT: referenced column: f2
		WARNING: null value in column "f2" violates not-null

Description	Syntax	Difference

		constraint
		DETAIL: Failing row contains (4, null).
		WARNING: division by zero
		CONTEXT: referenced column: f2
		WARNING: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (6, null).
		INSERT 0 3
		– Number of warnings generated in MySQL
		mysql> CREATE TABLE test(f1 INT, f2 INT not null);
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> INSERT INTO test VALUES(1,0),(3,0),(5,0);
		Query OK, 3 rows affected (0.00 sec)
		Records: 3 Duplicates: 0 Warnings: 0
		
		mysql> INSERT IGNORE INTO test SELECT f1+1, f1/f2 FROM test;
		Query OK, 3 rows affected, 4 warnings (0.00 sec)
		Records: 3 Duplicates: 0 Warnings: 4
		• The differences between MySQL's and GaussDB's INSERT IGNORE in triggers are as follows:
		– INSERT IGNORE used in a GaussDB trigger
		gaussdb=# CREATE TABLE test1(f1 INT NOT NULL);
		CREATE TABLE
		gaussdb=# CREATE TABLE test2(f1 INT);
		CREATE TABLE
		gaussdb=# CREATE OR REPLACE FUNCTION trig_test() RETURNS TRIGGER AS $$
		gaussdb$# BEGIN
		gaussdb$# INSERT IGNORE INTO test1 VALUES(NULL);
		gaussdb$# RETURN NEW;
		gaussdb$# END;
		gaussdb$# $$ LANGUAGE plpgsql;
		CREATE FUNCTION
		gaussdb=# CREATE TRIGGER trig2 BEFORE INSERT ON test2 FOR EACH ROW EXECUTE PROCEDURE

Description	Syntax	Difference

		trig_test();
		CREATE TRIGGER
		gaussdb=# INSERT INTO test2 VALUES(NULL);
		WARNING: null value in column "f1" violates not-null constraint
		DETAIL: Failing row contains (null).
		CONTEXT: SQL statement "INSERT IGNORE INTO test1 VALUES(NULL)"
		PL/pgSQL function trig_test() line 3 at SQL statement
		INSERT 0 1
		gaussdb=# SELECT * FROM test1;
		 f1
		---- 	
		  0
		(1 rows)
		
		gaussdb=# SELECT * FROM test2;
		 f1
		---- 	
		
		(1 rows)
		– INSERT IGNORE used in a MySQL trigger
		mysql> CREATE TABLE test1(f1 INT NOT NULL);
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> CREATE TABLE test2(f1 INT);
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> DELIMITER ||
		mysql> CREATE TRIGGER trig2 BEFORE INSERT ON test2 FOR EACH ROW
		    -> BEGIN
		    -> INSERT IGNORE into test1 values(NULL);
		    -> END||
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> DELIMITER ;
		mysql> INSERT INTO test2 VALUES(NULL);
		ERROR 1048 (23000): Column 'f1' cannot be null
		mysql> INSERT IGNORE INTO test2 VALUES(NULL);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> SELECT * FROM test1;
		+----+
		| f1 |

Description	Syntax	Difference

		+----+
		| 0 |
		+----+
		1 row in set (0.00 sec)
		
		mysql> SELECT * FROM test2;
		+------+
		| f1 |
		+------+
		| NULL |
		+------+
		1 row in set (0.00 sec)
		• The implementation mechanism of Boolean and serial in GaussDB is different from that in MySQL. Therefore, the default zero value in GaussDB is different from that in MySQL. For example:
		– Behavior in GaussDB
		gaussdb=# CREATE TABLE test(f1 SERIAL, f2 BOOL NOT NULL);
		NOTICE: CREATE TABLE will create implicit sequence "test_f1_seq" for serial column "test.f1"
		CREATE TABLE
		gaussdb=# INSERT IGNORE INTO test values(NULL,NULL);
		WARNING: null value in column "f1" violates not-null constraint
		DETAIL: Failing row contains (null, null).
		WARNING: null value in column "f2" violates not-null constraint
		DETAIL: Failing row contains (null, null).
		INSERT 0 1
		gaussdb=# SELECT * FROM test;
		f1 | f2
		----+----
		0 | f
		(1 row)
		– Behavior in MySQL
		mysql> CREATE TABLE test(f1 SERIAL, f2 BOOL NOT NULL);
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> INSERT IGNORE INTO test values(NULL,NULL);
		Query OK, 1 row affected, 1 warning (0.00 sec)

Description	Syntax	Difference
		mysql> SELECT * FROM test; +----+----+ | f1 | f2 | +----+----+ | 1 | 0 | +----+----+ 1 row in set (0.00 sec)

3.3.6.3 DCL

Table 3-59 DCL syntax compatibility

Description	Syntax	Difference
Set names with COLLATE specified.	SET [SESSION | LOCAL] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};	GaussDB does not allow **charset_name** to be different from the database character set. For details, see "SQL Reference > SQL Syntax > S > SET" in *Developer Guide*.

3.3.7 Drivers

3.3.7.1 JDBC

3.3.7.1.1 JDBC API Reference

The JDBC API definitions in GaussDB are the same as those in MySQL and comply with industry standards. This section describes the behavior differences of JDBC APIs between GaussDB in MySQL-compatible mode and MySQL.

Obtaining Data from a Result Set

ResultSet objects provide a variety of methods to obtain data from a result set. **Table 3-60** describes the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 3-60 Common methods for obtaining data from a result set

Method	Description	Difference
int getInt(int columnIndex)	Obtains int data by column index.	-

Method	Description	Difference
int getInt(String columnLabel)	Obtains int data by column name.	-
String getString(int columnIndex)	Obtains string data by column index.	If the column type is integer and the column contains the **ZEROFILL** attribute, GaussDB pads 0s to meet the width required by the **ZEROFILL** attribute and outputs the result. MySQL directly outputs the result.
String getString(String columnLabel)	Obtains string data by column name.	If the column type is integer and the column contains the **ZEROFILL** attribute, GaussDB pads 0s to meet the width required by the **ZEROFILL** attribute and outputs the result. MySQL directly outputs the result.
Date getDate(int columnIndex)	Obtains date data by column index.	-
Date getDate(String columnLabel)	Obtains date data by column name.	-

