云数据库 GaussDB

与 MySQL 兼容性参考（集中式）

文档版本	01
发布日期	2025-06-30

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 概述.....1
1.1 MySQL 兼容性 B 模式概述.....1
1.2 MySQL 兼容性 M-Compatibility 模式概述.....1
2 MySQL 兼容性 B 模式.....3
2.1 数据类型.....3
2.1.1 数值数据类型.....3
2.1.2 日期与时间数据类型.....10
2.1.3 字符串数据类型.....21
2.1.4 二进制数据类型.....24
2.1.5 JSON 数据类型.....27
2.1.6 数据类型支持的属性.....27
2.1.7 数据类型转换.....27
2.2 系统函数.....30
2.2.1 流量控制函数.....30
2.2.2 日期和时间函数.....32
2.2.3 字符串函数.....42
2.2.4 强制转换函数.....46
2.2.5 加密函数.....46
2.2.6 信息函数.....47
2.2.7 JSON 函数.....47
2.2.8 聚合函数.....49
2.2.9 数字操作函数.....50
2.2.10 其他函数.....51
2.3 操作符.....51
2.4 字符集.....52
2.5 排序规则.....53
2.6 表达式.....53
2.7 SQL.....54
2.7.1 DDL.....54
2.7.2 DML.....62
2.7.3 DCL.....74
2.8 驱动.....74
2.8.1 JDBC.....74

2.8.1.1 JDBC 接口参考 74
3 MySQL 兼容性 M-Compatibility 模式 76
3.1 数据类型 76
3.1.1 数值数据类型 76
3.1.2 日期与时间数据类型 79
3.1.3 字符串数据类型 80
3.1.4 二进制数据类型 82
3.1.5 JSON 类型 85
3.1.6 数据类型支持的属性 87
3.1.7 数据类型转换 87
3.2 系统函数 103
3.2.1 系统函数兼容性概述 103
3.2.2 流程控制函数 104
3.2.3 日期和时间函数 106
3.2.4 字符串函数 109
3.2.5 类型转换函数 113
3.2.6 加密函数 115
3.2.7 比较函数 116
3.2.8 聚合函数 118
3.2.9 JSON 函数 125
3.2.10 窗口函数 127
3.2.11 数字操作函数 129
3.2.12 网络地址函数 131
3.2.13 其他函数 131
3.3 操作符 134
3.4 字符集 148
3.5 排序规则 149
3.6 事务 150
3.7 SQL 154
3.7.1 关键字 154
3.7.2 标识符 155
3.7.3 DDL 157
3.7.4 DML 182
3.7.5 DCL 218
3.7.6 其他语句 221
3.7.7 用户与权限 222
3.7.8 系统表和系统视图 226
3.8 驱动 229
3.8.1 ODBC 229
3.8.1.1 ODBC 接口参考 230

1 概述

1.1 MySQL 兼容性 B 模式概述

MySQL兼容性B模式主要介绍GaussDB数据库的MySQL兼容性B模式（即sql_compatibility='B'、且设置参数b_format_version='5.7'、b_format_dev_version='s1'时）与MySQL 5.7数据库的兼容性对比信息。仅介绍503.0.0版本后新增的兼容性特性，特性的相关规格和约束建议在《开发指南》中查看。

说明

当前形态下B模式（即sql_compatibility='B'）与分布式形态下MYSQL模式（即sql_compatibility='MYSQL'）实现逻辑相近。

当前形态下GaussDB数据库的“MySQL兼容性B模式“匹配分布式下的“MySQL兼容性MYSQL模式“，具体信息也可参见分布式形态下的《MySQL兼容性说明》中的“概述 > MySQL兼容性MYSQL模式概述“。

GaussDB数据库在数据类型、SQL功能和数据库对象等基本功能上与MySQL数据库兼容。

由于GaussDB数据库与MySQL数据库在底层框架实现上存在差异，GaussDB数据库与MySQL数据库仍存在部分差异。

1.2 MySQL 兼容性 M-Compatibility 模式概述

MySQL兼容性M-Compatibility模式主要介绍GaussDB数据库的MySQL兼容性M-Compatibility模式（即sql_compatibility='M'）与MySQL 5.7数据库的兼容性对比信息。仅介绍505.1版本后新增的兼容性特性，特性的相关规格和约束建议在《M-Compatibility开发指南》中查看。

GaussDB数据库在数据类型、SQL功能和数据库对象等基本功能上与MySQL数据库兼容。

GaussDB的执行计划和优化、EXPLAIN显示结果与MySQL不同。

由于GaussDB数据库与MySQL数据库在底层框架实现上存在差异，GaussDB数据库与MySQL数据库仍存在部分差异。

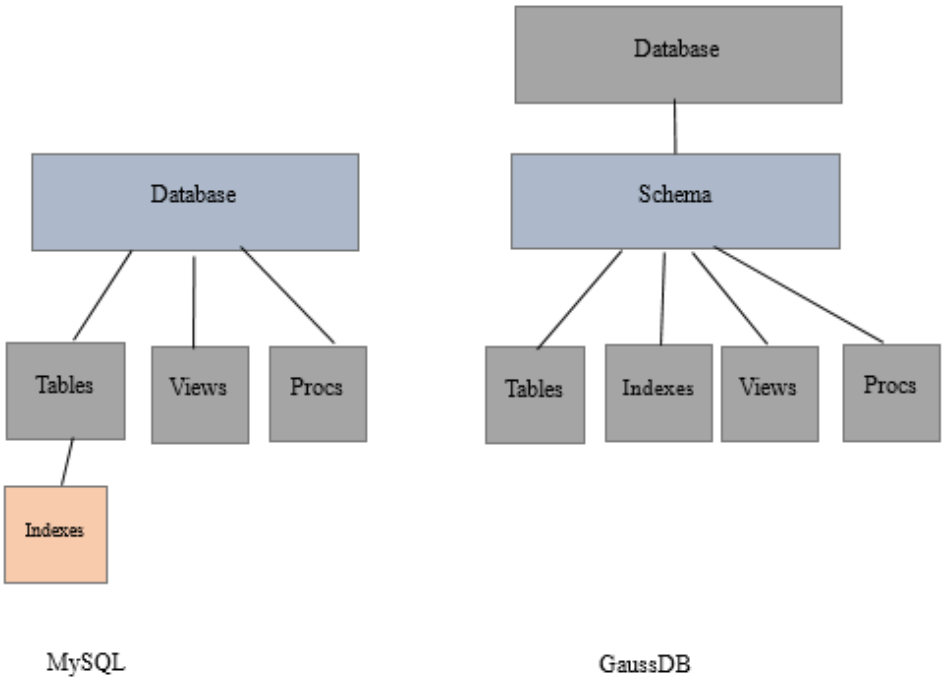
说明

- M-Compatibility模式在语法、数据类型、元数据、协议等功能上对MySQL数据库有更好的兼容度，推荐使用。B模式由于架构限制无法很好的与MySQL兼容，后续不再演进，不推荐使用。
- 由于M-Compatibility的底层架构与MySQL存在差异，对于information_schema和m_schema下与MySQL名称相同的Schema（具体请参见《M-Compatibility开发指南》中的“Schema”章节），其查询性能可能存在差异。例如count函数无法做执行上的优化，表现为SELECT *和SELECT count(*)语句耗时近似。

Database 和 Schema 设计

MySQL的数据对象包括DATABASE、TABLE、INDEX、VIEW、TRIGGER、PROC等，MySQL的对象层次跟GaussDB的对应关系是从上至下且一对多包含关系。如下图所示：

图 1-1 MySQL 和 GaussDB 中 Database 和 Schema 之间的差异



- 在MySQL中Database和Schema是同义词；而在GaussDB中，一个Database 下可以有多个Schema。在该特性中，每个MySQL中的Database都被映射到GaussDB的一个Schema。
- 在MySQL中，INDEX从属于一个TABLE，但在GaussDB中，INDEX从属于一个Schema。这个差异导致INDEX名在GaussDB中要求在Schema内唯一，但在MySQL中仅要在在一个表内唯一。这个差异将作为当前约束予以保留。

2 MySQL 兼容性 B 模式

2.1 数据类型

2.1.1 数值数据类型

整数类型

除特别说明外，MySQL兼容性B模式中的数据类型精度、标度、位数大小等默认不支持用浮点型数值定义，建议使用合法的整型数值定义。

整数类型公共差异说明：

- 输入格式：
  - MySQL

    对于类似“asbd”、“12dd”、“12 12”等字符场景的输入，会采取截断或返回0值并上报WARNING处理，在严格模式时插表会失败。
  - GaussDB
    - 整数类型（TINYINT、SMALLINT、MEDIUMINT、INT、INTEGER、BIGINT）的输入，当非法字符串部分被截断时，如“12@3”，会直接截断并无提示信息，插表成功。
    - 当整数类型全部被截断（如“@123”）或字符串为空时，返回0，且插表成功。
- 操作符：
  - +、-、*操作符

    GaussDB：INT/INTEGER/SMALLINT/BIGINT在进行运算时，返回值为类型本身，不会向上提升类型，当返回值超范围时报错。

    MySQL：支持提升类型到BIGINT后计算。
  - |、&、^、~运算符

    GaussDB：在类型所占用BIT位中计算；GaussDB中^表示指数运算，如需使用异或运算符，使用#替换。

    MySQL：提升类型计算。

- 负数显式类型转换：

  GaussDB：宽松模式结果为0，严格模式报错。

  MySQL：依据其对应的二进制将最高位替换成数值位计算结果，例如(-1)::uint4 = 4294967295。

- 其他差异：

  INT[(M)]精度，MySQL控制格式化输出，GaussDB仅语法支持，不支持功能。

- 聚集函数：
  - variance：GaussDB表示样本方差，MySQL表示总体方差。
  - stddev：GaussDB表示样本标准差，MySQL表示总体标准差。
- 显示宽度：
  - 在为整型数字列指明宽度信息时，如果不同时指定ZEROFILL，则宽度信息在表结构描述中不显示。
  - INSERT语句插入字符类型字段，GaussDB统一补齐0后插入。
  - JOIN USING语句，涉及类型推导，MySQL默认第一张表列，GaussDB若结果为有符号类型则宽度信息失效，否则为第一张表字段宽度。
  - greatest/least、ifnull/if、case when/decode，MySQL不补齐0，GaussDB在类型及宽度信息一致时补齐0。
  - 作为函数/存储过程出入参、返回值时，MySQL支持功能、GaussDB语法不报错功能不支持。

GaussDB数据库和MySQL数据库整数类型具体差异请参见**表2-1**。

表 2-1 整数类型

MySQL数据库	GaussDB数据库	差异
BOOL	支持，存在差异	MySQL：BOOL/BOOLEAN类型实际映射为TINYINT类型。 GaussDB：支持BOOL，其中： • “真”值的有效文本值是：TRUE、't'、'true'、'y'、'yes'、'1' 、'TRUE'、true、'on'以及所有非0数值。 • “假”值的有效文本值是：FALSE、'f'、'false'、'n'、'no'、'0'、0、'FALSE'、false、'off'。 使用TRUE和FALSE是比较规范的用法（也是SQL兼容的用法）。
BOOLEAN	支持，存在差异	
TINYINT[(M)] [UNSIGNED]	支持，存在差异	详情请参见**整数类型公共差异说明**。
SMALLINT[(M)] [UNSIGNED]	支持，存在差异	详情请参见**整数类型公共差异说明**。

MySQL数据库	GaussDB数据库	差异
MEDIUMINT[(M)] [UNSIGNED]	支持，存在差异	MySQL存储MEDIUMINT数据需要3字节。 • 带符号的范围是-8,388,608 ~ +8,388,607。 • 无符号的范围是0 ~ +16,777,215。 GaussDB映射为INT类型，存储需要4字节。 • 带符号的范围是-2,147,483,648 ~ +2,147,483,647。 • 无符号的范围是0 ~ +4,294,967,295。 其他差异请参见**整数类型公共差异说明**。
INT[(M)] [UNSIGNED]	支持，存在差异	详情请参见**整数类型公共差异说明**。
INTEGER[(M)] [UNSIGNED]	支持，存在差异	详情请参见**整数类型公共差异说明**。
BIGINT[(M)] [UNSIGNED]	支持，存在差异	详情请参见**整数类型公共差异说明**。

任意精度类型

表 2-2 任意精度类型

MySQL数据库	GaussDB数据库	差异
DECIMAL[(M[,D])]	支持，存在差异	• 操作符：GaussDB中“^”表示指数运算，如需使用异或运算符，使用“#”替换；MySQL中“^”表示异或。 • 取值范围：精度M，标度D不支持浮点型数值输入，只支持整型数值输入。 • 输入格式：当字符串入参全部被截断时不会报错，如“@123”；只有被部分截断时才会报错，如“12@3”。
NUMERIC[(M[,D])]	支持，存在差异	
DEC[(M[,D])]	支持，存在差异	
FIXED[(M[,D])]	不支持	-

浮点类型

表 2-3 浮点类型

MySQL数据库	GaussDB数据库	差异
FLOAT[(M,D)]	支持，存在差异	• 分区表支持：FLOAT数据类型不支持KEY键值分区策略分区表。 • 操作符：GaussDB中“^”表示指数运算，如需使用异或运算符，使用“#”替换；MySQL中“^”表示异或。 • 取值范围：精度M，标度D不支持浮点型数值输入，只支持整型数值输入。 • 输出格式：对于非法入参一律报错ERROR，不会在sql_mode=''的宽松模式下报WARNING。
FLOAT(p)	支持，存在差异	• 分区表支持：FLOAT数据类型不支持KEY键值分区策略分区表。 • 操作符：数值类型使用^操作符，与MySQL不一致，GaussDB中^操作符为取指数运算。 • 取值范围：定义精度p时，仅支持使用合法的整型数据类型。 • 输出格式：对于非法入参一律报错ERROR，不会在sql_mode=''的宽松模式下报WARNING。
DOUBLE[(M,D)]	支持，存在差异	• 分区表支持：DOUBLE数据类型不支持KEY键值分区策略分区表。 • 操作符：GaussDB中“^”表示指数运算，如需使用异或运算符，使用“#”替换；MySQL中“^”表示异或。 • 取值范围：精度M，标度D不支持浮点型数值输入，只支持整型数值输入。 • 输出格式：对于非法入参一律报错ERROR，不会在sql_mode=''的宽松模式下报WARNING。
DOUBLE PRECISION[(M,D)]	支持，存在差异	• 操作符：GaussDB中“^”表示指数运算，如需使用异或运算符，使用“#”替换；MySQL中“^”表示异或。 • 取值范围：精度M，标度D不支持浮点型数值输入，只支持整型数值输入。 • 输出格式：对于非法入参一律报错ERROR，不会在sql_mode=''的宽松模式下报WARNING。

MySQL数据库	GaussDB数据库	差异
REAL[(M,D)]	支持，存在差异	● 分区表支持：REAL数据类型不支持KEY值分区策略分区表。 ● 操作符：GaussDB中“^”表示指数运算，如需使用异或运算符，使用“#”替换；MySQL中“^”表示异或。 ● 取值范围：精度M，标度D不支持浮点型数值输入，只支持整型数值输入。 ● 输出格式：对于非法入参一律报错ERROR，不会在sql_mode=''的宽松模式下报WARNING。

序列整数

表 2-4 序列整数

MySQL数据库	GaussDB数据库	差异
SERIAL	支持，存在差异	GaussDB中SERIAL具体介绍请参见《开发指南》手册中的“SQL参考 > 数据类型 > 数值类型”章节。 规格上与MySQL的差异如下： `CREATE TABLE test(f1 serial, f2 CHAR(20));` ● 类型定义的差异，MySQL的serial是映射到BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE，GaussDB的serial是映射到INTEGER NOT NULL DEFAULT nextval('test_f1_seq'::regclass)。如： `-- MySQL serial的定义：` `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)` `-- GaussDB serial的定义` `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` ● INSERT场景下serial类型DEFAULT值的差异。如： `-- MySQL插入serial的DEFAULT值` `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 |`

MySQL数据库	GaussDB数据库	差异

			11	aaaa |
				+----+------+
				3 rows in set (0.00 sec)
				
				-- GaussDB插入serial的DEFAULT值
				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)
								● REPLACE场景下serial类型引用列的差异，GaussDB引用列的介绍请参见《开发指南》手册中的“SQL参考 > SQL语法 > R > REPLACE”章节。如：
								-- MySQL插入serial引用列的值
								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)
		
		-- GaussDB插入serial引用列的值
		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)

2.1.2 日期与时间数据类型

表 2-5 日期与时间数据类型

MySQL数据库	GaussDB数据库	差异
DATE	支持，存在差异	GaussDB支持date数据类型，与MySQL相比规格上存在如下差异： ● 输入格式 – GaussDB只支持字符类型，不支持数值类型。如支持'2020-01-01'或'20200101'字符串格式，不支持20200101数值输入。MySQL支持数值输入转换为date类型。 – 分隔符：GaussDB不支持加号“+”、冒号“:”作为年、月、日之间的分隔符，其他符号都支持。MySQL所有符号均可作为分隔符。 分隔符混合使用的某些场景也不支持，与MySQL也有差异，如'2020-01>01'，'2020/01+01'等，不建议混合使用分隔符，建议使用最常用的“-”、“/”作为分隔符。 – 无分隔符：推荐使用完整格式，如'YYYYMMDD' 或者 'YYMMDD'。其他不完整的格式（包括超长格式）解析的规则与MySQL存在差异，可能报错或者解析的结果与MySQL不一致，不推荐使用。 ● 输出格式 GaussDB在sql_mode参数不包含'strict_trans_tables'选项（定义为宽松模式，否则为严格模式）时，允许年、月、日的值是0，但是输出时会按照年、月、日的顺序依次转换为合法的值，如date '0000-00-10' 转换为：0002-12-10 BC。 非法输入或者超过范围时，会报warning信息，并返回0000-00-00值。MySQL对于包含0值年、月、日的date值会原样输出。 ● 取值范围 GaussDB的范围是4713-01-01 BC ~ 5874897-12-31 AD，支持公元前的日期，宽松模式下超过范围时，返回的是0值：0000-00-00，严格模式下会报错。MySQL的范围是 0000-00-00 ~ 9999-12-31，宽松模式下超过范围后，各个场景下的表现并不一致，可能报错（如SELECT查询语句中），也可能返回0000-00-00值（如INSERT时）。此差异会导致date类型作为函数入参时，函数返回的结果存在差异。 ● 操作符

MySQL数据库	GaussDB数据库	差异

		– GaussDB仅支持date类型之间的比较操作符“=”、“!=”、“<”、“<=”、“>”、“>=”，返回true或者false；date与interval类型的加法运算，返回结果为date类型；date与interval类型的减法运算，返回结果为date类型；date类型之间的减法运算，返回结果为interval类型。
		– MySQL date类型和其他数值类型运算时，会先将date转换为数值类型，然后按照数值类型运算，结果也为数值类型。与GaussDB存在差异。如：
		`-- MySQL： date + 数值，先将date类型转换为数值20200101,再与1相加，结果为数值类型20200102。`
		`mysql> SELECT date'2020-01-01' + 1;`
		`+----------------------+`
		`| date'2020-01-01' + 1 |`
		`+----------------------+`
		`| 20200102 |`
		`+----------------------+`
		`1 row in set (0.00 sec)`
		
		`-- GaussDB： date + 数值，数值类型会转换为interval类型1 day，然后相加得到新的日期。`
		`gaussdb=# SELECT date'2020-01-01' + 1;`
		`?column?`
		`------------`
		`2020-01-02`
		`(1 row)`
		● 类型转换
		相比较MySQL，GaussDB仅支持date类型与char(n)、nchar(n)、datetime、timestamp类型之间的相互转换，不支持与binary、decimal、json、integer、unsigned integer、time 类型之间的转换。集合等场景和复杂表达式场景下公共类型的确定原则与MySQL也不一致，参考**数据类型转换**章节的描述。

MySQL数据库	GaussDB数据库	差异
DATETIME[(fsp)]	支持，存在差异	GaussDB支持datetime数据类型，与MySQL相比规格上存在如下差异： ● 输入格式： – GaussDB只支持字符类型，不支持数值类型。如支持'2020-01-01 10:20:30.123456'或'20200101102030.123456'字符串格式，不支持如20200101102030.123456的数值类型输入。MySQL支持数值输入转换为datetime类型。 – 分隔符：GaussDB不支持加号“+”、冒号“:”作为年、月、日之间的分隔符，其他的符号都支持。仅支持冒号“:”作为时、分、秒之间的分隔符，其他的符号都不支持。分隔符混合使用的某些场景也不支持，与MySQL也有差异，不推荐使用。MySQL支持所有符号作为分隔符。 – 无分隔符：GaussDB推荐使用完整格式'YYYYMMDDhhmiss.ffffff'。其他不完整的格式（包括超长格式）解析的规则可能与MySQL存在差异，可能报错或者解析的结果与MySQL不一致，不推荐使用。 ● 输出格式： – 统一为'YYYY-MM-DD hh:mi:ss.ffffff'的格式，格式与MySQL无差异，且不受DateStyle参数的影响。但是对于精度部分，如果最后几位为0，GaussDB不显示，MySQL会显示。 – GaussDB在sql_mode参数不包含'strict_trans_tables'选项（定义为宽松模式，否则为严格模式）时，允许年、月、日值是0，但是输出时会按照年、月、日的顺序依次转换为合法的值，如datetime '0000-00-10 00:00:00' 转换为：0002-12-10 00:00:00 BC。 非法输入或者超过范围时，会报warning信息，并返回0000-00-00 00:00:00值。MySQL对于包含0值年、月、日的datetime值会原样输出。 ● 取值范围 4713-11-24 00:00:00.000000 BC ~ 294277-01-09 04:00:54.775806 AD。294277-01-09 04:00:54.775807 AD 返回的是infinity。对于超过范围的值，严格模式下GaussDB会报错，MySQL是否报错取决于使用场景。一般查询场景不报错，而执行DML SQL语句更改表属性的值时报错。宽松模式下GaussDB返回0000-00-00 00:00:00值，MySQL根据使用场景可能报错，也可能返回

MySQL数据库	GaussDB数据库	差异

		0000-00-00 00:00:00值或者null值。这个差异会导致以datetime类型为入参的函数执行结果与MySQL也存在差异。
		● 精度
		范围0~6，作为表列的类型时缺省为0，与MySQL一致。对于 datetime[(p)] 'str' 表达式场景，GaussDB将(p)作为精度解析，缺省为6，将'str'按照p指定的精度格式化成datetime类型。MySQL不支持datetime[(p)] 'str'表达式。
		● 操作符
		– GaussDB仅支持datetime类型之间的比较操作符“=”、“!=”、“<”、“<=”、“>”、“>=”，返回true或者false；datetime与interval类型的加法运算，返回结果为datetime类型；datetime与interval类型的减法运算，返回结果为datetime类型；datetime类型之间的减法运算，返回结果为interval类型。
		– MySQL datetime类型和其他数值类型运算时，会先将datetime转换为数值类型，然后按照数值类型运算，结果也为数值类型。与GaussDB存在差异。如：
		-- MySQL： datetime + 数值，先将datetime类型转换为数值20201010123456,再与1相加，结果为数值类型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 + 数值，数值类型会转换为interval类型 1 day，然后相加得到新的datetime。
		gaussdb=# SELECT cast('2020-10-10 12:34:56.123456' AS datetime) + 1;
		?column?
		--------------------- 	
		2020-10-11 12:34:56
		(1 row)
		将datetime类型与数值的运算结果作为函数的入参，可能导致函数的结果与MySQL也存在差异。
		● 类型转换
		相比较MySQL，GaussDB仅支持datetime类型与char(n)、nchar(n)、timestamp类型之间的相互转换、datetime到date、time类型的转换（仅赋值和显式转换）。不支持与binary、decimal、json、integer、unsigned integer类

MySQL数据库	GaussDB数据库	差异
		型之间的转换。集合等场景和复杂表达式场景下公共类型的确定原则与MySQL也不一致，参考**数据类型转换**章节的描述。 ● 时区 GaussDB支持datetime值中携带时区信息（时区偏移或者时区名），如'2020-01-01 12:34:56.123456 +01:00' 或者 '2020-01-01 2:34:56.123456 CST'。GaussDB会将其转换为当前服务器时区的时间。MySQL不支持（5.7版本不支持，8.0及之后的版本支持）。 ● GaussDB的datetime数据类型的表字段实际上会被转换为timestamp(p) without time zone类型，查询表信息或者使用工具导出的表结构，其字段的数据类型显示的是timestamp(p) without time zone，而不是datetime。MySQL显示的是datetime(p)。

MySQL数据库	GaussDB数据库	差异
TIMESTAMP[(fsp)]	支持，存在差异	GaussDB支持timestamp数据类型，与MySQL相比规格上存在如下差异： ● 输入格式 – 只支持字符类型，不支持数值类型。如支持'2020-01-01 10:20:30.123456'或'20200101102030.123456'字符串格式，不支持如20200101102030.123456的数值类型输入。MySQL支持数值输入转换为timestamp类型。 – 分隔符：不支持加号“+”、冒号“:”作为年、月、日之间的分隔符，其他的符号都支持。仅支持冒号“:”作为时、分、秒之间的分隔符，其他的符号都不支持。分隔符混合使用的某些场景也不支持，与MySQL也有差异，不推荐使用。MySQL支持所有符号作为分隔符。 – 无分隔符：推荐使用完整格式'YYYYMMDDhhmiss.ffffff'。其他不完整的格式（包括超长格式）解析的规则可能与MySQL存在差异，可能报错或者解析的结果与MySQL不一致，不推荐使用。 ● 输出格式 – 统一为'YYYY-MM-DD hh:mi:ss.ffffff'的格式，格式与MySQL无差异，且不受DateStyle参数的影响。但是对于精度部分，如果最后几位为0，GaussDB不显示，MySQL会显示。 – GaussDB在sql_mode参数不包含'strict_trans_tables'选项（定义为宽松模式，否则为严格模式）时，允许年、月、日值是0，但是输出时会按照年、月、日的顺序依次转换为合法的值，如timestamp '0000-00-10 00:00:00' 转换为：0002-12-10 00:00:00 BC。 非法输入或者超过范围时，会报warning信息，并返回0000-00-00 00:00:00值。MySQL对于包含0值年、月、日的timestamp值会原样输出。 ● 取值范围 4713-11-24 00:00:00.000000 BC ~ 294277-01-09 04:00:54.775806 AD。294277-01-09 04:00:54.775807 AD 返回的是infinity。对于超过范围的值，严格模式下GaussDB会报错，MySQL是否报错取决于使用场景。一般查询场景不报错，而执行DML SQL语句更改表属性的值时报错。宽松模式下GaussDB返回0000-00-00 00:00:00值，MySQL根据使用场景可能报错，也可能返回

MySQL数据库	GaussDB数据库	差异
		0000-00-00 00:00:00值或者null值。这个差异会导致以timestamp类型为入参的函数执行结果与MySQL也存在差异。 ● 精度 范围0~6，作为表列的类型时缺省为0，与MySQL一致。对于 timestamp[(p)] 'str' 表达式场景： – GaussDB将(p)作为精度解析，缺省为6，将'str'按照p指定的精度格式化成timestamp类型。 – MySQL将timestamp 'str'的含义与GaussDB一致，缺省精度也为6。但是将timestamp(p) 'str'解析为函数调用，p作为timestamp函数的入参，结果返回一个timestamp类型的值，'str'作为投影列的别名。 ● 操作符 – GaussDB仅支持timestamp类型之间的比较操作符“=”、“!=”、“<”、“<=”、“>”、“>=”，返回true或者false；timestamp与interval类型的加法运算，返回结果为timestamp类型；timestamp与interval类型的减法运算，返回结果为timestamp类型；timestamp类型之间的减法运算，返回结果为interval类型。 – MySQL timestamp类型和其他数值类型运算时，会先将timestamp转换为数值类型，然后按照数值类型运算，结果也为数值类型。与GaussDB存在差异。如： -- MySQL： timestamp + 数值，先将timestamp类型转换为数值20201010123456.123456,再与1相加，结果为数值类型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 + 数值，数值类型会转换为interval类型 1 day，然后相加得到新的timestamp。 gaussdb=# SELECT timestamp '2020-10-10 12:34:56.123456' + 1; ?column? ---------------------------- 2020-10-11 12:34:56.123456 (1 row) 将timestamp类型与数值的运算结果作为函数的入参，可能导致函数的结果与MySQL也存在差异。

MySQL数据库	GaussDB数据库	差异
		• 类型转换 相比较MySQL，GaussDB仅支持timestamp类型与char(n)、varchar(n)、datetime类型之间的相互转换、timestamp到date、time类型的转换（仅赋值和显式转换）。不支持与binary、decimal、json、integer、unsigned integer类型之间的转换。集合等场景和复杂表达式场景下公共类型的确定原则与MySQL也不一致，参考**数据类型转换**章节的描述。 • 时区 GaussDB支持timestamp值中携带时区信息（时区偏移或者时区名），如'2020-01-01 12:34:56.123456 +01:00' 或者 '2020-01-01 2:34:56.123456 CST'。GaussDB会将其转换为当前服务器时区的时间。如果更改服务器时区，timestamp类型的值输出时会转换为更改后时区的时间戳。MySQL不支持（5.7版本不支持，8.0及之后的版本支持）。 • GaussDB的timestamp数据类型的表字段实际上会被转换为timestamp(p) with time zone类型，查询表信息或者使用工具导出的表结构，其字段的数据类型显示的是timestamp(p) with time zone，而不是timestamp。MySQL显示的是timestamp(p)。

MySQL数据库	GaussDB数据库	差异
TIME[(fsp)]	支持，存在差异	GaussDB支持time数据类型，与MySQL相比规格上存在如下差异： ● 输入格式 – 只支持字符类型，不支持数值类型。如支持'1 10:20:30'或'102030'字符串格式，不支持102030数值输入。MySQL支持数值输入转换为time类型。 – 分隔符：GaussDB仅支持冒号“:”作为时、分、秒之间的分隔符，其他的符号都不支持。MySQL支持所有的符号作为分隔符。 – 无分隔符：推荐使用完整格式，如'hhmiss.ffffff'。其他不完整的格式（包括超长格式）解析的规则可能与MySQL存在差异，可能报错或者解析的结果与MySQL不一致，不推荐使用。 – 分、秒、精度输入负数时，GaussDB数据库可能会忽略第一个负数开始的部分，涉及的部分解析为0，如：'00:00:-10' 解析结果为'00:00:00'。 也可能报错，如：'00:00:-10000' 会解析报错。取决于输入值的范围。而MySQL数据库统一报错。 ● 输出格式 统一为hh:mi:ss.ffffff的格式，格式与MySQL无差异。但是对于精度部分，如果最后几位为0，GaussDB不显示，MySQL会显示。 ● 取值范围 -838:59:59.000000 ~ 838:59:59.000000，与MySQL一致。对于超过范围的值，GaussDB在宽松模式下执行SELECT、INSERT、UPDATE等DML操作时，返回的值都是就近的边界值：-838:59:59或838:59:59。MySQL是查询时报错，DML操作返回的值才是就近边界值，场景上存在差异。此差异会导致time类型作为函数入参时，函数返回的结果也存在差异。 ● 精度 范围0~6，作为表列的类型时缺省为0，与MySQL一致。对于 time(p) 'str' 表达式场景，GaussDB将(p)作为精度解析，缺省为6，将'str'按照p指定的精度格式化成time类型。MySQL是解析为time函数，p是入参，'str'是投影列的别名。 ● 操作符 – GaussDB仅支持time类型之间的比较操作符“=”、“!=”、“<”、“<=”、“>”、“>=”，返回true或者false；time与

MySQL数据库	GaussDB数据库	差异
		interval类型的加法运算，返回结果为time类型；time与interval类型的减法运算，返回结果为time类型；time类型之间的减法运算，返回结果为interval类型。 – MySQL time类型和其他数值类型运算时，会先将time转换为数值类型，然后按照数值类型运算，结果也为数值类型。与GaussDB存在差异。如： -- MySQL： time + 数值，先将time类型转换为数值123456，再与1相加，结果为数值类型123457。 mysql> SELECT time '12:34:56' + 1; +---------------------+ | time '12:34:56' + 1 | +---------------------+ | 123457 | +---------------------+ 1 row in set (0.00 sec) -- GaussDB： time + 数值，数值类型会转换为interval类型1 day，然后相加得到新的time，由于是加了24小时，得到的仍然是12:34:56。 gaussdb=# SELECT time '12:34:56' + 1; ?column? ---------- 12:34:56 (1 row) 将time类型与数值的运算结果作为函数的入参，可能导致函数的结果与MySQL也存在差异。 ● 类型转换 相比较MySQL，GaussDB仅支持time类型与char(n)、nchar(n)类型之间的相互转换、datetime、timestamp到time类型的转换。不支持与binary、decimal、date、json、integer、unsigned integer类型之间的转换。集合等场景和复杂表达式场景下公共类型的确定原则与MySQL也不一致，参考**数据类型转换**章节的描述。

MySQL数据库	GaussDB数据库	差异
YEAR[(4)]	支持，存在差异	GaussDB支持year数据类型，与MySQL相比规格上存在如下差异： ● 操作符 – GaussDB仅支持year类型之间的比较操作符“=”、“!=”、“<”、“<=”、“>”、“>=”，返回true或者false。 – GaussDB仅支持year类型与int4类型之间的算术操作符“+”、“-”，返回整型值，MySQL是返回无符号整型值。 ● 类型转换 相比较MySQL，GaussDB仅支持year类型与int4类型的转换，仅支持int4、varchar、numeric、date、time、timestamp、timestamptz类型到year类型的转换。集合等场景和复杂表达式场景下公共类型的确定原则与MySQL也不一致，参考**数据类型转换**章节的描述。
INTERVAL	支持，存在差异	GaussDB支持INTERVAL数据类型，但INTERVAL在MySQL中为表达式，同时存在以下差异： ● 不支持字符串类型的日期输入作为运算，如：SELECT '2023-01-01' + interval 1 day。 ● 不支持interval expr unit语法中，expr为负整数或浮点数的输入，如：SELECT date'2023-01-01' + interval -1 day。 ● 不支持interval expr unit语法中，expr为运算表达式的输入，如：SELECT date'2023-01-01' + interval 4/2 day。 ● interval表达式参与运算时，返回值固定为datetime类型，MySQL为datetime或date类型。运算的逻辑与原有GaussDB保持一致，与MySQL有差异。 ● interval expr unit语法中，expr数值支持的范围会根据unit单位的不同有所差异，最大可支持的范围为[-2147483648, 2147483647]。超过范围时，严格模式报error，宽松模式报warning并返回0值。 ● interval expr unit语法中，expr指定的字段数量大于unit预期的字段数量时，在严格模式，报error；在宽松模式，报warning并返回0值。如unit取值为DAY_HOUR，预期的字段数量为2，expr取值为'1-2-3'，字段数量为3。

2.1.3 字符串数据类型

表 2-6 字符串数据类型

MySQL数据库	GaussDB数据库	差异
CHAR[(M)]	支持，存在差异	• 输入格式 – GaussDB自定义函数参数和返回值不支持长度校验，存储过程参数不支持长度校验，同时也不支持在PAD_CHAR_TO_FULL_LENGTH打开时补齐正确的空格，MySQL支持。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 • 语法 GaussDB的 Cast（expr as char）语法无法根据输入的字符串长度转成对应的类型，只支持转成varchar类型。不支持cast('' as char) 和 cast('' as char(0))将空串转成char(0)类型。MySQL支持按长度转成对应的类型。 • 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。

MySQL数据库	GaussDB数据库	差异
VARCHAR(M)	支持，存在差异	● 输入格式 – GaussDB的自定义函数参数和返回值不支持长度校验，存储过程参数不支持长度校验，MySQL支持。 – GaussDB的自定义函数和存储过程中的临时变量支持长度校验以及严格宽松模式下的报错和截断告警，MySQL不支持。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 ● 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。
TINYTEXT	支持，存在差异	● 输入格式 – GaussDB中该类型的长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过255字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 ● 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。

MySQL数据库	GaussDB数据库	差异
TEXT	支持，存在差异	● 输入格式 – GaussDB中该类型的长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过65535字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 ● 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。
MEDIUMTEXT	支持，存在差异	● 输入格式 – GaussDB中该类型的长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过16777215字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 ● 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。

MySQL数据库	GaussDB数据库	差异
LONGTEXT	支持，存在差异	● 输入格式 – GaussDB只支持不超过1GB的长度，MySQL支持4GB-1字节的长度。 – GaussDB不支持转义字符输入，不支持""双引号输入，MySQL支持。 ● 操作符 – GaussDB能正常转成浮点型的字符串与整型值进行加、减、乘、除、求余计算，返回值是整型值，MySQL是返回浮点型。 – GaussDB除以0会报错，MySQL返回null。 – “~”：GaussDB返回负数，MySQL返回8字节无符号整数。 – “^”：GaussDB表示次方幂，MySQL表示按位异或。
ENUM('value1','value2',...)	不支持	-
SET('value1','value2',...)	支持	-

2.1.4 二进制数据类型

表 2-7 二进制数据类型

MySQL数据库	GaussDB数据库	差异
BINARY[(M)]	不支持	-
VARBINARY(M)	不支持	-

MySQL数据库	GaussDB数据库	差异
TINYBLOB	支持，存在差异	● 取值范围：GaussDB中该类型由BYTEA类型映射得来，长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过255字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 ● 输入格式：不支持转义字符输入，不支持""双引号输入。 ● 输出格式：对于'\0'字符，查询结果表现为“\000”，使用JDBC驱动的getBytes接口获取表现为'\0'字符。 ● 操作符：不支持算数运算符“+”、“-”、“*”、“/”、“%”；不支持常用逻辑运算符或、与、非（“||”、“&&”、“!”）；不支持常用位运算符“~”、“&”、“|”、“^”。
BLOB	支持，存在差异	● 取值范围：GaussDB中该类型由BYTEA类型映射得来，长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过65535字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 ● 输入格式：不支持转义字符输入，不支持""双引号输入。 ● 输出格式：对于'\0'字符，查询结果表现为“\000”，使用JDBC驱动的getBytes接口获取表现为'\0'字符。 ● 操作符：不支持算数运算符“+”、“-”、“*”、“/”、“%”；不支持常用逻辑运算符或、与、非（“||”、“&&”、“!”）；不支持常用位运算符“~”、“&”、“|”、“^”。

MySQL数据库	GaussDB数据库	差异
MEDIUMBLOB	支持，存在差异	● 取值范围：GaussDB中该类型由BYTEA类型映射得来，长度不能超过1GB，超过长度限制后会报错。MySQL中该类型不能超过16777215字节，超过长度限制后，在严格模式下会报错，在宽松模式下会对数据进行截断并告警。 ● 输入格式：不支持转义字符输入，不支持""双引号输入。 ● 输出格式：对于'\0'字符，查询结果表现为“\000”，使用JDBC驱动的getBytes接口获取表现为'\0'字符。 ● 操作符：不支持算数运算符“+”、“-”、“*”、“/”、“%”；不支持常用逻辑运算符或、与、非（“||”、“&&”、“!”）；不支持常用位运算符“~”、“&”、“|”、“^”。
LONGBLOB	支持，存在差异	● 取值范围：GaussDB中该类型由BYTEA类型映射得来，只支持不超过1GB的长度，具体范围参照BYTEA数据类型集中式和分布式规格。 ● 输入格式：不支持转义字符输入，不支持""双引号输入。 ● 输出格式：对于'\0'字符，查询结果表现为“\000”，使用JDBC驱动的getBytes接口获取表现为'\0'字符。 ● 操作符：不支持算数运算符“+”、“-”、“*”、“/”、“%”；不支持常用逻辑运算符或、与、非（“||”、“&&”、“!”）；不支持常用位运算符“~”、“&”、“|”、“^”。
BIT[(M)]	不支持	-

2.1.5 JSON 数据类型

表 2-8 JSON 数据类型

MySQL数据库	GaussDB数据库	差异
JSON	支持，存在差异	• GaussDB数据库MySQL兼容性B模式中的JSON类型与GaussDB数据库原生的JSON类型行为一致，与MySQL行为差异较大，此处不再逐个列出。 • GaussDB数据库MySQL兼容性B模式中的JSON类型具体行为请参见《开发指南》中的“SQL参考 > 数据类型 > JSON/JSONB类型”章节。

2.1.6 数据类型支持的属性

表 2-9 数据类型支持的属性

MySQL数据库	GaussDB数据库
NULL	支持
NOT NULL	支持
DEFAULT	支持
ON UPDATE	支持
PRIMARY KEY	支持
AUTO_INCREMENT	支持
CHARACTER SET name	支持
COLLATE name	支持

2.1.7 数据类型转换

不同的数据类型之间支持转换。有如下场景涉及到数据类型转换：

- 操作符（比较操作符、运算操作符等）的操作数的数据类型不一致。常见于查询条件或者关联条件中的比较运算。
- 函数调用时实参和形参的数据类型不一致。
- DML语句要更新（包括INSERT、UPDATE、MERGE、REPLACE等）的目标列，数据的类型和列的定义类型不一致。
- 显式的类型转换：cast(expr as datatype)，将expr表达式类型转换为datatype类型。

- 集合运算（UNION、MINUS、EXCEPT、INTERSECT）确定最终投影列的目标数据类型后，各个SELECT查询的投影列的类型和目标数据类型不一致。
- 其他表达式计算场景，根据不同表达式的数据类型， 来决定用于比较或者最终结果的目标数据类型。
  - DECODE
  - CASE WHEN
  - lexpr [NOT] IN (expr_list)
  - BETWEEN AND
  - JOIN USING(a,b)
  - GREATEST和LEAST
  - NVL 和 COALESCE

GaussDB和MySQL数据库对于数据类型转换、转换的目标数据类型有着完全不同的规则。如下示例体现了两者处理的差异：

```
-- MySQL： in 执行结果为0，表示false。根据规则，会将'1970-01-01'与列表中的表达式依次比较，结果都为0，因此最终结果为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： in 执行结果为true，与MySQL结果相反：根据规则选定的公共类型为int类型，因此将左表达式'1970-01-01'转换为int类型与列表中的表达式转换为int类型后的值依次比较。
-- '1970-01-01'和'1970-01-02'转换为int类型时都为1970（MySQL兼容性下，转换时遇到非法字符后忽略，将前面部分转换为int类型），比较结果为相等，因此返回结果为true。
gaussdb=# SELECT '1970-01-01' IN ('1970-01-02', 1::int, '1970-01-02') AS result;
 result
--------
 t
(1 row)
```

数据类型转换规则的差异：

- GaussDB数据库对于不同数据类型之间的转换规则有明确的定义：
  - 是否支持转换：pg_cast系统表中是否定义两种类型的转换路径，没有定义则不支持。
  - 支持转换的场景：支持任意场景转换、仅支持显式（cast表达式）转换、仅支持赋值时转换。不支持的场景下即使定义了转换路径，也不能进行数据类型转换。
- MySQL数据库支持任意两种数据类型之间做转换。

由于存在以上差异，基于MySQL数据库的应用程序向GaussDB数据库迁移时，SQL语句可能由于不支持不同数据类型之间的转换而报错。或者支持转换的场景下，转换的规则有差异导致SQL语句执行的结果不同。

推荐的做法是：SQL语句中尽量使用相同的数据类型做比较或者赋值等操作，避免因为数据类型转换导致非预期结果或者性能损耗。

选择目标数据类型的规则差异：

对于有些场景，比较的数据类型或者返回的数据类型需要综合考虑多个表达式的类型才能确定。比如UNION运算中，不同SELECT语句中相同位置的投影列具有不同的数据

类型，查询结果的最终数据类型，需要由各个SELECT语句投影列的数据类型共同确定。

确定目标数据类型的规则，GaussDB数据库和MySQL数据库存在体系上的差异。

- GaussDB数据库规则：
  - 操作符的操作数类型不一致时，并不是将操作数的类型统一转换为目标类型再计算。而是直接注册两个数据类型的操作符，操作符处理中定义两个不同类型的处理规则。此方式不存在类型隐式转换，但自定义的处理规则隐含了转换的操作。
  - 集合运算和表达式场景，确定目标数据类型的规则：
    - 如果所有类型都相同，则此类型即为目标类型。
    - 两个数据类型如果不同，检查数据类型是否属于同一种类的数据类型，如数值类型、字符类型、日期时间类型等。不属于同一种类的数据类型，无法确定目标类型，此时SQL语句执行会报错。
    - 对于category属性（在pg_type系统表中定义）相同的数据类型，具有preferred属性（在pg_type系统表中定义）的数据类型会被选为目标类型。或者操作数1能转换为操作数2（没有转换路径），而操作数2无法转换为操作数1或数值类型优先级小于操作数2，则选择操作数2作为目标类型。
    - 如果涉及到3个及以上的数据类型，确定目标类型的规则为：common_type(type1,type2,type3) = common_type(common_type(type1,type2),type3)，依次迭代处理，得到最终的结果。
    - 对于IN和NOT IN表达式，如果根据以上规则无法确认目标类型，会将lexpr与expr_list中每一个表达式单独按照等值操作符（=）逐个比较。
    - 精度的确定：以最终选定的表达式的精度作为最终结果。
- MySQL数据库规则：
  - 操作符的操作数类型不一致时，先按照如下规则确定目标类型。确定后将类型不一致的操作数转换成目标类型后再做处理。
    - 两个参数都是string类型，则都按照string类型比较。
    - 两个参数都是integer类型，则都按照integer类型比较。
    - 十六进制数值如果不与数值比较，则当做二进制字符串比较。
    - 一个参数是datetime/timestamp类型，另一个参数是常量，将常量转换为时间戳类型然后比较。
    - 如果其中一个参数是decimal类型，比较时使用的数据类型取决于另外一个参数。另外一个是decimal或者integer类型时，按照decimal类型；另外一个是其他类型，按照real类型比较。
    - 其他场景都转换为 real 类型后比较。
  - 集合运算和表达式场景，确定目标数据类型的规则如下：
    - 建立任意两个类型之间的目标类型矩阵。给定两个类型，通过矩阵即可以确定目标类型。

- 如果涉及到3个及以上的数据类型，确定目标类型的规则为：common_type(type1,type2,type3) = common_type(common_type(type1,type2),type3)，依次迭代处理，得到最终的结果。
- 如果目标类型是integer类型，且各个表达式类型包含有符号和无符号的混合场景，则会将类型提升到更高精度的integer类型。符号的确定：所有表达式都是无符号时，结果才为无符号，否则结果为有符号。
- 精度确定：以表达式中的最大精度作为最终结果。

从以上规则可知：GaussDB和MySQL数据库在数据类型的转换规则上有很大差异，不能直接对比。在上述场景下，SQL语句的执行结果可能和MySQL数据库不一致。当前版本推荐各个表达式使用相同的类型，或提前使用cast转换成需要的类型来规避差异。

2.2 系统函数

2.2.1 流量控制函数

表 2-10 流量控制函数列表

MySQL数据库	GaussDB数据库	差异
IF()	支持，存在差异	• expr1入参仅支持bool类型。非bool类型入参若不能转换为bool类型则报错。 • 若expr2、expr3两个入参类型不同且两类型间不存在隐式转换函数则报错。 • 两个入参类型相同时，返回该入参类型。 • 若expr2、expr3两个入参类型分别为NUMERIC、STRING或TIME其中一个，GaussDB输出为TEXT类型，MySQL输出为VARCHAR类型。
IFNULL()	支持，存在差异	• 若expr1、expr2两个入参类型不同且两类型间不存在隐式转换函数则报错。 • 两个入参类型相同时，返回该入参类型。 • 若expr1、expr2两个入参类型范畴分别为NUMERIC、STRING或TIME其中一个，GaussDB输出为TEXT类型，MySQL输出为VARCHAR类型。 • 两个入参类型第一个入参为float4，另一个为bigint或unsigned bigint时返回double类型，MySQL返回float类型。

MySQL数据库	GaussDB数据库	差异

NULLIF()	支持，存在差异	● GaussDB中NULLIF()类型推导遵从以下逻辑：
		– 如果两个参数的数据类型不同，且两个入参类型存在等值比较操作符，则返回对应等值操作符对应的左值类型，否则会对两个入参类型进行强制类型兼容。
		– 若强制类型兼容后，存在等值比较操作符，则返回强制类型兼容后对应等值操作符的左值类型。
		– 若强制类型兼容后，仍找不到对应等值操作符，则报错。
		--两个入参类型存在等值比较操作符
		gaussdb=# SELECT pg_typeof(nullif(1::int2, 2::int8));
		pg_typeof
		----------- 	
		smallint
		(1 row)
		--两个入参类型不存在等值比较操作符，但在强制类型兼容后可以找到等值比较操作符
		gaussdb=# SELECT pg_typeof(nullif(1::int1, 2::int2));
		pg_typeof
		----------- 	
		bigint
		(1 row)
		
		--两个入参类型不存在等值比较操作符，且强制类型兼容后也不存在等值比较操作符
		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
		● MySQL输出类型仅与第一个入参类型有关：
		– 第一个入参为tinyint、smallint、mediumint、int、bool时，输出为int类型。
		– 第一个入参为bigint时，输出为bigint类型。
		– 第一个入参为unsigned tinyint、unsigned smallint、unsigned mediumint、unsigned int、bit时，输出为unsigned int类型。
		– 第一个入参为unsigned bigint时，输出为unsigned bigint。
		– 第一个入参为浮点型即float、double、real时，输出为double类型。
		– 第一个入参类型为decimal或numeric类型时，输出为decimal类型。
		– 第一个入参类型为时间类型或字符串类型即date、time、date、datetime、timestamp、char、varchar以及tinytext、enum、set时，输出为varchar类型。

MySQL数据库	GaussDB数据库	差异
		– 第一个入参类型为text、mediumtext、longtext时，输出为longtext类型。 – 第一个入参类型为tinyblob时，输出为varbinary类型。 – 第一个入参类型为mediumblob或longblob时，输出为longblob类型。 – 第一个入参为blob时，输出为blob类型。
ISNULL()	支持，存在差异	GaussDB中返回值为BOOLEAN类型的t或f，MySQL中返回值为INT类型的1或0。

2.2.2 日期和时间函数

以下为GaussDB数据库MySQL兼容性B模式中日期时间函数的公共说明，与MySQL行为一致。

- 函数入参为时间类型表达式的情况：

  时间类型表达式主要包括text、datetime、date或time，但所有可以隐式转换为时间表达式的类型都可以作为入参，比如数字类型可以通过先隐式转化为text，再作为时间类型表达式生效。

  但是，不同函数的具体生效情况会有所不同。例如：datediff函数仅计算日期之间的差值，因此时间表达式会被解析为日期；而timestampdiff函数在计算时间差值时，会根据unit参数来决定将时间表达式解析为date、time或 datetime。

- 函数入参为无效日期的情况：

  一般而言，日期时间函数支持date、datetime的范围和MySQL保持一致。date支持的范围为'0000-01-01'到'9999-12-31'，datetime支持的范围为'0000-01-01 00:00:00'到'9999-12-31 23:59:59'。虽然GaussDB支持的date、datetime范围大于MySQL，但是越界仍然算无效日期。

  大部分时间函数对于入参是无效日期时，会告警并返回NULL，只有能通过cast正常转换的日期，才是正常合理的日期。

- 函数入参的分隔符场景：

  对于时间函数，处理入参时会将所有非数字字符视作分隔符，然后根据数字所处的位置进行计算，推荐使用标准写法，年月日之间使用-分隔符，时分秒之间使用:分隔符，毫秒之前通过.来进行分隔。

  易错场景：在MySQL兼容性B模式数据库中执行“SELECT timestampdiff(hour, '2020-03-01 00:00:00', '2020-02-28 00:00:00+08');”时，时间函数不会自动计算时区，所以此处+08并未识别为时区，而是+作为分隔符当作秒来进行计算。

GaussDB的日期时间函数的大部分功能场景与MySQL一致，但仍有差异，部分差异如下：

- 函数入参为NULL时，函数返回NULL，无warning或error告警。这些函数包括：

  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、curtime

示例：

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

(1 row)
```

- 纯数字入参个别函数与MySQL有差异，不带引号的数字入参统一转成text入参来处理。

  示例：

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

  (1 row)
  ```

- 时间日期运算函数：adddate、subdate、date_add、date_sub。当运算后的结果为日期时，支持的范围为[0000-01-01, 9999-12-31]，当运算后的结果为日期时间时，支持的范围为[0000-01-01 00:00:00.000000, 9999-12-31 23:59:59.999999]，当运算后的结果超过支持的范围时，在严格模式下报error，在宽松模式下报warning。另外，当运算后的日期结果在范围[0000-01-01, 0001-01-01]中时，GaussDB正常返回结果，MySQL返回'0000-00-00'。

  示例：

  ```
  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)
  ```

- 对于日期和时间函数的date或datetime类型入参，含有0月或0日时为非法值，在严格模式下报error；在宽松模式，当输入为字符串或数字时，报warning，输入为date或datetime类型时视为上一年12月或上一月最后一日处理。

  对于cast函数，转换为date、datetime时，严格模式下会报error。宽松模式下不会报warning，而是视为上一年12月或上一月最后一日处理，需要注意此区别。MySQL对于包含0年、0月或0日的情况会原样输出。

  示例：

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

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

  (1 row)

  gaussdb=# SELECT adddate(date'2023-00-00', 1);-- 宽松模式
  ```



```
 adddate
------------
 2022-12-01
(1 row)

gaussdb=# SELECT cast('2023/00/00' AS date);-- 宽松模式
    date
------------
 2022-11-30
(1 row)

gaussdb=# SELECT cast('0000-00-00' AS datetime);-- 宽松模式
      timestamp
---------------------
 0000-00-00 00:00:00
(1 row)
```

- 若函数入参为numeric数据类型，在非法输入的情况下不会产生报错，会把入参当做0值处理。

  示例：

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

- 最多保留6位小数，不保留后置都为0的小数。

  示例：

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

- 时间函数参数为字符串时，只保证年月日之间使用“-”分隔，时分秒之间使用“:”分隔时结果正确。

  示例：

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

- 在MySQL中，当函数的返回值为varchar时，在GaussDB中，函数对应的返回值为text。

  ```
  -- GaussDB中函数的返回值。
  gaussdb=# SELECT pg_typeof(adddate('2023-01-01', 1));
   pg_typeof
  -----------
   text
  (1 row)

  -- MySQL中函数的返回值。
  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)
  ```


表 2-11 日期与和时间函数列表

MySQL数据库	GaussDB数据库	差异
ADDDATE()	支持，存在差异	此函数的表现会因为interval表达式的差异与MySQL有差异，具体可见**INTERVAL差异说明**。

MySQL数据库	GaussDB数据库	差异
ADDTIME()	支持，存在差异	• MySQL对第二入参为DATETIME样式字符串返回NULL，GaussDB可以计算。 • 入参取值范围为['0001-01-01 00:00:00', 9999-12-31 23:59:59.999999]。 • MySQL中ADDTIME函数如果第一个参数是动态参数（例如在预准备语句中），则返回类型为TIME。否则，函数的解析类型派生自第一个参数的解析类型。GaussDB中ADDTIME函数的返回值规则如下： – 第一个入参为date，第二个入参为date，返回值为time。 – 第一个入参为date，第二个入参为text，返回值为text。 – 第一个入参为date，第二个入参为datetime，返回值为time。 – 第一个入参为date，第二个入参为time，返回值为time。 – 第一个入参为text，第二个入参为date，返回值为text。 – 第一个入参为text，第二个入参为text，返回值为text。 – 第一个入参为text，第二个入参为datetime，返回值为text。 – 第一个入参为text，第二个入参为time，返回值为text。 – 第一个入参为datetime，第二个入参为date，返回值为datetime。 – 第一个入参为datetime，第二个入参为text，返回值为text。 – 第一个入参为datetime，第二个入参为datetime，返回值为datetime。 – 第一个入参为datetime，第二个入参为time，返回值为datetime。 – 第一个入参为time，第二个入参为date，返回值为time。 – 第一个入参为time，第二个入参为text，返回值为text。 – 第一个入参为time，第二个入参为datetime，返回值为time。 – 第一个入参为time，第二个入参为time，返回值为time。
CONVERT_TZ()	支持	-

MySQL数据库	GaussDB数据库	差异
CURDATE()	支持	-
CURRENT_DATE(), CURRENT_DATE	支持	-
CURRENT_TIME(), CURRENT_TIME	支持，存在差异	GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示。GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，其他均报错；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	支持，存在差异	GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示。GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，超过6的整型值，会告警并按照精度6输出时间值；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
CURTIME()	支持，存在差异	GaussDB此函数输入字符串或者非整型值，会被隐式转成整型，然后再校验精度，[0,6]范围之外的会报错，范围之内会正常输出时间值；MySQL直接报错。GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示，GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，其他均报错；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
YEARWEEK()	支持	-
DATE_ADD()	支持，存在差异	此函数的表现会因为interval表达式的差异与MySQL有差异，具体可见**INTERVAL差异说明**。
DATE_FORMAT()	支持	-
DATE_SUB()	支持，存在差异	此函数的表现会因为interval表达式的差异与MySQL有差异，具体可见**INTERVAL差异说明**。
DATEDIFF()	支持	-
DAY()	支持	-
DAYNAME()	支持	-

MySQL数据库	GaussDB数据库	差异
DAYOFMONTH()	支持	-
DAYOFWEEK()	支持	-
DAYOFYEAR()	支持	-
EXTRACT()	支持	-
FROM_DAYS()	支持	-
FROM_UNIXTIME()	支持	-
GET_FORMAT()	支持	-
HOUR()	支持	-
LAST_DAY	支持	-
LOCALTIME(), LOCALTIME	支持，存在差异	GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示。GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，其他整型值直接报错；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
LOCALTIMESTAMP, LOCALTIMESTAMP()	支持，存在差异	GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示。GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，超过6的整型值，会告警并按照精度6输出时间值；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
MAKEDATE()	支持	-
MAKETIME()	支持，存在差异	与MySQL相比, 入参为NULL时，GaussDB不支持maketime函数自嵌套，MySQL支持。
MICROSECOND()	支持	-
MINUTE()	支持	-
MONTH()	支持	-

MySQL数据库	GaussDB数据库	差异
MONTHNAME()	支持	-
NOW()	支持，存在差异	GaussDB的按精度输出的时间值（小数点后的值）是四舍五入的；MySQL是直接截断的。GaussDB按精度输出的时间值（小数点后的值）末尾0都不显示；MySQL会显示。GaussDB只支持输入[0,6]范围内的整型值，作为返回时间的精度，超过6的整型值，会告警并按照精度6输出时间值；MySQL的精度值有效值是[0,6]，但是输入的整型值内部会对256求余（例257，会返回精度1的时间值）。
PERIOD_ADD()	支持，存在差异	当入参period或结果小于0时，GaussDB参考MySQL 8.0.x版本的表现，报错处理。MySQL 5.7会发生整数回绕，导致计算结果异常。
PERIOD_DIFF()	支持，存在差异	当入参或结果小于0时，GaussDB参考MySQL 8.0.x版本的表现，报错处理。MySQL 5.7会发生整数回绕，导致计算结果异常。
QUARTER()	支持	-
SEC_TO_TIME()	支持	-
SECOND()	支持	-
STR_TO_DATE()	支持，存在差异	返回值与MySQL有差异，GaussDB返回的是text，MySQL返回的是datetime、date。
SUBDATE()	支持，存在差异	此函数的表现会因为interval表达式的差异与MySQL有差异，具体可见**INTERVAL差异说明**。

MySQL数据库	GaussDB数据库	差异
SUBTIME()	支持，存在差异	• MySQL对第二入参为DATETIME样式字符串返回NULL，GaussDB可以计算。 • 入参取值范围为['0001-01-01 00:00:00', 9999-12-31 23:59:59.999999]。 • MySQL中SUBTIME函数如果第一个参数是动态参数（例如在预准备语句中），则返回类型为TIME。否则，函数的解析类型派生自第一个参数的解析类型。GaussDB中SUBTIME函数的返回值规则如下： – 第一个入参为date，第二个入参为date，返回值为time。 – 第一个入参为date，第二个入参为text，返回值为text。 – 第一个入参为date，第二个入参为datetime，返回值为time。 – 第一个入参为date，第二个入参为time，返回值为time。 – 第一个入参为text，第二个入参为date，返回值为text。 – 第一个入参为text，第二个入参为text，返回值为text。 – 第一个入参为text，第二个入参为datetime，返回值为text。 – 第一个入参为text，第二个入参为time，返回值为text。 – 第一个入参为datetime，第二个入参为date，返回值为datetime。 – 第一个入参为datetime，第二个入参为text，返回值为text。 – 第一个入参为datetime，第二个入参为datetime，返回值为datetime。 – 第一个入参为datetime，第二个入参为time，返回值为datetime。 – 第一个入参为time，第二个入参为date，返回值为time。 – 第一个入参为time，第二个入参为text，返回值为text。 – 第一个入参为time，第二个入参为datetime，返回值为time。 – 第一个入参为time，第二个入参为time，返回值为time。
SYSDATE()	支持，存在差异	MySQL入参整型值会按照一字节最大值255整数回绕，GaussDB不回绕。

MySQL数据库	GaussDB数据库	差异
YEAR()	支持	-
TIME_FORMAT()	支持	-
TIME_TO_SEC()	支持	-
TIMEDIFF()	支持	-
WEEKOFYEAR()	支持	-
TIMESTAMPADD()	支持	-
TIMESTAMPDIFF()	支持	-
TO_DAYS()	支持	-
TO_SECONDS()	支持	-
UNIX_TIMESTAMP()	支持，存在差异	返回值与MySQL有差异，GaussDB返回的是numeric，MySQL返回的是int。
UTC_DATE()	支持，存在差异	• MySQL支持无括号调用，GaussDB不支持。MySQL入参整型值会按照一字节最大值255整数回绕。 • MySQL入参只支持0-6整数， GaussDB支持可以隐式转为0-6的输入。
UTC_TIME()	支持，存在差异	
UTC_TIMESTAMP()	支持，存在差异	
WEEK()	支持	-
WEEKDAY()	支持	-

2.2.3 字符串函数

表 2-12 字符串函数列表

MySQL数据库	GaussDB数据库	差异
BIN()	支持，存在差异	函数入参支持类型存在差异，GaussDB入参支持类型如下： ● 整数类型：tinyint、smallint、mediumint、int、bigint ● 无符号整数类型：tinyint unsigned、smallint unsigned、int unsigned、bigint unsigned ● 字符和文本类型：char、varchar、tinytext、text、mediumtext、longtext，仅支持纯数字整数字符串，且整数范围在bigint范围内。 ● 浮点类型：float、real、double ● 定点类型：numeric、decimal、dec ● 布尔类型：bool
CONCAT()	支持，存在差异	无论参数的数据类型如何，concat返回值的数据类型始终为text；MySQL的concat在含有二进制类型参数时，返回值为二进制类型。
CONCAT_WS()	支持，存在差异	无论参数的数据类型如何，concat_ws返回值的数据类型始终为text；MySQL的concat_ws在含有二进制类型参数时，返回值为二进制类型，其他情况返回值为字符串类型。

MySQL数据库	GaussDB数据库	差异
ELT()	支持，存在差异	• 函数入参1支持类型存在差异，GaussDB入参1支持类型如下： – 整数类型：tinyint、smallint、mediumint、int、bigint – 无符号整数类型：tinyint unsigned、smallint unsigned、int unsigned – 字符和文本类型：char、varchar、tinytext、text、mediumtext、longtext，仅支持纯数字整数字符串，且整数范围在bigint范围内。 – 浮点类型：float、real、double – 定点类型：numeric、decimal、dec – 布尔类型：bool • 函数入参2支持类型存在差异，GaussDB入参2支持类型如下： – 整数类型：tinyint、smallint、mediumint、int、bigint – 无符号整数类型：tinyint unsigned、smallint unsigned、int unsigned、bigint unsigned – 字符和文本类型：char、varchar、tinytext、text、mediumtext、longtext – 浮点类型：float、real、double – 定点类型：numeric、decimal、dec – 布尔类型：bool – 大对象类型：tinyblob、blob、mediumblob、longblob – 日期类型：datetime、timestamp、date、time
FIELD()	支持，存在差异	函数入参为在bigint最大值~ bigint unsigned最大值范围内的数字，存在不兼容。 函数入参为浮点型float(m, d)、double(m, d)、real(m, d)时精度更高，存在不兼容。
FIND_IN_SET()	支持，存在差异	当数据库encoding = 'SQL_ASCII'时，不支持默认的大小写判断规则，即在用户不指定字符集规则的情况下，大写与小写区分判断。

MySQL数据库	GaussDB数据库	差异
INSERT()	支持，存在差异	• Int64类型传参有范围限制，一旦超出-9223372036854775808~9223372036854775807范围会直接报错，MySQL对数值类型传参范围无限制，异常会告警按照上限或下限数值处理。 • 字符串传参有限制，入参text类型字符串长度最大为2^30-5字节，入参bytea类型字符串长度最大为2^30-512字节。 • s1和s2任意参数为bytea类型时，涉及到结果出现非法字符的情况可能展示结果与MySQL有差异但是字符编码与MySQL是一致的。
LOCATE()	支持，存在差异	入参1为bytea类型，入参2为text类型时，GaussDB与MySQL行为存在差异。
MAKE_SET()	支持，存在差异	• bits参数为整型时，最大范围支持到int128，低于MySQL范围。 • bits参数为日期类型datetime、timestamp、date、time，由于时间类型转整型与MySQL存在差异，目前均未做支持。 • bit类型或bool类型由于此类数据类型GaussDB与MySQL存在差异，返回结果导致的差异为GaussDB与MySQL固有差异。bits入参为bool类型，str入参为bit类型与bool类型均不做支持。 • bits入参为字符串或文本类型时，仅支持纯整型数字形式，其他形式存在差异。且纯整型数字范围限制在bigint范围。 • str入参整型数值超过正负81个9，返回值与MySQL有差异。 • str入参当以科学计数法表示时，GaussDB末尾0值会显示，MySQL不显示，以科学计数法打印，此为固有差异。

MySQL数据库	GaussDB数据库	差异
QUOTE()	支持，存在差异	● 已知str字符串中含有“\Z”，“\r”，“\%”，“_”，GaussDB未进行转义，与MySQL存在差异。斜线后跟部分数字也会引起差异，如“\563”。由转义字符引起的本函数与MySQL的差异，此为GaussDB与MySQL的转义字符差异。 ● str字符串中的“\b”，输出结果表现形式与MySQL有差异。此为GaussDB与MySQL的固有差异 ● str字符串中含有“\0”时，GaussDB由于UTF-8字符集不识别该字符，输入不成功。此为GaussDB与MySQL的固有差异 ● str为bit或bool类型时，由于GaussDB与MySQL此类型目前有差异，暂不支持此类类型。 ● GaussDB最大支持1GB数据传输，str入参长度最大支持536870908字节，函数返回结果字符串最大支持1GB。 ● str入参整型数值超过正负81个9，返回值与MySQL有差异。 ● str入参当以科学计数法表示时，GaussDB末尾0值会显示，MySQL不显示，以科学计数法打印，此为固有差异。
SPACE()	支持，存在差异	● GaussDB入参最大支持1073741818字节，超出返回空字符串。MySQL的入参默认最大支持4194304字节，超出告警。 ● 函数入参支持类型存在差异，GaussDB入参支持类型如下： – 整数类型：tinyint、smallint、mediumint、int、bigint – 无符号整数类型：tinyint unsigned、smallint unsigned、int unsigned – 字符和文本类型：char、varchar、tinytext、text、mediumtext、longtext，仅支持纯数字整数字符串，且整数范围在bigint范围内。 – 浮点类型：float、real、double – 定点类型：numeric、decimal、dec – 布尔类型：bool
SUBSTR()	支持	-
SUBSTRING()	支持	-
SUBSTRING_INDEX()	支持	-

MySQL数据库	GaussDB数据库	差异
STRCMP()	支持，存在差异	• 函数入参支持类型存在差异，GaussDB入参支持类型如下： – 字符类型：char、varchar、nvarchar2、text – 二进制类型：bytea – 数值类型：tinying [unsigned]、smallint [unsigned]、integer [unsigned]、bigint [unsigned]、float4、float8、numeric – 日期时间类型：date、time without time zone、datetime、timestamptz • 对于数值类型中的浮点类型，由于连接参数设置不同，精度可能与M有差异，不建议使用该场景，或使用NUMERIC类型代替。
SHA() / SHA1()	支持	-
SHA2()	支持	-

2.2.4 强制转换函数

表 2-13 强制转换函数列表

MySQL数据库	GaussDB数据库	差异
CAST()	支持，存在差异	数据类型转换规则和支持的转换类型均以GaussDB支持的转换范围和规则为准。
CONVERT()	支持，存在差异	数据类型转换规则和支持的转换类型均以GaussDB支持的转换范围和规则为准。

2.2.5 加密函数

表 2-14 加密函数列表

MySQL数据库	GaussDB数据库	差异
AES_DECRYPT()	支持	-
AES_ENCRYPT()	支持	-

2.2.6 信息函数

表 2-15 信息函数列表

MySQL数据库	GaussDB数据库	差异
LAST_INSERT_ID()	支持	-

2.2.7 JSON 函数

JSON函数差异说明：

- 对于JSON函数和其他字符入参函数，如果输入中包含转义字符，默认情况下会与MySQL有一定差异。要实现与MySQL的兼容，需要设置GUC参数standard_conforming_strings取值为off，在这种情况下，转义字符的处理将与MySQL兼容，但是会产生非标准字符输入的warning告警，转义字符\t、\u以及转义数字与MySQL存在差异。JSON_UNQUOTE()函数已经做了兼容处理，即使不设置 GUC 参数，也能与 MySQL 兼容，并且不会产生告警。
- 在处理超长数字（数字的字符长度超过64）时，GaussDB的JSON函数会将数字解析为一个double处理，并使用科学计数法计数。和MySQL的非JSON类型入参相同。但是在JSON类型入参时，由于JSON类型未完全与MySQL兼容，此场景下会产生差异。MySQL会完整显示数字（并且当数字长度超过82时，MySQL会给出错误的结果），GaussDB依然将超长数字解析为一个double精度的值。考虑到超长数字内部都是使用浮点数进行储存，进行运算时无论GaussDB还是MySQL都会有精度丢失，建议您使用字符串来储存超长数字。

```
gaussdb=# SELECT json_insert('[1, 4,
99999999999999999999999999999999999999999999999999999999999999999999999999999999999]','$[6]',json
_insert('[1,4]','$[5]',99999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999));
        json_insert
------------------------------
 [1, 4, 1e+74, [1, 4, 1e+74]]
(1 row)
```

表 2-16 JSON 函数列表

MySQL数据库	GaussDB数据库	差异
JSON_APPEND()	支持	-
JSON_ARRAY()	支持	-
JSON_ARRAY_APPEND()	支持	-
JSON_ARRAY_INSERT()	支持	-

MySQL数据库	GaussDB数据库	差异
JSON_CONTAINS()	支持	-
JSON_CONTAINS_PATH()	支持	-
JSON_DEPTH()	支持	-
JSON_EXTRACT()	支持	-
JSON_INSERT()	支持	-
JSON_KEYS()	支持	-
JSON_LENGTH()	支持	-
JSON_MERGE()	支持	-
JSON_OBJECT()	支持	-
JSON_QUOTE()	支持	-
JSON_REMOVE()	支持	-
JSON_REPLACE()	支持	-
JSON_SEARCH()	支持，存在差异	返回值与MySQL有差异，GaussDB返回的是text，MySQL返回的是JSON。
JSON_SET()	支持	-
JSON_TYPE()	支持	-
JSON_UNQUOTE()	支持	-
JSON_VALID()	支持	-

2.2.8 聚合函数

表 2-17 聚合函数列表

MySQL数据库	GaussDB数据库	差异
GROUP_CONCAT()	支持，存在差异	● 当group_concat参数中同时有DISTINCT和ORDER BY语法时，所有ORDER BY后的表达式必须也在DISTINCT的表达式之中。 ● group_concat(... order by 数字)不代表按照第几个参数的顺序，数字只是一个常量表达式，相当于不排序。 ● 无论入参的数据类型是什么，group_concat返回值的数据类型始终为text；MySQL的group_concat在含有二进制类型参数时，返回值为二进制类型，其他情况返回值为字符串类型，并且返回值长度大于512时，其数据类型为字符串大对象或二进制大对象。 ● GUC参数group_concat_max_len有效范围是0-1073741823，最大值比MySQL小。
DEFAULT()	支持，存在差异	● 字段默认值为数组形式，GaussDB返回数组形式，MySQL不支持数组类型。 ● GaussDB字段是隐藏列（比如xmin、cmin），default函数返回空值。 ● GaussDB支持分区表、临时表、多表连接查询默认值。 ● GaussDB支持查询列名包含字符串值节点（表示名称）和A_Star节点（表示出现“*”），如default(tt.t4.id)和default(tt.t4.*)。不合法的查询列名和A_Star节点，GaussDB和MySQL报错信息有差异。 ● GaussDB创建字段默认值，没有检验字段类型的范围，使用default函数可能报错。 ● 字段的默认值是函数表达式时，GaussDB的default函数返回建表时字段的default表达式的计算值。MySQL的default函数返回NULL。

2.2.9 数字操作函数

表 2-18 数字操作函数列表

MySQL数据库	GaussDB数据库	差异
log2()	支持，存在差异	• 小数位显示与MySQL存在差异，受GaussDB浮点数据类型限制，可通过参数extra_float_digits控制小数位个数显示； • 由于输入精度内部处理差异，GaussDB与MySQL会存在结果计算差异； • 支持数据类型有： – 整数类型：bigint、int16、int、smallint 、tinyint。 – 无符号整数类型：bigint unsigned、integer unsigned、smallint unsigned、tinyint unsigned。 – 浮点数类型：numeric、real。 – 字符串类型：character、character varying、clob、text，仅支持纯数字整数字符串。 – set类型。 – NULL空类型。
log10()	支持，存在差异	• 小数位显示与MySQL存在差异，受GaussDB浮点数据类型限制，可通过参数extra_float_digits控制小数位个数显示； • 由于输入精度内部处理差异，GaussDB与MySQL会存在结果计算差异； • 支持数据类型有： – 整数类型：bigint、int16、int、smallint 、tinyint。 – 无符号整数类型：bigint unsigned、integer unsigned、smallint unsigned、tinyint unsigned。 – 浮点数类型：numeric、real。 – 字符串类型：character、character varying、clob、text，仅支持纯数字整数字符串。 – set类型。 – NULL空类型。

2.2.10 其他函数

表 2-19 其他函数列表

MySQL数据库	GaussDB数据库	差异
UUID()	支持	-
UUID_SHORT()	支持	-

2.3 操作符

GaussDB数据库兼容绝大多数MySQL的操作符，但存在部分差异。除特别说明外，MySQL兼容性B模式中的操作符行为默认为GaussDB原生行为。

表 2-20 操作符

MySQL数据库	GaussDB数据库	差异
安全等于（<=>）	支持	-

MySQL数据库	GaussDB数据库	差异
[NOT] REGEXP	支持，存在差异	• 当设置GUC参数b_format_dev_version='s2'时，模式字符串pattern中有“\\a”、“\\d”、“\\e”、“\\n”、“\\Z”、“\\u”等转义字符时，匹配源字符串“\a”、“\d”、“\e”、“\n”、“\Z”、“\u”时，GaussDB行为与MySQL 5.7不一致，与MySQL 8.0一致。 • 当设置GUC参数b_format_dev_version='s2'时，GaussDB中“\b”可以与“\\b”匹配，MySQL会匹配失败。 • 模式字符串pattern非法入参，只存在右单括号“)”时，GaussDB会报错，MySQL 5.7会报错，MySQL 8.0不会报错。 • 在de|abc匹配序列de或abc的匹配规则，当|左右存在空值时，GaussDB不会报错，MySQL 5.7会报错，MySQL 8.0不会报错。 • 制表符“\t”正则匹配字符类[:blank:]，GaussDB可匹配，MySQL 5.7不能匹配，MySQL 8.0可匹配。 • GaussDB支持非贪婪模式匹配，即尽可能少的匹配字符，在部分特殊字符后加“?”问号字符，例如：“??, *?, +?, {n}?, {n,}?, {n,m}?”。MySQL 5.7版本不支持非贪婪模式匹配，并报错：Got error 'repetition-operator operand invalid' from regexp。MySQL 8.0版本已经支持。 • 在binary字符集下，text类型、blob类型均会转换成bytea类型，由于REGEXP操作符不支持bytea类型，因此无法匹配。
[NOT] RLIKE	支持，存在差异	同[NOT] REGEXP。

2.4 字符集

GaussDB数据库支持指定数据库、模式、表或列的字符集，支持的范围如下。

表 2-21 字符集列表

MySQL数据库	GaussDB数据库
utf8mb4	支持
gbk	支持
gb18030	支持

MySQL数据库	GaussDB数据库
utf8	支持
binary	支持

2.5 排序规则

GaussDB数据库支持指定库、模式、表或列的排序规则，支持的范围如下。

说明

排序规则差异说明：

- 当前仅有字符串类型、部分二进制类型支持指定排序规则，其他类型不支持指定排序规则，可以通过查询pg_type系统表中类型的typcollation属性不为0来判断该类型支持字符序。MySQL中所有类型可以指定字符序，但除字符串、二进制类型其他排序规则无实际意义。
- 当前排序规则（除binary外）仅支持在其对应字符集与库级字符集一致时可以指定，GaussDB数据库中，字符集必须与数据库的字符集一致，且不支持表内多种字符集混合使用。
- utf8mb4字符集下默认字符序为utf8mb4_general_ci，与MySQL 5.7保持一致。
- GaussDB中utf8和utf8mb4为同一个字符集。

表 2-22 排序规则列表

MySQL数据库	GaussDB数据库
utf8mb4_general_ci	支持
utf8mb4_unicode_ci	支持
utf8mb4_bin	支持
gbk_chinese_ci	支持
gbk_bin	支持
gb18030_chinese_ci	支持
gb18030_bin	支持
binary	支持
utf8mb4_0900_ai_ci	支持
utf8_general_ci	支持
utf8_bin	支持

2.6 表达式

GaussDB数据库兼容绝大多数MySQL的表达式，但存在部分差异。如未列出，表达式行为默认为GaussDB原生行为。

表 2-23 表达式

MySQL数据库	GaussDB数据库
用户自定义变量@var_name	部分支持
全局变量@@var_name	部分支持

2.7 SQL

2.7.1 DDL

表 2-24 DDL 语法兼容介绍

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
建表和修改表时支持创建主键、UNIQUE索引、外键约束	ALTER TABLE、CREATE TABLE	• GaussDB当前不支持UNIQUE INDEX|KEY index_name语法，使用UNIQUE INDEX|KEY index_name语法时会报错。 • 当约束被建立为全局二级索引，SQL语句中指定using btree时，底层会建立为ubtree。 • 当约束关联的表为ustore，且SQL语句中指定为using btree时，底层会建立为ubtree。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
支持自增列	ALTER TABLE、CREATE TABLE	• 自动增长列建议为索引（非全局二级索引）的第一个字段，否则建表时产生警告，含有自动增长列的表进行某些操作时会产生错误，例如：ALTERT TABLE EXCHANGE PARTITION。MySQL自动增长列必须为索引第一个字段。 • AUTO_INCREMENT = value语法，value必须为小于2^127的正数。MySQL不校验value。 • 当自增值已经达到字段数据类型的最大值时，继续自增将产生错误。MySQL有些场景产生错误或警告，有些场景仍自增为最大值。 • 不支持innodb_autoinc_lock_mode系统变量，GaussDB的GUC参数auto_increment_cache=0时，批量插入自动增长列的行为与MySQL系统变量innodb_autoinc_lock_mode=1相似。 • 自动增长列在导入数据或者进行Batch Insert执行计划的插入操作时，对于混合0、NULL和确定值的场景，如果产生错误，后续插入自增值不一定与MySQL完全一致。提供auto_increment_cache参数，可以控制预留自增值的数量。 • 并行导入或插入自动增长列触发自增时，每个并行线程预留的缓存值也只在其线程中使用，未完全使用完毕的话，也会出现表中自动增长列的值不连续的情况。并行插入产生的自增值结果无法保证与MySQL完全一致。 • 本地临时表中的自动增长列批量插入时不会预留自增

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
		值，正常场景不会产生不连续的自增值。MySQL临时表与普通表中的自动增长列自增结果一致。 ● SERIAL数据类型为原有的自增列，与AUTO_INCREMENT自增列有差异。MySQL的SERIAL数据类型就是AUTO_INCREMENT自增列。 ● 不允许auto_increment_offset的值大于auto_increment_increment的值，会产生错误。MySQL允许，并说明auto_increment_offset会被忽略。 ● 在表有主键或索引的情况下，ALTER TABLE命令重写表数据的顺序与MySQL不一定相同，GaussDB按照表数据存储顺序重写，MySQL会按主键或索引顺序重写，导致自增值的顺序可能不同。 ● ALTER TABLE命令添加或修改自增列时，第一次预留自增值的数量是表统计信息中的行数，统计信息的行数不一定与MySQL一致。 ● last_insert_id函数返回值为128位的整型。 ● 在触发器或用户自定义函数中自增时，刷新last_insert_id返回值。MySQL不刷新。 ● 对GUC参数auto_increment_offset和auto_increment_increment设置超出范围的值会产生错误。MySQL会自动改为边界值。 ● sql_mode设置no_auto_value_on_zero参

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
		数，表定义的自动增长列为非NOT NULL约束，向表中插入数据不指定自动增长列的值时，GaussDB中自动增长列插入NULL值，且不触发自增；MySQL中自动增长列插入NULL值，触发自增。
支持前缀索引	CREATE INDEX、ALTER TABLE、CREATE TABLE	• 前缀长度不得超过2676，键值的实际长度受内部页面限制，若字段中含有多字节字符或者一个索引上有多个键，索引行长度可能会超限报错。 • CREATE INDEX语法中，不支持以下关键字作为前缀键的字段名称：COALESCE、EXTRACT、GREATEST、LEAST、LNNVL、NULLIF、NVL、NVL2、OVERLAY、POSITION、REGEXP_LIKE、SUBSTRING、TIMESTAMPDIFF、TREAT、TRIM、XMLCONCAT、XMLELEMENT、XMLEXISTS、XMLFOREST、XMLPARSE、XMLPI、XMLROOT、XMLSERIALIZE。 • 主键索引中不支持前缀键。
支持指定字符集与排序规则	ALTER SCHEMA、ALTER TABLE、 CREATE SCHEMA、CREATE TABLE	-
修改表时支持在表第一列前面或者在指定列后面添加列	ALTER TABLE	-
修改列名称/定义语法兼容	ALTER TABLE	-

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
定时任务EVENT语法兼容	ALTER EVENT、CREATE EVENT、DROP EVENT、 SHOW EVENTS	-
创建分区表语法兼容	CREATE TABLE PARTITION、CREATE TABLE SUBPARTITION	-
建表和修改表时支持指定表级和列级comment	CREATE TABLE、ALTER TABLE	-
创建索引时支持指定索引级comment	CREATE INDEX	-

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
交换普通表和分区表分区的数据	ALTER TABLE PARTITION	ALTER TABLE EXCHANGE PARTITION的差异点： ● 对于自增列，MySQL执行ALTER TABLE EXCHANGE PARTITION后，自增列会被重置；GaussDB则不会被重置，自增列则按照旧的自增值递增。 ● MySQL表或分区使用tablespace时，则无法进行分区和普通表数据的交换；GaussDB表或分区使用不同的tablespace时，仍可进行分区和普通表数据的交换。 ● 对于列默认值，MySQL不会校验默认值，因此默认值不同时也可进行分区和普通表数据的交换；GaussDB会校验默认值，如果默认值不同，则无法进行分区和普通表数据的交换。 ● MySQL在分区表或普通表上进行DROP列操作后，表结构仍然一致，则可进行分区和普通表数据的交换；GaussDB需要保证普通表和分区表的被删除列严格对齐才能进行分区和普通表数据的交换。 ● MySQL和GaussDB的哈希算法不同，所以两者在相同的hash分区存储的数据可能不一致，导致最后交换的数据也可能不一致。 ● MySQL的分区表不支持外键，普通表包含外键或其他表引用普通表的外键，则无法进行分区和普通表数据的交换；GaussDB的分区表支持外键，在两个表的外键约束一致时，则可进行分区和普通表数据的交换，GaussDB的分区表不带外键，普通表有其他表引用，如果分区表和普通表表一致，则可进行分区和普通表数据的交换。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
支持删除表的主键外键约束	ALTER TABLE DROP [PRIMARY | FOREIGN]KEY	-
支持CREATE TABLE ... LIKE语法兼容	CREATE TABLE ... LIKE	• 在MySQL 8.0.16 之前的版本中，CHECK约束会被语法解析但功能会被忽略，表现为不复制CHECK约束，GaussDB支持复制CHECK约束。 • 对于主键约束名称，在建表时，MySQL所有主键约束名称固定为PRIMARY KEY，GaussDB不支持复制。 • 对于唯一键约束名称，在建表时，MySQL支持复制，GaussDB不支持复制。 • 对于CHECK约束名称，在建表时，MySQL 8.0.16 之前的版本无CHECK约束信息，GaussDB支持复制。 • 对于索引名称，在建表时，MySQL支持复制，GaussDB不支持复制。 • 在跨sql_mode模式建表时，MySQL受宽松模式和严格模式控制，GaussDB可能存在严格模式失效的情况。 例如：源表存在默认值“0000-00-00”，在“no_zero_date”严格模式下，GaussDB建表成功，且包含默认值“0000-00-00”，严格模式失效；而MySQL建表失败，受严格模式控制。 • 针对跨数据库创建表，MySQL支持，GaussDB不支持。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
支持更改表名兼容语法	ALTER TABLE tbl_name RENAME [TO | AS | =] new_tbl_name; RENAME {TABLE | TABLES} tbl_name TO new_tbl_name [, tbl_name2 TO new_tbl_name2, ...];	• GaussDB的alter rename语法仅支持修改表名称功能操作，不能耦合其它功能操作。 • GaussDB中，仅旧表名字段支持使用schema.table_name格式，且新表名与旧表名将属于同一个Schema。 • GaussDB不支持新旧表跨Schema重命名操作；但如有权限，则可在当前Schema下修改其它Schema下表名称。 • GaussDB的rename多组表的语法支持全为本地临时表的重命名，不支持本地临时表和非本地临时表组合的场景。 • GaussDB的RENAME TABLE校验顺序与MySQL不一致，导致报错信息不一致。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
支持增加子分区语法兼容	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})}	• 不支持如下语法添加多分区： ALTER TABLE table_name ADD PARTITION (partition_definition1, partition_definition1,…); • 仅支持原有添加多分区语法： ALTER TABLE table_name ADD PARTITION (partition_definition1), ADD PARTITION (partition_definition2[y1]), …;

2.7.2 DML

表 2-25 DML 语法兼容介绍

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
DELETE支持从多个表中删除数据	DELETE	-
DELETE支持ORDER BY和LIMIT	DELETE	-
DELETE支持从指定分区（或子分区）删除数据	DELETE	-

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
UPDATE支持从多个表中更新数据	UPDATE	-
UPDATE支持ORDER BY和LIMIT	UPDATE	-
SELECT INTO语法兼容	SELECT	● GaussDB可以使用SELECT INTO根据查询结果创建一个新表，MySQL不支持。 ● GaussDB的SELECT INTO语法不支持将多个查询进行集合运算后的结果作为查询结果。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

REPLACE INTO语法兼容	REPLACE	• 时间类型初始值的差异。例如:
		– MySQL不受严格模式和宽松模式的影响，可向表中插入时间0值，即：
		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)
		– GaussDB在宽松模式下才可以成功插入时间0值，即
		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);
		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)

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		在严格模式下，则报错date/time field value out of range: "0000-00-00 00:00:00"。
		• 位串类型初始值的差异。例如：
		– MySQLBIT类型的初始值为空串''，即：
		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)
		– GaussDB位串类型BIT的初始值为NULL，则报错。
		gaussdb=# CREATE TABLE test(f1 BIT(3) NOT NULL);
		CREATE TABLE
		gaussdb=# REPLACE INTO test VALUES(f1);
		ERROR: null value in column "f1" violates not-null constraint
		DETAIL: Failing row contains (null).

SELECT支持指定多分区查询	SELECT	-
UPDATE支持指定多分区更新	UPDATE	-

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异
LOAD DATA导入数据功能	LOAD DATA	• LOAD DATA语法执行结果与MySQL严格模式一致，宽松模式暂未适配。 • IGNORE与LOCAL参数功能仅为当导入数据与表中数据存在冲突时，忽略当前冲突行数据功能和当文件中字段数小于指定表中列数时自动为其余列填充默认值功能，其余功能暂未适配。 • 指定LOCAL关键字，且文件路径为相对路径时，文件从二进制目录下搜索；不指定LOCAL关键字，且文件路径为相对路径时，文件从数据目录下搜索。 • 语法中指定分隔符，转义字符，分行符等符号时，若指定为单引号，将导致词法解析错误。 • [(col_name_or_user_var [, col_name_or_user_var] ...)]指定列参数不支持重复指定列。 • [FIELDS TERMINATED BY 'string']指定换行符不能与[LINES TERMINATED BY 'string']分隔符相同。 • 执行LOAD DATA语法写入表中的数据若无法转换为表中数据类型格式时报错。 • LOAD DATA SET表达式中不支持指定列名计算。 • 若set表达式返回值类型与对应列类型之间不存在隐式转换函数则报错。 • LOAD DATA只能用于表，不能用于视图。 • Windows下的文件与Linux环境下文件默认换行符存在差异，LOAD DATA无法识别此场景会报错，建议用户导入时检查导入文件行尾换行符。

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

INSERT IGNORE兼容	INSERT IGNORE	• GaussDB会返回降级后的错误信息，MySQL则会将降级后的错误信息记录到错误堆栈中，然后调用show warnings;命令查看。
		• 时间类型的差异。例如：
		– GaussDB中date、datetime、timestamp默认零值。
		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)
		– MySQL中date、datetime、timestamp默认零值。
		mysql> CREATE TABLE test(f1 DATE NOT NULL, f2 DATETIME NOT NULL, f3 TIMESTAMP 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数据库功能概述	详细语法说明	GaussDB数据库实现差异

		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)
		• 由于GaussDB不支持MySQL的bit类型，因此忽略bit类型NOT NULL约束和插入的bit类型长度与定义不同的场景下不支持INSERT IGNORE错误降级。
		– GaussDB中bit类型
		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');
		ERROR: bit string length 4
		does not match type bit(10)
		CONTEXT: referenced column:
		f1
		– MySQL中bit类型
		mysql> CREATE TABLE test(f1
		BIT(10) NOT NULL);
		Query OK, 0 rows affected
		(0.00 sec)
		
		mysql> INSERT IGNORE INTO
		test VALUES(NULL);

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		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)
		• MySQL数据库时间类型指定精度时，插入时间零值会显示精度，GaussDB则不显示，例如：
		– GaussDB指定时间精度
		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)
		– MySQL指定时间精度
		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;
		+--------------

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		+-------------------------
		+-------------------------+
		| 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)
		• 由于MySQL数据库和GaussDB执行过程的差异，因此，产生的warnings条数可能不同，例如：
		– GaussDB产生的warnings条数
		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 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
		– MySQL产生的warnings条数
		mysql> CREATE TABLE test(f1 INT, f2 INT not null);
		Query OK, 0 rows affected (0.01 sec)

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		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
		• MySQL数据库和GaussDB INSERT IGNORE在触发器中的差异，例如：
		– GaussDB触发器中使用INSERT IGNORE
		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 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
		---- 	

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		0
		(1 rows)
		
		gaussdb=# SELECT * FROM test2;
		f1
		---- 	
		
		(1 rows)
		– MySQL触发器中使用INSERT IGNORE
		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 |
		+----+
		| 0 |
		+----+
		1 row in set (0.00 sec)
		
		mysql> SELECT * FROM test2;
		+------+
		| f1 |
		+------+
		| NULL |
		+------+
		1 row in set (0.00 sec)
		● GaussDB的bool、serial的实现机制与MySQL不同，因此其默认零值与MySQL不同，例如：

MySQL数据库功能概述	详细语法说明	GaussDB数据库实现差异

		– GaussDB的行为
		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)
		– MySQL的行为
		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)
		
		mysql> SELECT * FROM test;
		+----+----+
		| f1 | f2 |
		+----+----+
		| 1 | 0 |
		+----+----+
		1 row in set (0.00 sec)

2.7.3 DCL

表 2-26 DCL 语法兼容介绍

概述	详细语法说明	差异
支持SET用户自定义变量	SET	自定义变量长度的差异。例如： • MySQL自定义变量名长度没有约束。 • GaussDB自定义变量名长度不超过64字节，超过部分的变量名会截断并提示告警。
SET TRANSACTION语法兼容	SET TRANSACTION	MySQL可以设置当前会话（session）和全局（global）的事务隔离级别和读写模式，GaussDB设置当前会话需要设置参数b_format_behavior_compat_options包含set_session_tranasction，设置全局只对当前数据库生效。
SET NAMES指定COLLATE子句	SET [SESSION | LOCAL] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};	GaussDB暂不支持指定charset_name与数据库字符集不同。具体请参见《开发指南》中“SQL参考 > SQL语法 > S > SET ”章节。

2.8 驱动

2.8.1 JDBC

2.8.1.1 JDBC 接口参考

GaussDB与MySQL的JDBC接口定义一致，均遵循业界规范，本章节主要介绍GaussDB数据库的MySQL兼容性B模式与MySQL数据库JDBC接口的行为差异。

获取结果集中的数据

ResultSet对象提供了丰富的方法，以获取结果集中的数据。获取数据常用的方法如表2-27所示，其他方法请参考JDK官方文档。

表 2-27 ResultSet 对象的常用方法

方法	描述	差异
int getInt(int columnIndex)	按列标获取int型数据。	-
int getInt(String columnLabel)	按列名获取int型数据。	-
String getString(int columnIndex)	按列标获取String型数据。	字段类型为整型且带有ZEROFILL属性时，GaussDB按照ZEROFILL属性要求的宽度信息用0进行补位后输出结果，MySQL直接输出结果。
String getString(String columnLabel)	按列名获取String型数据。	字段类型为整型且带有ZEROFILL属性时，GaussDB按照ZEROFILL属性要求的宽度信息用0进行补位后输出结果，MySQL直接输出结果。
Date getDate(int columnIndex)	按列标获取Date型数据。	-
Date getDate(String columnLabel)	按列名获取Date型数据。	-

3 MySQL 兼容性 M-Compatibility 模式

3.1 数据类型

3.1.1 数值数据类型

除特别说明外，GaussDB数据库中的数据类型精度、标度、位数大小等默认不支持用浮点型数值定义，建议使用合法的整型数值定义。

表 3-1 整数类型

数据类型	与MySQL的差异
BOOL	输出格式：GaussDB中SELECT TRUE/FALSE输出结果为t/f，MySQL为1/0。 MySQL：BOOL/BOOLEAN类型实际映射为TINYINT类型。
BOOLEAN	
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	具体差异请参见表格下方的示例内容。
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]	具体差异请参见表格下方的示例内容。
INT[(M)] [UNSIGNED] [ZEROFILL]	具体差异请参见表格下方的示例内容。
INTEGER[(M)] [UNSIGNED] [ZEROFILL]	

数据类型	与MySQL的差异
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	

差异1：MySQL 5.7在形如'1.2.3.4'这类字符串中含有多个小数点的输入，在宽松模式（sql_mode未包含'strict_trans_tables'选项）下会错误的将第一个小数点的内容也插入进去。MySQL 8.0版本修复了该问题， GaussDB与MySQL 8.0保持一致。

```
-- GaussDB
m_db=# SET SQL_MODE='';
SET
m_db=# CREATE TABLE test_int(a int);
CREATE TABLE
m_db=# INSERT INTO test_int VALUES('1.2.3');
WARNING:  Data truncated for column
LINE 1: INSERT INTO test_int VALUES('1.2.3');
                                    ^
CONTEXT:  referenced column: a
INSERT 0 1
m_db=# SELECT * FROM test_int;
 a
---
 1
(1 row)

m_db=# DROP TABLE test_int;
DROP TABLE

-- MySQL 5.7
mysql> SET SQL_MODE='';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE test_int(a int);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO test_int VALUES('1.2.3');
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT * FROM test_int;
+------+
| a    |
+------+
|   12 |
+------+
1 row in set (0.00 sec)

mysql> DROP TABLE test_int;
Query OK, 0 rows affected (0.01 sec)
```

差异2：开启精度开关参数（m_format_behavior_compat_options包含'enable_precision_decimal'选项）时，在UNION的CREATE TABLE AS场景中，GaussDB对于直接输入的整型会取其最大长度（INT为11，BIGINT为20）来计算列的长度，MySQL会根据整型的实际长度来计算列的长度。

```
-- GaussDB
m_db=# set m_format_behavior_compat_options= 'enable_precision_decimal';
SET
m_db=# CREATE TABLE test_int AS SELECT 1234567 UNION ALL SELECT '456789';
INSERT 0 2
m_db=# DESC test_int;
  Field   |    Type     | Null | Key | Default | Extra
```



```
----------+-------------+------+-----+---------+-------
 ?column? | varchar(11) | YES  |     |         |
(1 row)

m_db=# DROP TABLE test_int;
DROP TABLE
m_db=# CREATE TABLE test_int AS SELECT 1234567890 UNION ALL SELECT '456789';
INSERT 0 2
m_db=# DESC test_int;
  Field   |    Type     | Null | Key | Default | Extra
----------+-------------+------+-----+---------+-------
 ?column? | varchar(20) | YES  |     |         |
(1 row)

m_db=# DROP TABLE test_int;
DROP TABLE

-- MySQL
mysql> CREATE TABLE test_int AS SELECT 1234567 UNION ALL SELECT '456789';
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DESC test_int;
+---------+------------+------+-----+---------+-------+
| Field   | Type       | Null | Key | Default | Extra |
+---------+------------+------+-----+---------+-------+
| 1234567 | varchar(7) | NO   |     |         |       |
+---------+------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE test_int;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE test_int AS SELECT 1234567890 UNION ALL SELECT '456789';
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DESC test_int;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| 1234567890 | varchar(10) | NO   |     |         |       |
+------------+-------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE test_int;
Query OK, 0 rows affected (0.00 sec)
```

表 3-2 任意精度类型

数据类型	与MySQL的差异
DECIMAL[(M[,D])] [ZEROFILL]	MySQL中该类型用一个9*9的数组存储数值，整数部分和小数部分分开存储，超过该长度时优先截断小数部分。GaussDB只会在整数位数超过81位时发生截断。
NUMERIC[(M[,D])] [ZEROFILL]	
DEC[(M[,D])] [ZEROFILL]	
FIXED[(M[,D])] [ZEROFILL]	

表 3-3 浮点类型

数据类型	与MySQL的差异
FLOAT[(M,D)] [ZEROFILL]	驱动中FLOAT类型和DOUBLE类型带精度标度场景，数据输入超范围不支持报错。
FLOAT(p) [ZEROFILL]	
DOUBLE[(M,D)] [ZEROFILL]	
DOUBLE PRECISION[(M,D)] [ZEROFILL]	
REAL[(M,D)] [ZEROFILL]	

3.1.2 日期与时间数据类型

表 3-4 日期与时间数据类型

数据类型	与MySQL的差异
DATE	无差异。
DATETIME[(fsp)]	具体差异请参见表格下方说明中的内容。
TIMESTAMP[(fsp)]	GaussDB支持timestamp数据类型，与MySQL相比规格上存在如下差异： ● MySQL 5.7中timestamp列默认有default value，为数据插入时的实时时间。GaussDB与MySQL 8.0一致，均不设置默认值，即插入null时，值为null。 ● 其余差异请参见表格下方说明中的内容。
TIME[(fsp)]	GaussDB支持time数据类型，与MySQL相比规格上存在如下差异： ● 当时间类型的时、分、秒、纳秒为0时，GaussDB和MySQL可能存在符号位不同的情况。 ● 其余差异请参见表格下方说明中的内容。
YEAR[(4)]	GaussDB支持year数据类型，与MySQL相比规格上存在如下差异： MySQL 5.7中year列默认显示为year(4)。GaussDB与MySQL 8.0一致，只显示year。

说明

- GaussDB不支持ODBC语法的字面量：
  { d 'str' }
  { t 'str' }
  { ts 'str' }
- GaussDB支持标准SQL字面量，且类型关键字后面可选择添加精度，MySQL不支持：
  DATE[(n)] 'str'
  TIME[(n)] 'str'
  TIMESTAMP[(n)] 'str'
- 当指定DATETIME、TIME、TIMESTAMP数据类型的精度超过其支持的最大精度时，GaussDB会将精度截断成支持的最大精度，MySQL则会报错。

3.1.3 字符串数据类型

表 3-5 字符串数据类型

数据类型	与MySQL的差异
CHAR(M)	具体差异请参见表格下方说明中的内容。
VARCHAR(M)	具体差异请参见表格下方说明中的内容。
TINYTEXT	具体差异请参见表格下方说明中的内容。
TEXT	具体差异请参见表格下方说明中的内容。
MEDIUMTEXT	具体差异请参见表格下方说明中的内容。
LONGTEXT	输入格式： • GaussDB最大支持1GB-512字节长度，MySQL最大支持4GB-1字节长度。 • 其余差异请参见表格下方说明中的内容。

说明

- 对于无法转义的二进制或十六进制字符串，MySQL会输出空字符串，GaussDB输出为十六进制结果。
- 对于TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT类型：
  - MySQL中不允许设置默认值，GaussDB中创建表列时语法上允许设置默认值。
  - 主键：MySQL中不支持主键，GaussDB支持。
  - 索引：MySQL中不支持除前缀索引外其他索引方法，GaussDB支持。
  - 外键：MySQL中不支持作为外键的参考列/被参考列，GaussDB支持。

示例：

```
-- GaussDB
m_db=# CREATE TABLE test_text(a text);
CREATE TABLE

m_db=# INSERT INTO test_text VALUES(0x1);
INSERT 0 1
```



```
m_db=# INSERT INTO test_text VALUES(0x111111);
INSERT 0 1

m_db=# INSERT INTO test_text VALUES(0x61);
INSERT 0 1

m_db=# SELECT * FROM test_text;
      a
--------------
 \x01
 \x11\x11\x11
 a
(3 rows)

m_db=# DROP TABLE test_text;
DROP TABLE

-- MySQL 5.7
mysql> CREATE TABLE test_text(a text);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO test_text VALUES(0x1);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO test_text VALUES(0x111111);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO test_text VALUES(0x61);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test_text;
+------+
| a    |
+------+
|     |
|   |
| a    |
+------+
3 rows in set (0.00 sec)

mysql> DROP TABLE test_text;
Query OK, 0 rows affected (0.01 sec)
```


3.1.4 二进制数据类型

表 3-6 二进制数据类型

数据类型	与MySQL的差异
BINARY[(M)]	● 输入格式： – 插入字符串长度小于目标长度时，GaussDB填充符是0x20，MySQL是0x00。 ● 字符集：默认字符集为数据库初始化字符集，MySQL默认类型字符集为BINARY字符集。 ● 输出格式： – JDBC协议输出时BINARY类型的末尾空格显示为空格，MySQL末尾空格显示为\x00。 – 宽松模式下， BINARY类型面对输入超过n的字节数的字符输入(例如中文字符), 会将超限的整个字符截断。MySQL中会将超限的整个字符的前几位满足n范围内的字节信息保留，但输出时字符信息显示乱码。 其余差异请参见表格下方说明中的内容。 **说明** GaussDB中，由于BINARY类型填充符与MySQL的差异，在操作符比较计算，字符串相关系统函数计算，索引匹配，数据导入导出等场景下与MySQL的表现会存在差异。具体差异场景请参见本章节示例。
VARBINARY(M)	字符集：默认字符集为数据库初始化字符集，MySQL默认类型字符集为BINARY字符集。 其余差异请参见表格下方说明中的内容。
TINYBLOB	具体差异请参见表格下方说明中的内容。
BLOB	具体差异请参见表格下方说明中的内容。
MEDIUMBLOB	具体差异请参见表格下方说明中的内容。
LONGBLOB	取值范围：只支持最大1GB-512字节长度，MySQL最大支持4GB-1字节长度。
BIT[(M)]	输出格式： ● GaussDB中会以二进制字符串形式输出，MySQL 5.7中根据ASCII码表转义，无法转义的输出无法显示的字符串。MySQL 8.0 中统一输出0x形式的十六进制结果。 ● 在MySQL 8.0以上版本，默认会开头补0，GaussDB不会补0。 其余差异请参见表格下方说明中的内容。

说明

- 对于无法转义的二进制或十六进制字符串，MySQL 5.7会输出无法显示的字符串， MySQL 8.0输出格式为0x形式的十六进制结果，GaussDB输出格式为多个\x的十六进制结果。
- 对于TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB类型：
  - MySQL中不允许设置默认值，GaussDB中创建表列时语法上允许设置默认值。
  - 主键：MySQL中不支持主键，GaussDB支持。
  - 索引：MySQL中不支持除前缀索引外其他索引方法，GaussDB支持。
  - 外键：MySQL中不支持作为外键的参考列/被参考列，GaussDB支持。

示例：

```
-- GaussDB
m_db=# CREATE TABLE test_blob(a blob);
CREATE TABLE

m_db=# INSERT INTO test_blob VALUES(0x1);
INSERT 0 1

m_db=# INSERT INTO test_blob VALUES(0x111111);
INSERT 0 1

m_db=# INSERT INTO test_blob VALUES(0x61);
INSERT 0 1

m_db=# SELECT * FROM test_blob;
     a
--------------
 \x01
 \x11\x11\x11
 a
(3 rows)

m_db=# DROP TABLE test_blob;
DROP TABLE

-- MySQL 5.7
mysql> CREATE TABLE test_blob(a blob);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO test_blob VALUES(0x1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO test_blob VALUES(0x111111);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO test_blob VALUES(0x61);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM test_blob;
+------+
| a    |
+------+
|      |
|  |
| a    |
+------+
3 rows in set (0.00 sec)

mysql> DROP TABLE test_blob;
Query OK, 0 rows affected (0.00 sec)

-- MySQL 8.0
mysql> CREATE TABLE test_blob(a blob);
Query OK, 0 rows affected (0.08 sec)
```



```
mysql> INSERT INTO test_blob VALUES(0x1);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO test_blob VALUES(0x111111);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO test_blob VALUES(0x61);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM test_blob;
+------------+
| a          |
+------------+
| 0x01       |
| 0x111111   |
| 0x61       |
+------------+
3 rows in set (0.00 sec)

mysql> DROP TABLE test_blob;
Query OK, 0 rows affected (0.04 sec)
```


3.1.5 JSON 类型

表 3-7 JSON 数据类型

数据类型	与MySQL的差异
JSON	GaussDB支持JSON数据类型与MySQL相比，规格存在如下差异： ● 取值范围： 在MySQL中，JSON数据类型的最大长度为4GB，但在GaussDB中，JSON数据类型的最大长度为1GB-512字节，对象的键值对个数最大值和数组的元素个数最大值也小于MySQL。 ● 字符序差异： 在MySQL中，使用collation函数单独查询JSON类型的列，返回的字符序是BINARY，但GaussDB中返回utf8mb4_bin。其他使用的场景都使用utf8mb4_bin，与MySQL相同。 ● 使用ORDER BY在非标量JSON类型行为上的差异： 在MySQL中，使用ORDER BY对非标量JSON类型排序的行为不做规定，会存在不支持对非标量JSON类型排序的告警。 在GaussDB中，支持对非标量的JSON进行排序，并按照一定的排序规则进行排序： – 首先比较JSON数据的类型，OPAQUE > TIMESTAMP = DATETIME > TIME > DATE > BOOL > 数组 > 对象 > 字符串类型 > DOUBLE = UINT = INT = DECIMAL > NULL，结果相等则比较内容。 – 标量之间比较值的大小，OPAQUE先进行类型之间的比较，TYPE_STRING > TYPE_VAR_STRING > TYPE_BLOB > TYPE_BIT > TYPE_VARCHAR > TYPE_YEAR，类型相同时依次比较每个字节的大小。 – 数组之间依次比较元素的大小，当其中一个数组的元素比较完成之后，则元素个数多的大。对象之间比较长度，长度相等则依次比较每个键值对，先比较键key，再比较值value。

说明

- 在GaussDB中，BLOB、TINYBLOB、MEDIUMBLOB、LONGBLOB、BINARY、VARBINARY、BIT以及YEAR类型转换为JSON类型，结果与MySQL不同。

  示例：

```
-- GaussDB
m_db=# CREATE TABLE test_blob (c1 BLOB, c2 TINYBLOB, c3 MEDIUMBLOB, c4 LONGBLOB, c5
BINARY(32), c6 VARBINARY(100), c7 BIT(64), c8 YEAR);
CREATE TABLE
m_db=# INSERT INTO test_blob VALUES('[1, "json"]', 'true', 'abc', '{"jsnid": 1, "tag": "ab"}', '[1,
"json"]', '{"jsnid": 1, "tag": "ab"}', '20', '2020');
INSERT 0 1
m_db=# SELECT CAST(c1 AS JSON), CAST(c2 AS JSON), CAST(c3 AS JSON), CAST(c4 AS JSON),
CAST(c5 AS JSON), CAST(c6 AS JSON), CAST(c7 AS JSON), CAST(c8 AS JSON) FROM test_blob;
      CAST       |  CAST  | CAST  |               CAST                |                CAST                |
CAST               | CAST |  CAST
-----------------+--------+-------+-----------------------------------+------------------------------------
+-----------------------------------+------+--------
 "[1, \"json\"]" | "true" | "abc" | "{\"jsnid\": 1, \"tag\": \"ab\"}" | "[1, \"json\"]                    " |
"{\"jsnid\": 1, \"tag\": \"ab\"}" | "20" | "2020"
(1 row)
m_db=# DROP TABLE test_blob;
DROP TABLE

-- MySQL
mysql> CREATE TABLE test_blob (c1 BLOB, c2 TINYBLOB, c3 MEDIUMBLOB, c4 LONGBLOB, c5
BINARY(32), c6 VARBINARY(100), c7 BIT(64), c8 YEAR);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO test_blob VALUES('[1, "json"]', 'true', 'abc', '{"jsnid": 1, "tag": "ab"}', '[1,
"json"]', '{"jsnid": 1, "tag": "ab"}', '20', '2020');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT CAST(c1 AS JSON), CAST(c2 AS JSON), CAST(c3 AS JSON), CAST(c4 AS JSON),
CAST(c5 AS JSON), CAST(c6 AS JSON), CAST(c7 AS JSON), CAST(c8 AS JSON) FROM test_blob;
+-----------------------------------+---------------------------+----------------------
+-------------------------------------------------------
+-------------------------------------------------------------
+-------------------------------------------------------+-----------------------------+--------------------------
+
| CAST(c1 AS JSON)                  | CAST(c2 AS JSON)          | CAST(c3 AS JSON)      | CAST(c4 AS
JSON)                              | CAST(c5 AS JSON)                                    | CAST(c6 AS
JSON)                             | CAST(c7 AS JSON)            | CAST(c8 AS JSON)        |
+-----------------------------------+---------------------------+----------------------
+-------------------------------------------------------
+-------------------------------------------------------------
+-------------------------------------------------------+-----------------------------+--------------------------
+
| "base64:type252:WzEsICJqc29uIl0=" | "base64:type249:dHJ1ZQ==" | "base64:type250:YWJj" |
"base64:type251:eyJqc25pZCI6IDEsICJ0YWciOiAiYWIifQ==" |
"base64:type254:WzEsICJqc29uIl0AAAAAAAAAAAAAAAAAAAAAAAAAAAAA=" |
"base64:type15:eyJqc25pZCI6IDEsICJ0YWciOiAiYWIifQ==" | "base64:type16:AAAAAAAAMjA=" |
"base64:type13:MjAyMA==" |
+-----------------------------------+---------------------------+----------------------
+-------------------------------------------------------
+-------------------------------------------------------------
+-------------------------------------------------------+-----------------------------+--------------------------
+
1 row in set (0.00 sec)
mysql> DROP TABLE test_blob;
Query OK, 0 rows affected (0.01 sec)
```

- 如果在当前版本内执行了::JSON强制类型转换，其实际转换的JSON类型与GUC参数m_format_behavior_compat_options兼容性选项是否设置了cast_as_new_json有关。

3.1.6 数据类型支持的属性

表 3-8 数据类型支持的属性

数据类型支持的属性
NULL
NOT NULL
DEFAULT
ON UPDATE
PRIMARY KEY
AUTO_INCREMENT
CHARACTER SET name
COLLATE name
ZEROFILL

差异点：

建表时对VARBINARY类型的字段设置默认值，在使用DESC等方式查询表结构时与MySQL存在差异，GaussDB显示为转换成十六进制后的值，而MySQL显示为原始值。

示例：

```
-- GaussDB
m_db=# CREATE TABLE test_varbinary(a varbinary(20) DEFAULT 'GaussDB');
CREATE TABLE

m_db=# DESC test_varbinary;
 Field |     Type      | Null | Key |      Default      | Extra
-------+---------------+------+-----+-------------------+-------
 a     | varbinary(20) | YES  |     | X'47617573734442' |
(1 row)

m_db=# DROP TABLE test_varbinary;
DROP TABLE

-- MySQL
mysql> CREATE TABLE test_varbinary(a varbinary(20) DEFAULT 'GaussDB');
Query OK, 0 rows affected (0.02 sec)

mysql> DESC test_varbinary;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | varbinary(20) | YES  |     | GaussDB |       |
+-------+---------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE test_varbinary;
Query OK, 0 rows affected (0.01 sec)
```

3.1.7 数据类型转换

不同的数据类型之间支持转换。有如下场景涉及到数据类型转换：

- 操作符（比较操作符、运算操作符等）的操作数的数据类型不一致。常见于查询条件或者关联条件中的比较运算。
- 函数调用时实参和形参的数据类型不一致。
- DML语句要更新（包括INSERT、UPDATE、MERGE、REPLACE等）的目标列，数据的类型和列的定义类型不一致。
- 集合运算（UNION以及EXCEPT）确定最终投影列的目标数据类型后，各个SELECT查询的投影列的类型和目标数据类型不一致。
- 其他表达式计算场景，根据不同表达式的数据类型， 来决定用于比较或者最终结果的目标数据类型。
- 普通的字符串类型当字符序为BINARY时，将转换成对应的二进制类型（TEXT转换成BLOB，VARCHAR转换成VARBINARY等）。

数据类型转换差异点主要分为：隐式转换，UNION/CASE、decimal类型。

隐式类型转换差异点

- GaussDB中统一平铺成小类型到小类型的转换规则，MySQL中使用小类型转大类型，大类型转小类型的转换规则。
- WHERE条件子句中只有字符串的场景，GaussDB中't'、'true'、'y'、'yes'、'on'、'1'以及其大写版本会被识别为TRUE，能得到查询结果，'f'、'false'、'n'、'no'、'off'、'0'以及其大写版本会被识别为FALSE，无法得到查询结果。除上述字符串外，其余字符串均报错。MySQL中字符串首位为非0的数字即可得到查询结果，其余场景均无法得到查询结果。

  示例：

```
-- GaussDB
m_db=# CREATE TABLE test_where(a int);
CREATE TABLE

m_db=# INSERT INTO test_where VALUES(1);
INSERT 0 1

m_db=# SELECT * FROM test_where WHERE 't';
 a
---
 1
(1 row)

m_db=# SELECT * FROM test_where WHERE '1';
 a
---
 1
(1 row)

m_db=# SELECT * FROM test_where WHERE '1a';
ERROR:  invalid input syntax for type boolean: "1a"
LINE 1: SELECT * FROM test_where WHERE '1a';
                                       ^
m_db=# SELECT * FROM test_where WHERE 'f';
 a
---
(0 rows)

m_db=# SELECT * FROM test_where WHERE '0';
 a
---
(0 rows)

m_db=# DROP TABLE test_where;
DROP TABLE
```



```
-- MySQL
mysql> CREATE TABLE test_where(a int);
Query OK, 0 rows affected (0.02 sec)

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

mysql> SELECT * FROM test_where WHERE 't';
Empty set, 1 warning (0.00 sec)

mysql> SELECT * FROM test_where WHERE '1';
+------+
| a    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM test_where WHERE '1a';
+------+
| a    |
+------+
|    1 |
+------+
1 row in set, 1 warning (0.00 sec)

mysql> SELECT * FROM test_where WHERE 'f';
Empty set, 1 warning (0.00 sec)

mysql> SELECT * FROM test_where WHERE '0';
Empty set (0.00 sec)

mysql> DROP TABLE test_where;
Query OK, 0 rows affected (0.01 sec)
```

- 对于YEAR类型的表，输入字符串时若遇到'e'或'E'，MySQL会以科学计数法来处理，GaussDB直接报错或截断。

  示例：

```
-- GaussDB
m_db=# SET SQL_MODE='';
SET

m_db=# CREATE TABLE test_year(a year);
CREATE TABLE

m_db=# INSERT INTO test_year VALUES('2E3');
WARNING:  Data truncated for column.
LINE 1: INSERT INTO test_year VALUES('2E3');
                                     ^
CONTEXT:  referenced column: a
INSERT 0 1
m_db=# SELECT * FROM test_year;
  a
------
 2002
(1 row)

m_db=# DROP TABLE test_year;
DROP TABLE

-- MySQL
mysql> CREATE TABLE test_year(a year);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO test_year VALUES('2E3');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test_year;
```



```
+------+
| a    |
+------+
| 2000 |
+------+
1 row in set (0.00 sec)

mysql> DROP TABLE test_year;
Query OK, 0 rows affected (0.01 sec)
```

- GaussDB中函数嵌套场景下，涉及到聚合函数（如max、min、sum和avg）中存在字符串类型包含非数值字符，隐式转换到数值类型发生截断或置零，且包含操作符比较、having比较的场景时，GaussDB统一进行类型转换并产生告警，MySQL在相同场景下不会全部产生告警。

  示例：

```
-- GaussDB
m_db=# SET m_format_behavior_compat_options= 'enable_precision_decimal';
SET

m_db=# SELECT max(c4) <> 0 FROM ((SELECT 2.22 id, '2006-04-27 20:19:02.132' c4)) tb_1;
 ?column?
----------
 t
(1 row)

m_db=# SELECT sum(c4) <> 0 FROM ((SELECT 2.22 id, '2006-04-27 20:19:02.132' c4)) tb_1;
WARNING:  The double value '2006-04-27 20:19:02.132' is incorrect.
 ?column?
----------
 t
(1 row)

m_db=# SELECT (SELECT max(c4) f5 FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_1 WHERE EXISTS (SELECT max(c4) FROM
((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION ALL (SELECT 2.22 id, '1985-09-01 07:59:59'
c4)) tb_2) GROUP BY id WITH rollup HAVING f5<>0 LIMIT 0,1) + INTERVAL '33.22'
SECOND_MICROSECOND col5;
            col5
----------------------------
 2006-04-27 20:19:41.352000
(1 row)

m_db=# SELECT (SELECT sum(c4) f5 FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_1 WHERE EXISTS (SELECT sum(c4) FROM ((SELECT
2.22 id, '2006-04-27 20:19:08.132' c4) UNION ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_2)
GROUP BY id WITH rollup HAVING f5<>0 LIMIT 0,1) + INTERVAL '33.22' SECOND_MICROSECOND
col5;
WARNING:  The double value '2006-04-27 20:19:08.132' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  The double value '1985-09-01 07:59:59' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  The double value '2006-04-27 20:19:08.132' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  The double value '2006-04-27 20:19:08.132' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  The double value '1985-09-01 07:59:59' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  The double value '1985-09-01 07:59:59' is incorrect.
CONTEXT:  referenced column: col5
WARNING:  Incorrect datetime value: '3991'
CONTEXT:  referenced column: col5
 col5
------

(1 row)

-- MySQL
mysql> SELECT max(c4) <> 0 FROM ((SELECT 2.22 id, '2006-04-27 20:19:02.132' c4)) tb_1;
```



```
+--------------+
| max(c4) <> 0 |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT sum(c4) <> 0 FROM ((SELECT 2.22 id, '2006-04-27 20:19:02.132' c4)) tb_1;
+--------------+
| sum(c4) <> 0 |
+--------------+
|            1 |
+--------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW warnings;
+---------+------+-----------------------------------------------------------+
| Level   | Code | Message                                                   |
+---------+------+-----------------------------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: '2006-04-27 20:19:02.132' |
+---------+------+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT (SELECT max(c4) f5 FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_1
    -> WHERE EXISTS (SELECT max(c4) FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_2)
    -> GROUP BY id WITH rollup HAVING f5<>0 limit 0,1) + INTERVAL '33.22'
SECOND_MICROSECOND col5;
+----------------------------+
| col5                       |
+----------------------------+
| 2006-04-27 20:19:41.352000 |
+----------------------------+
1 row in set (0.00 sec)

mysql> SELECT (SELECT sum(c4) f5 FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_1
    -> WHERE EXISTS (SELECT sum(c4) FROM ((SELECT 2.22 id, '2006-04-27 20:19:08.132' c4) UNION
ALL (SELECT 2.22 id, '1985-09-01 07:59:59' c4)) tb_2)
    -> GROUP BY id WITH rollup HAVING f5<>0 LIMIT 0,1) + INTERVAL '33.22'
SECOND_MICROSECOND col5;
+------+
| col5 |
+------+
| NULL |
+------+
1 row in set, 7 warnings (0.01 sec)

mysql> SHOW warnings;
+---------+------+-----------------------------------------------------------+
| Level   | Code | Message                                                   |
+---------+------+-----------------------------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: '2006-04-27 20:19:08.132' |
| Warning | 1292 | Truncated incorrect DOUBLE value: '1985-09-01 07:59:59'     |
| Warning | 1292 | Truncated incorrect DOUBLE value: '2006-04-27 20:19:08.132' |
| Warning | 1292 | Truncated incorrect DOUBLE value: '2006-04-27 20:19:08.132' |
| Warning | 1292 | Truncated incorrect DOUBLE value: '1985-09-01 07:59:59'     |
| Warning | 1292 | Truncated incorrect DOUBLE value: '1985-09-01 07:59:59'     |
| Warning | 1292 | Incorrect datetime value: '3991'                            |
+---------+------+-----------------------------------------------------------+
7 rows in set (0.00 sec)
```

UNION，CASE 和相关构造差异点

- POLYGON + NULL、POINT + NULL、POLYGON + POINT组合在MySQL中均返回GEOMETRY类型，GaussDB中未涉及，暂时当做报错处理。

- SET和ENUM两种类型暂未支持，暂时当做报错处理。
- JSON和二进制类型（BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB）、BIT或者YEAR类型的UNION和UNION ALL组合，MySQL中返回LONGBLOB或者LONGTEXT类型，GaussDB中返回JSON类型，同时支持二进制类型（BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB）、BIT或者YEAR类型到JSON的隐式类型转换。
- 未设置m_format_behavior_compat_options为enable_precision_decimal时，常量类型和其他类型做类型聚合的时候，输出类型的精度为其他类型的精度。如“SELECT "helloworld" UNION SELECT p FROM t;”的结果的精度为属性p的精度。
- 未设置m_format_behavior_compat_options为enable_precision_decimal时，定点常量和不带精度约束的类型（非字符串类型如int、bool、year等，聚合结果类型为定点类型）聚合时，精度约束会按照定点数默认精度31输出。
- merge rule差异：

  MySQL 5.7中存在部分不合理的类型推导，如BIT类型和整型/YEAR类型推导会得出VARBINARY类型，UNSIGNED类型和非UNSIGNED类型推导会得到带UNSIGNED的类型等，同时CASE WHEN和UNION的聚合结果也存在差异，类型推导结果太小时存在数据溢出风险。在MySQL 8.0版本修复了上述相关的问题，因此merge rule聚合规则以8.0为准。
- MySQL中BINARY和CHAR填充字符不相同，BINARY填充'\0'，CHAR填充空格，GaussDB中BINARY和CHAR都是填充空格。
- 在精度传递场景下，使用CASE WHEN语句时，会进行类型转换和精度重新计算，导致最终的输出结果与CASE子句对比会出现末尾多零场景或末尾少零场景：
  - 末尾多零场景：CASE节点会根据CASE子句的精度计算CASE节点精度，当THEN子句的精度比CASE节点的精度小时，会在CASE节点末尾补零。
  - 末尾少零场景：多层CASE WHEN嵌套时，内层CASE执行类型转换之后，只保留内层CASE的精度，外层CASE无法得到THEN子句的精度信息，因此外层CASE会根据内层CASE的精度计算的精度进行类型转换。当外层CASE类型转换时内层CASE精度比THEN子句少，会出现末尾少零场景。

  示例：

  - 末尾多零少零场景。

```
-- 末尾多零场景。
m_db=# SELECT 15.6 AS result;
 result
--------
   15.6
(1 row)

m_db=# SELECT CASE WHEN 1 < 2 THEN 15.6 ELSE  23.578 END AS result;
 result
--------
 15.600
(1 row)

m_db=# SELECT greatest(12, 3.4, 15.6) AS result;
 result
--------
   15.6
(1 row)

m_db=# SELECT CASE WHEN 1 < 2 THEN greatest(12, 3.4, 15.6) ELSE greatest(123.4, 23.578,
36) END AS result;
 result
--------
 15.600
```



```
(1 row)

-- 末尾少零场景。
m_db=# CREATE TABLE t1 AS SELECT (false/-timestamp '2008-12-31 23:59:59.678') AS result;
INSERT 0  1
m_db=# DESC t1;
 Field  |    Type     | Null | Key | Default | Extra
--------+-------------+------+-----+---------+-------
 result | double(8,7) | YES  |     |         |
(1 row)

m_db=# SELECT (false/-timestamp '2008-12-31 23:59:59.678') AS result;
   result
------------
 -0.0000000
(1 row)

m_db=# CREATE TABLE t1 AS SELECT (CASE WHEN 1<2 THEN false/-timestamp '2008-12-31
23:59:59.678' ELSE 0016.11e3/'22.2' END) AS result;
INSERT 0 1
m_db=# DESC t1;
 Field  |  Type  | Null | Key | Default | Extra
--------+--------+------+-----+---------+-------
 result | double | YES  |     |         |
(1 row)

m_db=# SELECT (CASE WHEN 1<2 THEN false/-timestamp '2008-12-31 23:59:59.678' ELSE
0016.11e3/'22.2' END) AS result;
 result
--------
     -0
(1 row)

m_db=# DROP TABLE t1;
DROP TABLE
m_db=# CREATE TABLE t1 AS SELECT (CASE WHEN 1+1=2 THEN CASE WHEN 1<2 THEN false/-
timestamp '2008-12-31 23:59:59.678' ELSE 0016.11e3/'22.2' END ELSE 'test' END) AS result;
INSERT 0 1
m_db=# DESC t1;
 Field  |    Type     | Null | Key | Default | Extra
--------+-------------+------+-----+---------+-------
 result | varchar(23) | YES  |     |         |
(1 row)

m_db=# SELECT (CASE WHEN 1+1=2 THEN CASE WHEN 1<2 THEN false/-timestamp
'2008-12-31 23:59:59.678' ELSE 0016.11e3/'22.2' END ELSE 'test' END) AS result;
 result
--------
 -0
(1 row)
```

- 在开启精度传递的场景下，使用集合运算（UNION以及EXCEPT），如果参与集合运算的查询语句，其查询的字段为函数、表达式而不是直接使用表中的字段，且查询的结果数据类型为INT/INT UNSIGNED，则最后返回的数据类型存在差异。在MySQL中，返回的数据类型为BIGINT/BIGINT UNSIGNED；在GaussDB中，返回的数据类型为INT/INT UNSIGNED。

```
-- GaussDB执行结果。
m_db=# SET
m_format_behavior_compat_options='select_column_name,enable_precision_decimal';
SET
m_db=# DROP TABLE IF EXISTS t1,t2,ctas1,ctas2;
DROP TABLE
m_db=# CREATE TABLE t1(a INT, b INT);
CREATE TABLE
m_db=# CREATE TABLE t2(c INT UNSIGNED, d INT UNSIGNED);
CREATE TABLE
m_db=# CREATE TABLE ctas1 AS (SELECT a, ABS(a) FROM t1) UNION (SELECT b, ABS(b) FROM
t1);
INSERT 0 0
```



```
m_db=# DESC ctas1;
 Field  |    Type     | Null | Key | Default | Extra
--------+-------------+------+-----+---------+-------
 a      | integer(11) | YES  |     |         |
 ABS(a) | integer(11) | YES  |     |         |
(2 rows)

m_db=# CREATE TABLE ctas2 AS (SELECT c, ABS(c) FROM t2) UNION (SELECT d, ABS(d) FROM
t2);
INSERT 0 0
m_db=# DESC ctas2;
 Field  |         Type         | Null | Key | Default | Extra
--------+----------------------+------+-----+---------+-------
 c      | integer(11) unsigned | YES  |     |         |
 ABS(c) | integer(11) unsigned | YES  |     |         |
(2 rows)

m_db=# DROP TABLE IF EXISTS t1,t2,ctas1,ctas2;
DROP TABLE

-- MySQL执行结果。
mysql> DROP TABLE IF EXISTS t1,t2,ctas1,ctas2;
Query OK, 0 rows affected, 4 warnings (0.00 sec)

mysql> CREATE TABLE t1(a INT, b INT);
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE t2(c INT UNSIGNED, d INT UNSIGNED);
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE ctas1 AS (SELECT a, ABS(a) FROM t1) UNION (SELECT b, ABS(b) FROM
t1);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC ctas1;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| a      | int(11)    | YES  |     | NULL    |       |
| ABS(a) | bigint(20) | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> CREATE TABLE ctas2 AS (SELECT c, ABS(c) FROM t2) UNION (SELECT d, ABS(d) FROM
t2);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC ctas2;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| c      | int(11) unsigned    | YES  |     | NULL    |       |
| ABS(c) | bigint(20) unsigned | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> DROP TABLE IF EXISTS t1,t2,ctas1,ctas2;
Query OK, 0 rows affected (0.07 sec)
```

- 在开启精度传递的场景下，CASE WHEN被嵌套场景的结果与MySQL保持差异。MySQL的类型可以透过多层直接转换，而GaussDB结果精度是逐层确定并且逐层转换的，因此可能导致结果小数位或进位和MySQL不一致。

```
-- GaussDB:
m_db=# SET m_format_behavior_compat_options='enable_precision_decimal';
SET
m_db=# SELECT (CASE WHEN 1+1=3 THEN 'test' ELSE CASE WHEN 1>2 THEN '-1.5'%06.6600e1
ELSE -TIME '10:10:10.456'%2.2 END END) RES;
```



```
        res
--------------------
 -1.8559999999974321
(1 row)

-- MySQL:
mysql> SELECT (CASE WHEN 1+1=3 THEN 'test' ELSE CASE WHEN 1>2 THEN '-1.5'%06.6600e1
ELSE -TIME '10:10:10.456'%2.2 END END) RES;
+--------+
| res    |
+--------+
| -1.856 |
+--------+
1 row in set (0.00 sec)
```

- 对于需要int类型的运算符（如 ~，&，|，<<，>>）嵌套CASE WHEN语句，若CASE WHEN语句返回的是varchar类型，则实际情况可以会发生截断（根据原表数据分析是否会发生截断），GaussDB会报出相应错误（SELECT查询warning告警，CREATE建表error报错），MySQL不会报错。若GaussDB想要完成CREATE TABLE建表操作，可以通过设置sql_mode关闭严格模式。

```
-- GaussDB:
m_db=# CREATE TABLE t_base (num_var numeric(20, 10), time_var time(6));
CREATE TABLE
m_db=# INSERT INTO t_base VALUES ('-2514.1441000000','12:10:10.125000'),
('-417.2147000000',' 11:30:25.258000');
INSERT 0 2
m_db=# SELECT (~(CASE WHEN false THEN time_var ELSE num_var END)) AS res2 FROM
t_base;
WARNING:  Truncated incorrect INTEGER value: '-2514.1441000000'
CONTEXT:  referenced column: res2
WARNING:  Truncated incorrect INTEGER value: '-417.2147000000'
CONTEXT:  referenced column: res2
 res2
------
 2513
 416
(2 rows)
m_db=# CREATE TABLE t1 AS SELECT (~(CASE WHEN false THEN time_var ELSE num_var END))
AS res2 FROM t_base;
ERROR:  Truncated incorrect INTEGER value: '-2514.1441000000'
CONTEXT:  referenced column: res2
m_db=# SET sql_mode="";
SET
m_db=# CREATE TABLE t1 AS SELECT (~(CASE WHEN false THEN time_var ELSE num_var END))
AS res2 FROM t_base;
WARNING:  Truncated incorrect INTEGER value: '-2514.1441000000'
CONTEXT:  referenced column: res2
WARNING:  Truncated incorrect INTEGER value: '-417.2147000000'
CONTEXT:  referenced column: res2
INSERT 0 2
m_db=# DESC t1;
 Field |        Type         | Null | Key | Default | Extra
-------+---------------------+------+-----+---------+-------
 res2  | bigint(21) unsigned | YES  |     |         |
(1 row)

-- MySQL:
mysql> CREATE TABLE t_base (num_var numeric(20, 10), time_var time(6));
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO t_base VALUES ('-2514.1441000000','12:10:10.125000'),
('-417.2147000000',' 11:30:25.258000');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> SELECT (~(CASE WHEN false THEN time_var ELSE num_var END)) AS res2 FROM t_base;
+------+
| res2 |
+------+
| 2513 |
|  416 |
```



```
+------+
2 rows in set (0.00 sec)
mysql> CREATE TABLE t1 AS SELECT (~(CASE WHEN false THEN time_var ELSE num_var END))
AS res2 FROM t_base;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DESC t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| res2  | bigint(21) unsigned | YES  |     | NULL    |       |
+-------+---------------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

- 在开启精度传递的场景下，对于CREATE VIEW AS SELECT CASE WHEN语句和SELECT CASE WHEN语句嵌套常量（包括常量计算、函数嵌套常量等）的情况，GaussDB在该情况下值保持一致，MySQL在SELECT CASE WHEN语句中可能会丢失部分精度。

```
-- GaussDB：
m_db=# CREATE OR REPLACE VIEW test_view AS
m_db-# SELECT (CASE WHEN 1<2 THEN 3.33/4.46 ELSE 003.3630/002.2600 END) c1,(CASE
WHEN 1>2 THEN IFNULL(null,3.363/2.2) ELSE NULLIF(3.33/4.46,3.363/2.2) END) c2;
CREATE VIEW
m_db=# SELECT * FROM test_view;
    c1     |    c2
-----------+-----------
 0.74663677 | 0.7466368
(1 row)
m_db=# SELECT (CASE WHEN 1<2 THEN 3.33/4.46 ELSE 003.3630/002.2600 END) c1,(CASE
WHEN 1>2 THEN IFNULL(null,3.363/2.2) ELSE NULLIF(3.33/4.46,3.363/2.2) END) c2;
    c1     |    c2
-----------+-----------
 0.74663677 | 0.7466368
(1 row)

-- MySQL：
mysql> CREATE OR REPLACE VIEW test_view AS
    -> SELECT (CASE WHEN 1<2 THEN 3.33/4.46 ELSE 003.3630/002.2600 END) c1,(CASE WHEN
1>2 THEN IFNULL(null,3.363/2.2) ELSE NULLIF(3.33/4.46,3.363/2.2) END) c2;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM test_view;
+------------+-----------+
| c1         | c2        |
+------------+-----------+
| 0.74663677 | 0.7466368 |
+------------+-----------+
1 row in set (0.00 sec)
mysql> SELECT (CASE WHEN 1<2 THEN 3.33/4.46 ELSE 003.3630/002.2600 END) c1,(CASE
WHEN 1>2 THEN IFNULL(null,3.363/2.2) ELSE NULLIF(3.33/4.46,3.363/2.2) END) c2;
+----------+----------+
| c1       | c2       |
+----------+----------+
| 0.746637 | 0.746637 |
+----------+----------+
1 row in set (0.00 sec)
```

- 在开启精度传递的场景下，GaussDB数据库支持UNION/CASE WHEN语句建表，但是由于架构不同，GaussDB数据库无法保证创建的表的所有类型与MySQL 8.0完全相同。MySQL返回字符串、二进制相关类型的场景，以及部分函数嵌套场景，与GaussDB存在不一致。

```
-- GaussDB:
m_db=# CREATE TABLE IF NOT EXISTS testcase (id int, col_text1 tinytext, col_text2 text,
col_blob1 tinyblob, col_blob2 blob, col_blob3 mediumblob, col_blob4 longblob);
CREATE TABLE
m_db=# CREATE TABLE t1 AS SELECT id,(CASE WHEN id=2 THEN col_text1 ELSE 'test' END)
f35,  (CASE WHEN id=2 THEN col_text2 ELSE 'test' END) f36,(CASE WHEN id=2 THEN col_blob1
ELSE 'test' END) f41,  (CASE WHEN id=2 THEN col_blob2 ELSE 'test' END) f42,  (CASE WHEN
```



```
id=2 THEN col_blob3 ELSE 'test' END) f43,  (CASE WHEN id=2 THEN col_blob4 ELSE 'test' END)
f44 FROM testcase;
INSERT 0 0
m_db=# DESC t1;
Field |       Type       | Null | Key | Default | Extra
-------+------------------+------+-----+---------+-------
id    | integer(11)     | YES |     |         |
f35   | varchar(255)    | YES |     |         |
f36   | mediumtext      | YES |     |         |
f41   | varbinary(255) | YES |     |         |
f42   | blob            | YES |     |         |
f43   | mediumblob      | YES |     |         |
f44   | longblob        | YES |     |         |
(7 rows)

m_db=# CREATE TABLE IF NOT EXISTS testtext1 (col10 text);
CREATE TABLE
m_db=# CREATE TABLE IF NOT EXISTS testtext2 (col10 text);
CREATE TABLE
m_db=# CREATE TABLE testtext AS (SELECT * FROM testtext1) UNION (SELECT * FROM
testtext2);
CREATE TABLE
m_db=# DESC testtext;
m_db=#
 Field | Type | Null | Key | Default | Extra
-------+------+------+-----+---------+-------
 col10 | text | YES |     |         |
(1 row)

m_db=# CREATE TABLE testchar AS SELECT (SELECT lcase(-6873.4354)) a, (SELECT
sec_to_time(-485769.567)) b UNION ALL SELECT (SELECT bin(-58768923.21321)), (SELECT
asin(-0.7237465));
INSERT 0 2
m_db=# DESC testchar;
 Field |     Type      | Null | Key | Default | Extra
-------+---------------+------+-----+---------+-------
 a     | text          | YES |     |         |
 b     | varchar(23) | YES |     |         |
(2 rows)

m_db=# CREATE TABLE test_func (col_text char(29));
CREATE TABLE
m_db=# CREATE TABLE test1 AS SELECT * FROM ( SELECT
        GREATEST(2.22, col_text) f1, LEAST(2.22, col_text) f2,
        ADDDATE(col_text, INTERVAL '1.28.16.31' HOUR_MICROSECOND) f3,
        SUBDATE(col_text, INTERVAL '39.49.15' MINUTE_MICROSECOND) f4,
        DATE_SUB(col_text, INTERVAL '45' MICROSECOND) f5,
        DATE_ADD(col_text, INTERVAL '12.00.00.00.001' DAY_MICROSECOND) f6,
        ADDTIME(col_text, '8:20:20.3554') f7,
        SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t1
UNION ALL
        SELECT * FROM ( SELECT
        GREATEST(2.22, col_text) f1, LEAST(2.22, col_text) f2,
        ADDDATE(col_text, INTERVAL '1.28.16.31' HOUR_MICROSECOND) f3,
        SUBDATE(col_text, INTERVAL '39.49.15' MINUTE_MICROSECOND) f4,
        DATE_SUB(col_text, INTERVAL '45' MICROSECOND) f5,
        DATE_ADD(col_text, INTERVAL '12.00.00.00.001' DAY_MICROSECOND) f6,
        ADDTIME(col_text, '8:20:20.3554') f7,
        SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t2;
INSERT 0 0
m_db=# DESC test1;
 Field |     Type      | Null | Key | Default | Extra
-------+---------------+------+-----+---------+-------
 f1    | double        | YES |     |         |
 f2    | double        | YES |     |         |
 f3    | varchar(29) | YES |     |         |
 f4    | varchar(29) | YES |     |         |
 f5    | varchar(29) | YES |     |         |
 f6    | varchar(29) | YES |     |         |
```



```
 f7    | varchar(29) | YES |     |         |
 f8    | varchar(29) | YES |     |         |
(8 rows)

-- MySQL:
mysql> CREATE TABLE IF NOT EXISTS testcase (id int, col_text1 tinytext, col_text2 text,
col_blob1 tinyblob, col_blob2 blob, col_blob3 mediumblob, col_blob4 longblob);
Query OK, 0 rows affected (0.01 sec)
mysql> CREATE TABLE t1 AS SELECT id,(CASE WHEN id=2 THEN col_text1 ELSE 'test' END) f35,
(CASE WHEN id=2 THEN col_text2 ELSE 'test' END) f36,(CASE WHEN id=2 THEN col_blob1 ELSE
'test' END) f41,  (CASE WHEN id=2 THEN col_blob2 ELSE 'test' END) f42,  (CASE WHEN id=2
THEN col_blob3 else 'test' END) f43,  (CASE WHEN id=2 THEN col_blob4 ELSE 'test' END) f44
FROM testcase;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC t1;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| id    | int      | YES |     | NULL    |       |
| f35   | longtext | YES |     | NULL    |       |
| f36   | longtext | YES |     | NULL    |       |
| f41   | longblob | YES |     | NULL    |       |
| f42   | longblob | YES |     | NULL    |       |
| f43   | longblob | YES |     | NULL    |       |
| f44   | longblob | YES |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> CREATE TABLE IF NOT EXISTS testtext1 (col10 text);
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE IF NOT EXISTS testtext2 (col10 text);
Query OK, 0 rows affected (0.02 sec)

mysql>  CREATE TABLE testtext AS (SELECT * FROM testtext1) UNION (SELECT * FROM
testtext2);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC testtext;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| col10 | mediumtext | YES |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> SET sql_mode='';
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> CREATE TABLE testchar AS SELECT (SELECT lcase(-6873.4354)) a, (SELECT
sec_to_time(-485769.567)) b UNION ALL SELECT (SELECT bin(-58768923.21321)), (SELECT
asin(-0.7237465));
Query OK, 2 rows affected, 1 warning (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 1

mysql> DESC testchar;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | varchar(21) | YES |     | NULL    |       |
| b     | varchar(53) | YES |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE test_func (col_text char(29));
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> CREATE TABLE test1 AS SELECT * FROM ( SELECT
    -> GREATEST(2.22, col_text) f1, LEAST(2.22, col_text) f2,
    -> ADDDATE(col_text, INTERVAL '1.28.16.31' HOUR_MICROSECOND) f3,
    -> SUBDATE(col_text, INTERVAL '39.49.15' MINUTE_MICROSECOND) f4,
    -> DATE_SUB(col_text, INTERVAL '45' MICROSECOND) f5,
    -> DATE_ADD(col_text, INTERVAL '12.00.00.00.001' DAY_MICROSECOND) f6,
    -> ADDTIME(col_text, '8:20:20.3554') f7,
    -> SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t1
    -> UNION ALL
    -> SELECT * FROM ( SELECT
    -> GREATEST(2.22, col_text) f1, LEAST(2.22, col_text) f2,
    -> ADDDATE(col_text, INTERVAL '1.28.16.31' HOUR_MICROSECOND) f3,
    -> SUBDATE(col_text, INTERVAL '39.49.15' MINUTE_MICROSECOND) f4,
    -> DATE_SUB(col_text, INTERVAL '45' MICROSECOND) f5,
    -> DATE_ADD(col_text, INTERVAL '12.00.00.00.001' DAY_MICROSECOND) f6,
    -> ADDTIME(col_text, '8:20:20.3554') f7,
    -> SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t2;
    -> SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t1
    -> UNION ALL
    -> SELECT * FROM ( SELECT
    -> GREATEST(2.22, col_text) f1, LEAST(2.22, col_text) f2,
    -> ADDDATE(col_text, INTERVAL '1.28.16.31' HOUR_MICROSECOND) f3,
    -> SUBDATE(col_text, INTERVAL '39.49.15' MINUTE_MICROSECOND) f4,
    -> DATE_SUB(col_text, INTERVAL '45' MICROSECOND) f5,
    -> DATE_ADD(col_text, INTERVAL '12.00.00.00.001' DAY_MICROSECOND) f6,
    -> ADDTIME(col_text, '8:20:20.3554') f7,
    -> SUBTIME(col_text, '8:20:20.3554') f8 FROM test_func) t2;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC test1;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| f1    | binary(23) | YES  |     | NULL    |       |
| f2    | binary(23) | YES  |     | NULL    |       |
| f3    | char(29)   | YES  |     | NULL    |       |
| f4    | char(29)   | YES  |     | NULL    |       |
| f5    | char(29)   | YES  |     | NULL    |       |
| f6    | char(29)   | YES  |     | NULL    |       |
| f7    | char(29)   | YES  |     | NULL    |       |
| f8    | char(29)   | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

- 在开启精度传递的场景下，对于CREATE TABLE AS SELECT A % (CASE WHEN)语句，如果A是DECIMAL类型，CASE WHEN结果为日期类型（DATE、TIME、DATETIME），两者进行取模运算（%）得到的精度保持差异。GaussDB得到的精度跟decimal类型与日期类型直接取模运算得到的精度保持一致。

```
-- GaussDB：(decimal % date类型case)与(numeric%date)， 精度一致，都是decimal(24,10)。
m_db=# SET m_format_behavior_compat_options = 'enable_precision_decimal';
SET
m_db=# DROP TABLE IF EXISTS t1, t2;
DROP TABLE
m_db=# CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));
CREATE TABLE
m_db=# CREATE TABLE t2 AS SELECT num_var % (CASE WHEN true THEN dt_var ELSE dt_var
END) AS res1 FROM t1;
INSERT 0 0
m_db=# DESC t2;
 Field |      Type      | Null | Key | Default | Extra
-------+----------------+------+-----+---------+-------
 res1  | decimal(24,10) | YES  |     |         |
(1 row)

m_db=# DROP TABLE IF EXISTS t1, t2;
```



```
DROP TABLE
m_db=# CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));
CREATE TABLE
m_db=# CREATE TABLE t2 AS SELECT num_var % dt_var AS res1 FROM t1;
INSERT 0 0
m_db=# DESC t2;
 Field |      Type      | Null | Key | Default | Extra
-------+----------------+------+-----+---------+-------
 res1  | decimal(24,10) | YES  |     |         |
(1 row)

-- MySQL 5.7，精度存在差异。(decimal % date类型case)精度为decimal(65,10)，(numeric%date)
精度为decimal(24,10)。
mysql> DROP TABLE IF EXISTS t1, t2;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 AS SELECT num_var % (CASE WHEN true THEN dt_var ELSE dt_var
END) AS res1 FROM t1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC t2;
+-------+----------------+------+-----+---------+-------+
| Field | Type           | Null | Key | Default | Extra |
+-------+----------------+------+-----+---------+-------+
| res1  | decimal(65,10) | YES  |     | NULL    |       |
+-------+----------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE IF EXISTS t1, t2;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 AS SELECT num_var % dt_var AS res1 FROM t1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC t2;
+-------+----------------+------+-----+---------+-------+
| Field | Type           | Null | Key | Default | Extra |
+-------+----------------+------+-----+---------+-------+
| res1  | decimal(24,10) | YES  |     | NULL    |       |
+-------+----------------+------+-----+---------+-------+
1 row in set (0.00 sec)

-- MySQL 8.0，(decimal % date类型case)和(numeric%date)精度都为decimal(20,10)。
mysql> DROP TABLE IF EXISTS t1, t2;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 AS SELECT num_var % (CASE WHEN true THEN dt_var ELSE dt_var
END) AS res1 FROM t1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC t2;
+-------+----------------+------+-----+---------+-------+
| Field | Type           | Null | Key | Default | Extra |
```



```
+-------+----------------+------+-----+---------+-------+
| res1  | decimal(20,10) | YES  |     | NULL    |       |
+-------+----------------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> DROP TABLE IF EXISTS t1, t2;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t1 (num_var numeric(20, 10), date_var date, time_var time(6), dt_var
datetime(6));

Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 AS SELECT num_var % dt_var AS res1 FROM t1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC t2;
+-------+----------------+------+-----+---------+-------+
| Field | Type           | Null | Key | Default | Extra |
+-------+----------------+------+-----+---------+-------+
| res1  | decimal(20,10) | YES  |     | NULL    |       |
+-------+----------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

- 在开启精度传递的场景下，使用UNION，如果参与集合运算的查询语句，其查询的字段为常量，且查询的结果数据类型为INT/DECIMAL，则最后返回的精度存在差异。在MySQL 5.7中，返回的精度与UNION左右两侧的顺序有关；在MySQL 8.0中修复了这个问题，返回的精度与UNION左右两侧的顺序无关；在GaussDB中，返回的精度与UNION左右两侧的顺序无关，与MySQL 8.0一致，与MySQL 5.7不一致。

```
-- GaussDB:
m_db=# CREATE TABLE t1 AS (SELECT -23.45 c2) UNION ALL (SELECT -45.678 c2);
INSERT 0 2
m_db=# DESC t1;
Field |     Type     | Null | Key | Default | Extra
-------+--------------+------+-----+---------+-------
c2    | decimal(5,3) | YES  |     |         |
(1 row)
m_db=# CREATE TABLE t2 AS (SELECT -45.678 c2) UNION ALL (SELECT -23.45 c2);
INSERT 0 2
m_db=# DESC t2;
Field |     Type     | Null | Key | Default | Extra
-------+--------------+------+-----+---------+-------
c2    | decimal(5,3) | YES  |     |         |
(1 row)

-- MySQL 5.7:
mysql> CREATE TABLE t1 AS (SELECT -23.45 c2) UNION ALL (SELECT -45.678 c2);
Query OK, 2 rows affected (2.28 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> DESC t1;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| c2    | decimal(6,3) | NO   |     | 0.000   |       |
+-------+--------------+------+-----+---------+-------+
1 row in set (0.00 sec)
mysql> CREATE TABLE t2 AS (SELECT -45.678 c2) UNION ALL (SELECT -23.45 c2);
Query OK, 2 rows affected (2.22 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> DESC t2;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| c2    | decimal(5,3) | NO   |     | 0.000   |       |
+-------+--------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```



```
-- MySQL 8.0:
mysql> CREATE TABLE t1 AS (SELECT -23.45 c2) UNION ALL (SELECT -45.678 c2);
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> DESC t1;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| c2    | decimal(5,3) | NO   |     | 0.000   |       |
+-------+--------------+------+-----+---------+-------+
1 row in set (0.03 sec)
mysql>  CREATE TABLE t2 AS (SELECT -45.678 c2) UNION ALL (SELECT -23.45 c2);
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DESC t2;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| c2    | decimal(5,3) | NO   |     | 0.000   |       |
+-------+--------------+------+-----+---------+-------+
1 row in set (0.02 sec)
```

双冒号转换差异点

GaussDB中使用双冒号将函数入参转换为期望类型可能导致结果超出预期；MySQL中无双冒号功能。

示例：

```
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)
```

decimal 类型差异点

在CREATE TABLE ... AS (SELECT ...)语句中，使用decimal数据类型，若含有前缀0，GaussDB数据库下忽略前缀0，长度计算不包括0，在MySQL 5.7下，长度计算加上前缀0的数量，在MySQL 8.0下，无论存在多少个前缀0，长度计算只加上1。

```
-- GaussDB
m_db=# CREATE TABLE test AS SELECT 004.01 col1;
INSERT 0 1
m_db=# DESC test;
 Field |     Type     | Null | Key | Default | Extra
-------+--------------+------+-----+---------+-------
 col1  | decimal(3,2) | YES  |     |         |
(1 row)

-- MySQL 5.7
mysql> CREATE TABLE test AS SELECT 004.01 col1;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESC test;
+-------+--------------+------+-----+---------+-------+
```



```
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| col1  | decimal(5,2) | NO   |     | 0.00    |       |
+-------+---------------+------+-----+---------+-------+
1 row in set (0.00 sec)

-- MySQL 8.0
mysql> CREATE TABLE test AS SELECT 004.01 col1;
Query OK, 1 row affected (0.23 sec)
Records: 1  Duplicates: 0  Warnings: 0
mysql> DESC test;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| col1  | decimal(4,2) | NO   |     | 0.00    |       |
+-------+---------------+------+-----+---------+-------+
1 row in set (0.01 sec)
```

3.2 系统函数

3.2.1 系统函数兼容性概述

GaussDB数据库兼容绝大多数MySQL的系统函数，但存在部分差异。建议使用M-Compatibility兼容模式下支持的系统函数，避免使用原GaussDB的系统函数。

当前存在原GaussDB的系统函数和MySQL系统函数同名，但是M-Compatibility兼容模式下尚未支持这些函数的情况；**表3-9**中的函数会提示用户在M-Compatibility兼容模式下不支持，**表3-10**中的函数仍然保持原GaussDB系统函数的行为。同名函数会与MySQL的行为产生较大差异，因此建议用户尽量避免使用这些同名函数，只使用M-Compatibility兼容模式下支持的系统函数。

表 3-9 M-Compatibility 兼容模式下提示不支持的同名函数

isEmpty	variance	overlaps	point	stddev_pop
stddev_samp	var_pop	var_samp	-	-

表 3-10 M-Compatibility 兼容模式下保持原 GaussDB 系统函数行为的同名函数

ceil	decode	encode	format	instr
position	round	stddev	row_number	regexp_instr
regexp_like	regexp_replace	regexp_substr	-	-

在参数m_format_dev_version值为s2或以上版本并且参数m_format_behavior_compat_options值包含enable_conflict_funcs的情况下，下表中同名函数的行为会修改为M-Compatibility兼容模式下的函数行为。

表 3-11 M-Compatibility 兼容模式下受 GUC 参数控制的同名函数

ceil	format	instr	position	row_number

说明

- 函数regexp_instr、regexp_like、regexp_replace、regexp_substr在参数m_format_dev_version值为's2'或以上版本并且参数m_format_behavior_compat_options值包含'enable_conflict_funcs'的情况下使用会报错，并提示M-Compatibility兼容模式数据库不支持；其他行为和《开发指南》中的“SQL参考 > 函数和操作符 > 字符处理函数和操作符”章节中的同名函数保持一致。
- M-Compatibility模式下，系统函数存在以下公共差异：
  - 系统函数的返回值类型仅考虑入参node类型为Var（表中数据）和Const（常量输入）类型时的情况与MySQL保持一致，其他情况（如入参为运算表达式、函数表达式等）可能返回值的类型与MySQL有差异。
  - 系统函数在涉及LIMIT与OFFSET同时使用的查表场景下，由于GaussDB和MySQL的执行层机制不同，GaussDB会逐行调用函数，此行为会导致在存在报错的情况下会直接报错且中断执行，但MySQL不会逐行执行故不会报错中断，从而造成返回结果存在不一致。
  - 系统函数的调用不推荐使用pg_catalog.func_name()形式的调用，当被调用函数存在语法形式的入参时（如SELECT pg_catalog.substr('demo' FROM 1 FOR 2)），函数的调用可能存在错误。

3.2.2 流程控制函数

表 3-12 流程控制函数列表

函数名	与MySQL的差异
IF()	当第一个参数为TRUE且第三个参数表达式中存在隐式类型转换错误，或者第一个参数为FALSE且第二个参数表达式中存在隐式类型转换错误时，MySQL会忽略该错误，GaussDB会提示类型转换错误。

函数名	与MySQL的差异

IFNULL()	• 当第一个参数不为NULL且第二个参数表达式中存在隐式类型转换错误时，MySQL会忽略该错误，GaussDB会提示类型转换错误。
	• 当入参类型为float类型时，GaussDB结果值的精度与MySQL 8.0行为保持一致，例如：
	m_db=# CREATE TABLE t1(c1 float);
	CREATE TABLE
	m_db=# INSERT INTO t1 VALUES(2.123);
	INSERT 0 1
	
	-- GaussDB的行为。
	m_db=# SELECT ifnull(c1, c1) FROM t1;
	ifnull
	-------- 	
	2.123
	(1 row)
	
	-- MySQL 5.7的行为。
	mysql> SELECT ifnull(c1, c1) FROM t1;
	+-------------------+
	| ifnull(c1, c1) |
	+-------------------+
	| 2.122999906539917 |
	+-------------------+
	1 row in set (0.00 sec)
	
	-- MySQL 8.0的行为。
	mysql> SELECT ifnull(c1, c1) FROM t1;
	+-----------------+
	| ifnull(c1, c1) |
	+-----------------+
	| 2.123 |
	+-----------------+
	1 row in set (0.00 sec)

函数名	与MySQL的差异

NULLIF()	• 在MySQL 5.7和MySQL 8.0中，函数返回值类型存在差异。鉴于MySQL 8.0更合理，因此GaussDB函数返回值类型兼容MySQL 8.0。
	• 当入参类型为float类型时，GaussDB结果值的精度与MySQL 8.0行为保持一致，例如：
	m_db=# CREATE TABLE t1(c1 float);
	CREATE TABLE
	m_db=# INSERT INTO t1 VALUES(2.123);
	INSERT 0 1
	
	-- GaussDB的行为。
	m_db=# SELECT nullif(c1, 1) FROM t1;
	nullif
	-------- 	
	2.123
	(1 row)
	
	-- MySQL 5.7的行为。
	mysql> SELECT nullif(c1, 1) SELECT t1;
	+-------------------+
	| nullif(c1, 1) |
	+-------------------+
	| 2.122999906539917 |
	+-------------------+
	1 row in set (0.00 sec)
	
	-- MySQL 8.0的行为。
	mysql> SELECT nullif(c1, 1) SELECT t1;
	+---------------+
	| nullif(c1, 1) |
	+---------------+
	| 2.123 |
	+---------------+
	1 row in set (0.00 sec)

3.2.3 日期和时间函数

以下为GaussDB数据库M-Compatibility兼容性日期时间函数公共差异说明。

- 当SELECT子查询中包含且仅包含时间函数，且函数入参包含表中的列时，使用算数运算符（如+、-、*、/、取反等）对结果进行运算时，会截断日期与时间函数返回值后再进行算数运算。

```
m_db=# CREATE TABLE t1(int_var int);
CREATE TABLE
m_db=# INSERT INTO t1 VALUES(100);
INSERT 0 1
m_db=# SELECT (SELECT (1 * DATE_ADD('2020-10-20', interval int_var microsecond))) AS a FROM t1;
-- 不进行截断处理。
       a
----------------
 20201020000000
(1 row)

m_db=# SELECT (1 * (SELECT DATE_ADD('2020-10-20', interval int_var microsecond))) AS a FROM t1;
-- 进行截断处理。
  a
------
 2020
(1 row)
```

```
m_db=# SELECT 1 * a FROM (SELECT (SELECT 1 * DATE_ADD('2020-10-20', interval int_var
microsecond)) AS a FROM t1) AS t2; -- 不进行截断处理。
     1 * a
----------------
 20201020000000
(1 row)

m_db=# SELECT 1 * a FROM (SELECT (SELECT DATE_ADD('2020-10-20', interval int_var microsecond))
AS a FROM t1) AS t2; -- 进行截断处理。
 1 * a
-------
  2020
(1 row)
```

表 3-13 日期与和时间函数列表

函数名	与MySQL的差异
ADDDATE()	-
ADDTIME()	-
CONVERT_TZ()	-
CURDATE()	-
CURRENT_DATE()/CURRENT_DATE	-
CURRENT_TIME()/CURRENT_TIME	MySQL入参整型值会按照一字节最大值255整数回绕（例：SELECT CURRENT_TIME(257) == SELECT CURRENT_TIME(1)）。 GaussDB只支持[0,6]合法值，其他值报错。
CURRENT_TIMESTAMP()/CURRENT_TIMESTAMP	
CURTIME()	
LOCALTIME()/LOCALTIME	
LOCALTIMESTAMP/LOCALTIMESTAMP()	
NOW()	
SYSDATE()	
UTC_TIME()	
UTC_TIMESTAMP()	
DATE()	-
DATE_ADD()	-
DATE_FORMAT()	-

函数名	与MySQL的差异
DATE_SUB()	-
DATEDIFF()	-
DAY()	-
DAYNAME()	-
DAYOFMONTH()	-
DAYOFWEEK()	-
DAYOFYEAR()	-
EXTRACT()	-
FROM_DAYS()	-
FROM_UNIXTIME()	-
GET_FORMAT()	-
HOUR()	-
LAST_DAY()	-
MAKEDATE()	-
MAKETIME()	-
MICROSECOND()	-
MINUTE()	-
MONTH()	-
MONTHNAME()	-
PERIOD_ADD() PERIOD_DIFF()	MySQL8.0修复了以下问题，在以下场景中该函数的行为与MySQL8.0版本保持一致： ● 整数溢出处理的行为。 MySQL在5.7版本，此函数入参和结果的最大值都为2^32=4294967296，在入参或结果的period对应的月份累加值以及month_number超过uint32范围时存在整数回绕问题 ● 负数period的表现。 MySQL在5.7版本，会将负数年份解析为异常值而不是报错。GaussDB入参或结果（如100年1月减去10000月）出现负数时报错。 ● period月份越界的表现。 MySQL在5.7版本中，若月份大于12或等于0，例如200013、199900，会将其顺延到之后的年份，或者将0月作为上一年12月处理。

函数名	与MySQL的差异
QUARTER()	-
SEC_TO_TIME()	-
SECOND()	-
STR_TO_DATE()	-
SUBDATE()	-
SUBTIME()	-
TIME()	-
TIME_FORMAT()	-
TIME_TO_SEC()	-
TIMEDIFF()	-
TIMESTAMP()	-
TIMESTAMPADD()	-
TIMESTAMPDIFF()	-
TO_DAYS()	-
TO_SECONDS()	在MySQL 5.7版本中，此函数的精度信息有误。 开启精度传递参数下，GaussDB精度信息正常，和MySQL 8.0版本保持一致。
UNIX_TIMESTAMP()	MySQL会根据入参是否存在小数位，决定返回定点型还是整型。当前GaussDB在内层嵌套操作符或函数时，返回的类型与MySQL可能存在不同。当内层节点返回定点、浮点、字符型、时间类型（不包括DATE类型）时，MySQL可能返回整型，GaussDB会返回定点型。
UTC_DATE()	-
WEEK()	-
WEEKDAY()	-
WEEKOFYEAR()	-
YEAR()	-
YEARWEEK()	-

3.2.4 字符串函数

当GaussDB使用的字符编码是SQL_ASCII时，服务器会根据ASCII标准对字节值0~127进行解释，而字节值128~255则当作无法解析的字符。如果该函数的输入输出包含了

任何非ASCII数据，数据库将无法帮助用户转换或者校验非ASCII字符，从而与MySQL的行为产生较大差异。

表 3-14 字符串函数列表

函数名	与MySQL的差异
ASCII()	-
BIT_LENGTH()	-
CHAR_LENGTH()	如果数据库字符集是SQL_ASCII，该函数会返回字节数而非字符数。
CHARACTER_LENGTH()	
CONCAT()	-
CONCAT_WS()	-
HEX()	-
LENGTH()	-
LPAD()	MySQL默认最大填充长度为1398101，GaussDB默认最大长度为1048576。在不同字符集下，最大填充长度会有差异，例如字符集为GBK时，GaussDB默认最大长度为2097152。
RPAD()	
MD5()	当BINARY类型插入字符串长度小于目标长度时，GaussDB填充符和MySQL不同；因此导致入参为BINARY类型时，函数结果和MySQL不一致。
RANDOM_BYTES()	GaussDB与MySQL都使用OPENSSL生成随机字符串。GaussDB使用OPENSSL3.x.x生成随机字符串，与使用OPENSSL1.x.x版本的MySQL相比性能可能存在劣化。
REPEAT()	-

REPLACE()	当第三个入参为null，且第二个入参的字符串长度不为0时，GaussDB返回NULL，MySQL可能返回第一个参数的字符。例如：
	-- GaussDB的行为：
	m_db=# select replace('1.23', binary(1.1), null);
	replace
	--------- 	
	
	(1 row)
	
	-- MySQL的行为：
	mysql> select replace('1.23', binary(1.1), null);
	+-------------------------------------+
	| replace('1.23', binary(1.1), null) |
	+-------------------------------------+
	| 1.23 |
	+-------------------------------------+
	1 row in set (0.00 sec)

SHA()/SHA1()	-
SHA2()	-

函数名	与MySQL的差异
SPACE()	-
STRCMP()	-
FIND_IN_SET()	-
LCASE()	-
LEFT()	-
LOWER()	-
LTRIM()	-
REVERSE()	-
RIGHT()	-
RTRIM()	-
SUBSTR()	第一个入参节点返回的字符序为BINARY时，MySQL可能依旧以不同的字符序逻辑处理（取决于内层嵌套的函数），而GaussDB以BINARY字符序进行函数处理，导致截取的字节长度不同。
SUBSTRING()	
SUBSTRING_INDEX()	• 在第三个入参为负数时，MySQL与GaussDB的比较逻辑不同，会导致结果可能存在差异。 • 当第三个入参为正数时，由于MySQL 5.7以int32位存储，会存在回绕问题导致结果不正确。MySQL 8.0修复了该问题以int64为存储，GaussDB以8.0为准。当入参值超过2^63 -1时，也会发生回绕，可能导致第三个参数获取为负数，结果存在差异。
TRIM()	-
UCASE()	-
UPPER()	-
UNHEX()	-
FIELD()	-
COMPRESS()	-
UNCOMPRESS()	-
UNCOMPRESS_LENGTH()	-
EXPORT_SET()	-
POSITION()	-
LOCATE()	-

函数名	与MySQL的差异
CHAR()	● CHAR函数指定字符集时，若转码失败，GaussDB产生报错，MySQL提示WARNING并返回NULL。 ● MySQL在参数为ASCII表中第0~31个和127个码值时，返回结果不可见，GaussDB会以\x01、\x02等16进制返回。 ● MySQL中CHAR函数入参个数无限制，GaussDB函数的入参不超过8192个。
ELT()	MySQL中ELT函数入参个数无限制，GaussDB函数的入参不超过8192个。
FORMAT()	-
BIN()	-
MAKE_SET()	MySQL 5.7版本，当MAKE_SET函数选中的第一个参数为整型、浮点型或定点型且返回结果中存在非ASCII字符，显示结果可能为乱码；GaussDB显示结果正常，和MySQL 8.0版本保持一致。
TO_BASE64()	-
FROM_BASE64()	-
ORD()	-
MID()	-
QUOTE()	● 当入参字符串中含“\0”时，GaussDB中由于字符集不支持导致无法输入。由转义字符导致的本函数与MySQL的差异，与本函数无关。 ● GaussDB最大支持1GB数据传输，str入参长度最大支持536870908字节，函数返回结果字符串最大支持1GB。 ● 入参为固定长度BINARY类型时，对于未填充字符：MySQL默认补充空字符“\0”，GaussDB默认补充空格。
INSERT()	-
INSTR()	-
OCTET_LENGTH()	-

3.2.5 类型转换函数

表 3-15 类型转换函数列表

函数名	与MySQL的差异

CAST()	• 由于函数执行机制不同，在cast函数嵌套其他函数（如greatest、least等）时，内层函数返回小于1的值，结果与MySQL不一致。
	--GaussDB:
	m_db=# SELECT cast(least(1.23, 1.23, 0.23400) AS date);
	WARNING: Incorrect datetime value: '0.23400'
	CONTEXT: referenced column: cast
	cast
	------ 	
	(1 row)
	
	--MySQL 5.7:
	mysql> SELECT cast(least(1.23, 1.23, 0.23400) AS date);
	+---+
	| cast(least(1.23, 1.23, 0.23400) as date) |
	+---+
	| 0000-00-00 |
	+---+
	1 row in set (0.00 sec)
	• GaussDB支持使用CAST(expr AS FLOAT[(p)])或CAST(expr AS DOUBLE)将表达式转换为浮点类型，MySQL 5.7版本不支持此转换。
	• 对于CAST嵌套子查询场景，如果子查询语句返回的是FLOAT类型，GaussDB返回的是准确的数值，MySQL 5.7版本返回失真数值，BINARY函数使用CAST实现，同理。
	--GaussDB
	m_db=# CREATE TABLE sub_query_table(myfloat float);
	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) |
	+---+

函数名	与MySQL的差异

	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 |
	+---+
	1 row in set (0.00 sec)
	• GaussDB在JSON数据类型显式转换后运用于精度计算时，与MySQL 5.7不一致，与MySQL 8.0一致，计算时精度与使用JSON类型表中数据精度保持一致。
	示例：
	--GaussDB
	test=# SET
	m_format_behavior_compat_options='enable_precision_decimal';
	SET
	
	test=# DROP TABLE tt01;
	DROP TABLE
	
	test=# CREATE TABLE tt01 AS SELECT -cast('98.7654321' AS json) AS c1;
	INSERT 0 1
	
	test=# DESC tt01;
	Field | Type | Null | Key | Default | Extra
	---+---+---+---+---+---
	c1 | double | YES | | |
	(1 row)
	
	test=# SELECT * FROM tt01;
	c1
	--- 	
	-98.7654321
	(1 row)
	
	--MySQL 5.7
	mysql> SELECT version();
	+---+
	| version() |
	+---+
	| 5.7.44-debug-log |
	+---+
	1 row in set (0.00 sec)
	
	mysql> DROP TABLE tt01;
	Query OK, 0 rows affected (0.02 sec)
	
	mysql> CREATE TABLE tt01 AS SELECT -cast('98.7654321' AS json) AS c1;
	Query OK, 1 row affected (0.03 sec)
	Records: 1 Duplicates: 0 Warnings: 0
	
	mysql> DESC tt01;
	+---+---+---+---+---+---+
	| Field | Type | Null | Key | Default | Extra |
	+---+---+---+---+---+---+
	| c1 | double(17,0) | YES | | NULL | |
	+---+---+---+---+---+---+
	1 row in set (0.00 sec)
	
	mysql> SELECT * FROM tt01;
	+---+

函数名	与MySQL的差异
		c1 | +------+ | -99 | +------+ 1 row in set (0.00 sec) --MySQL 8.0 mysql> SELECT version(); +--------------+ | version() | +--------------+ | 8.0.36-debug | +--------------+ 1 row in set (0.00 sec) mysql> DROP TABLE tt01; Query OK, 0 rows affected (0.05 sec) mysql> CREATE TABLE tt01 AS SELECT -cast('98.7654321' AS json) AS c1; Query OK, 1 row affected (0.12 sec) Records: 1 Duplicates: 0 Warnings: 0 mysql> DESC tt01; +-------+--------+------+-----+---------+-------+ | Field	Type	Null	Key	Default	Extra | +-------+--------+------+-----+---------+-------+ | c1	double	YES		NULL	| +-------+--------+------+-----+---------+-------+ 1 row in set (0.01 sec) mysql> SELECT * FROM tt01; +-------------+ | c1 | +-------------+ | -98.7654321 | +-------------+ 1 row in set (0.00 sec)
CONVERT()	GaussDB支持使用CONVERT(expr, FLOAT[(p)])或CONVERT(expr, DOUBLE)将表达式转换为浮点类型，MySQL 5.7版本不支持此转换。

3.2.6 加密函数

表 3-16 加密函数列表

函数名	与MySQL的差异
AES_DECRYPT() AES_ENCRYPT()	• ecb为不安全加密模式，GaussDB不支持，默认为cbc模式。 • GaussDB中，当指定数据库使用的字符编码是SQL_ASCII时，服务器把字节值0~127根据ASCII标准解释，而字节值128~255则当作无法解析的字符；如果该函数的输入输出包含了任何非ASCII数据，数据库将无法帮助用户转换或者校验非ASCII字符。 • GUC参数： block_encryption_mode不支持设置数字。

函数名	与MySQL的差异
PASSWORD()	• MySQL中可以通过GUC参数old_passwords控制生成密码的哈希方式： – old_passwords的默认值为0。 – old_passwords为0：表示使用MySQL 4.1 native hashing加密。 – old_passwords为2：表示使用SHA-256 hashing加密。 • GaussDB中没有实现GUC参数old_passwords，password函数的行为只与默认（即old_passwords为0）的行为保持一致。 • 当BINARY类型插入字符串长度小于目标长度时，GaussDB填充符和MySQL不同；因此入参为BINARY类型时，函数结果和MySQL不一致。

3.2.7 比较函数

表 3-17 比较函数列表

函数名	与MySQL的差异
COALESCE()	union distinct场景下，返回值精度与MySQL不完全一致。 当第一个不为NULL的参数的后续参数表达式中存在隐式类型转换错误时，MySQL会忽略该错误，GaussDB会提示类型转换错误。当参数为MIN函数、MAX函数时，返回值类型与MySQL不一致。
INTERVAL()	-
GREATEST()	当该函数入参含有NULL且在WHERE关键字之后调用，返回结果与MySQL 5.7不一致，此处为MySQL 5.7存在的问题，MySQL 8.0修复了该问题，目前GaussDB和MySQL 8.0保持一致。
LEAST()	

函数名	与MySQL的差异

ISNULL()	• 函数返回值类型在MySQL 5.7和MySQL 8.0中存在差异，结合MySQL 8.0的行为更为合理，因此函数返回值类型兼容MySQL 8.0。
	• 内层嵌套部分聚合函数时，部分场景返回结果MySQL 5.7和MySQL 8.0中存在差异，结合MySQL 8.0的行为更为合理，因此函数返回值兼容MySQL 8.0。
	m_db=# SELECT isnull(avg(1.23));
	?column?
	---------- 	
	f
	(1 row)
	
	m_db=# SELECT isnull(group_concat(1.23));
	?column?
	---------- 	
	f
	(1 row)
	
	m_db=# SELECT isnull(max('1.23'));
	?column?
	---------- 	
	f
	(1 row)
	
	m_db=# SELECT isnull(min(1/2));
	?column?
	---------- 	
	f
	(1 row)
	
	m_db=# SELECT isnull(std(3.14159 * 1.2345));
	?column?
	---------- 	
	f
	(1 row)
	
	m_db=# SELECT isnull(sum('0.23400'));
	?column?
	---------- 	
	f
	(1 row)

3.2.8 聚合函数

表 3-18 聚合函数列表

函数名	与MySQL的差异
AVG()	• GaussDB中当expr中的列为BIT、BOOL、整数类型，且所有行的和超过BIGINT的范围时，会发生溢出导致整数翻转。 • GaussDB在AVG函数入参为TEXT/BLOB类型时行为存在差异： – MySQL 5.7中，AVG(TEXT/BLOB)返回值类型为MEDIUMTEXT类型；MySQL 8.0中，AVG(TEXT/BLOB)返回值类型为DOUBLE类型。 – 在GaussDB中，AVG(TEXT/BLOB)返回值类型与MySQL 8.0版本保持一致。

BIT_AND()	BIT_AND函数入参为NULL且被其他函数嵌套时行为有差异：在MySQL 5.7中，结果为-1；在MySQL 8.0中，结果为NULL；在GaussDB中，此函数嵌套的表现与MySQL 8.0版本保持一致。
	-- GaussDB：
	m_db=# SELECT acos(bit_and(null));
	acos
	------ 	
	
	(1 row)
	-- MySQL 5.7：
	mysql> SELECT acos(bit_and(null));
	+----------------------+
	| acos(bit_and(null)) |
	+----------------------+
	| 3.141592653589793 |
	+----------------------+
	1 row in set (0.03 sec)
	
	-- MySQL 8.0
	mysql> SELECT acos(bit_and(null));
	+----------------------+
	| acos(bit_and(null)) |
	+----------------------+
	| NULL |
	+----------------------+
	1 row in set (0.01 sec)

BIT_OR()	-
BIT_XOR()	-
COUNT()	GaussDB支持count(tablename.*)语法，MySQL不支持。

函数名	与MySQL的差异

GROUP_CONCAT()	• GaussDB中当GROUP_CONCAT参数中同时有DISTINCT和ORDER BY语法时，所有ORDER BY后的表达式必须也在DISTINCT的表达式之中。
	• GaussDB中GROUP_CONCAT(... ORDER BY 数字)不代表按照第几个参数的顺序，数字只是一个常量表达式，相当于不排序。
	• GaussDB中使用参数group_concat_max_len限制GROUP_CONCAT最大返回长度，超长截断，目前能返回的最大长度是1073741823，小于MySQL。
	• 默认UTF8字符集下，由于GaussDB的UTF8字符集的最大字节数与MySQL的UTF8字符集最大字节数不同。会导致创建的表结构与MySQL存在差异。
	-- GaussDB：
	m_db=# SET
	m_format_behavior_compat_options='enable_precision_decimal';
	SET
	m_db=# CREATE TABLE t1 AS SELECT * FROM (SELECT CASE WHEN 1 < 2 THEN group_concat(1.23, 3.24) ELSE 12.34 END v1) c1;
	INSERT 0 1
	m_db=# DESC t1;
	Field | Type | Null | Key | Default | Extra
	-------+--------------+------+-----+---------+-------
	v1 | varchar(256) | YES | | |
	(1 row)
	-- MySQL 5.7：
	mysql> CREATE TABLE t1 AS SELECT * FROM (SELECT CASE WHEN 1 < 2 THEN group_concat(1.23, 3.24) ELSE 12.34 END v1) c1;
	Query OK, 1 row affected (0.01 sec)
	Records: 1 Duplicates: 0 Warnings: 0
	
	mysql> DESC t1;
	+-------+--------------+------+-----+---------+-------+
	| Field | Type | Null | Key | Default | Extra |
	+-------+--------------+------+-----+---------+-------+
	| v1 | varchar(341) | YES | | NULL | |
	+-------+--------------+------+-----+---------+-------+
	1 row in set (0.00 sec)
	• GROUP_CONCAT函数作为NULLIF函数的入参，嵌套场景行为有差异：在MySQL 5.7中，NULLIF入参嵌套GROUP_CONCAT与非嵌套GROUP_CONCAT的数值会判断为相等，返回NULL；在MySQL 8.0中，因精度差异会判断为不等；在GaussDB中，此函数嵌套的表现与MySQL 8.0版本保持一致。
	-- GaussDB：
	m_db=# SELECT nullif(group_concat(1/7), 1/7);
	nullif
	-------- 	
	0.1429
	(1 row)
	-- MySQL 5.7：
	mysql> SELECT nullif(group_concat(1/7), 1/7);
	+--------------------------------+
	| nullif(group_concat(1/7), 1/7) |
	+--------------------------------+
	| NULL |
	+--------------------------------+
	1 row in set (0.00 sec)
	-- MySQL 8.0：

函数名	与MySQL的差异
	mysql> SELECT nullif(group_concat(1/7), 1/7); +--------------------------------+ | nullif(group_concat(1/7), 1/7) | +--------------------------------+ | 0.1429 | +--------------------------------+ 1 row in set (0.00 sec)
MAX() MIN()	• 当参数为非表字段时，MAX函数返回值类型和MySQL 5.7不一致。 • 开启精度传递时，MAX函数嵌套time、date、datetime、timestamp类型的时间间隔运算，返回值及返回类型与MySQL 8.0保持一致。 • 开启精度传递时，MAX函数和INTERVAL的时间间隔运算，返回值及返回类型与MySQL 8.0保持一致。
SUM()	GaussDB中当expr中的列为BIT、BOOL、整数类型，且所有行的和超过BIGINT的范围时，会发生溢出导致整数翻转。
STD()	-

函数名	与MySQL的差异
聚合函数	• GaussDB中指定DISTINCT且SQL语句包含GROUP BY子句时，不对结果进行排序，MySQL会进行排序。 • ORDER BY语句中包含聚合函数时GaussDB不报错，MySQL会报错。 • 在未开启精度传递（没有设置m_format_behavior_compat_options = 'enable_precision_decimal'）的情况下，当聚合函数以其他函数、操作符或SELECT子句等表达式作为入参时（如SELECT sum(abs(n)) FROM t），聚合函数将获取不到入参表达式传递的精度信息，导致函数的结果精度与MySQL有差异。 • 聚合函数的结果与数据输入顺序相关，不同的数据输入顺序会导致结果存在差异。 – 例如与ORDER BY同时使用时，改变了聚合函数的执行顺序，会导致结果与MySQL不一致。 --准备基表: CREATE TABLE test_n(col_unumeric1 decimal(4,3) unsigned, col_znumeric2 decimal(3,2) unsigned zerofill, col_znumeric3 decimal(5,3) unsigned zerofill); Query OK, 0 rows affected (0.01 sec) INSERT INTO test_n VALUES(1.010, 2.02, 3.303),(1.190, 2.29, 3.339), (1.180, 2.28, 3.338); Query OK, 3 rows affected (0.00 sec) Records: 3 Duplicates: 0 Warnings: 0 CREATE TABLE test_n_2(col_unumeric1 decimal(4,3) unsigned, col_znumeric2 decimal(3,2) unsigned zerofill, col_znumeric3 decimal(5,3) unsigned zerofill); Query OK, 0 rows affected (0.02 sec) INSERT INTO test_n_2 VALUES(1.180, 2.28, 3.338),(1.190, 2.29, 3.339), (1.010, 2.02, 3.303); Query OK, 3 rows affected (0.00 sec) Records: 3 Duplicates: 0 Warnings: 0 CREATE TABLE IF NOT EXISTS fun_op_case_tb_1 (id int, name varchar(20), col_unumeric1 NUMERIC(4,3) unsigned, col_znumeric2 DECIMAL(3,2) zerofill,col_znumeric3 DEC(5,3) zerofill); CREATE TABLE INSERT INTO fun_op_case_tb_1 (id, name, col_unumeric1, col_znumeric2, col_znumeric3) VALUES (1, '计算机', 1.11, 2.12, 3.133), (2, '计算机', 2.11, 2.22, 3.233), (3, '计算机', 3.11, 2.32, 3.333), (4, '计算机', 1.41, 2.42, 3.343), (5, '计算机', 1.51, 2.52, 3.353), (6, '计算机', 1.61, 2.26, 3.363), (7, '计算机', 1.17, 2.27, 3.337), (8, '计算机', 1.18, 2.28, 3.338), (9, '计算机', 1.19, 2.29, 3.339), (10, '计算机', 1.01, 2.02, 3.303), (1,'软件', 1.11, 2.12, 3.133), (2,'软件', 2.11, 2.22, 3.233), (3,'软件', 3.11, 2.32, 3.333), (4,'软件', 1.41, 2.42, 3.343), (5,'软件', 1.51, 2.52, 3.353), (6,'软件', 1.61, 2.26, 3.363),

函数名	与MySQL的差异

```
(7,'软件', 1.17, 2.27, 3.337),
(8,'软件', 1.18, 2.28, 3.338),
(9,'软件', 1.19, 2.29, 3.339),
(10,'软件', 1.01, 2.02, 3.303),
(1, '数据库', 1.11, 2.12, 3.133),
(2, '数据库', 2.11, 2.22, 3.233),
(3, '数据库', 3.11, 2.32, 3.333),
(4, '数据库', 1.41, 2.42, 3.343),
(5, '数据库', 1.51, 2.52, 3.353),
(6, '数据库', 1.61, 2.26, 3.363),
(7, '数据库', 1.17, 2.27, 3.337),
(8, '数据库', 1.18, 2.28, 3.338),
(9, '数据库', 1.19, 2.29, 3.339),
(10, '数据库', 1.01, 2.02, 3.303);
INSERT 0 30
--GaussDB：
m_db=# SELECT * FROM test_n;
 col_unumeric1 | col_znumeric2 | col_znumeric3
---------------+---------------+---------------
         1.010 |          2.02 |        03.303
         1.190 |          2.29 |        03.339
         1.180 |          2.28 |        03.338
m_db=# SELECT * FROM test_n_2;
 col_unumeric1 | col_znumeric2 | col_znumeric3
---------------+---------------+---------------
         1.180 |          2.28 |        03.338
         1.190 |          2.29 |        03.339
         1.010 |          2.02 |        03.303
m_db=# SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3))
FROM test_n_2 ;
         std
---------------------
 0.24779023386727736
(1 row)
m_db=# SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3))
FROM test_n ;
         std
---------------------
 0.24779023386727742
(1 row)

m_db=# SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3))
FROM fun_op_case_tb_1 GROUP BY name ORDER BY name;
        std
--------------------
 1.8167446160646796
 1.8167446160646794
 1.8167446160646796
(3 rows)

--MySQL:
mysql> SELECT * FROM test_n;
+---------------+---------------+---------------+
| col_unumeric1 | col_znumeric2 | col_znumeric3 |
+---------------+---------------+---------------+
|         1.010 |          2.02 |        03.303 |
|         1.190 |          2.29 |        03.339 |
|         1.180 |          2.28 |        03.338 |
+---------------+---------------+---------------+
3 rows in set (0.00 sec)
mysql> SELECT *FROM test_n_2;
+---------------+---------------+---------------+
| col_unumeric1 | col_znumeric2 | col_znumeric3 |
+---------------+---------------+---------------+
|         1.180 |          2.28 |        03.338 |
```

函数名	与MySQL的差异

	1.190 | 2.29 | 03.339 |
	1.010 | 2.02 | 03.303 |
	+---------------+---------------+---------------+
	3 rows in set (0.00 sec)
	mysql> SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3)) FROM test_n_2 ;
	+---+
	| std(col_unumeric1*(col_znumeric2 | col_znumeric3)) |
	+---+
	| 0.24779023386727736 |
	+---+
	1 row in set (0.00 sec)
	mysql> SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3)) FROM test_n;
	+---+
	| std(col_unumeric1*(col_znumeric2 | col_znumeric3)) |
	+---+
	| 0.24779023386727742 |
	+---+
	1 row in set (0.00 sec)
	
	mysql> SELECT std(col_unumeric1*(col_znumeric2 | col_znumeric3)) FROM fun_op_case_tb_1 GROUP BY name ORDER BY name;
	+---+
	| std(col_unumeric1*(col_znumeric2 | col_znumeric3)) |
	+---+
	| 1.8167446160646794 |
	| 1.8167446160646794 |
	| 1.8167446160646794 |
	+---+
	3 rows in set (0.00 sec)
	
	--删除基表：
	DROP TABLE test_n;
	DROP TABLE
	DROP TABLE test_n_2;
	DROP TABLE
	DROP TABLE fun_op_case_tb_1;
	DROP TABLE
	– 例如与WITH ROLLUP同时使用时，改变了聚合函数的执行顺序，会导致结果与MySQL不一致。
	--基表准备：
	CREATE TABLE IF NOT EXISTS t1 (name VARCHAR(20), c1 INT(100), c2 FLOAT(7,5));
	INSERT INTO t1 VALUES
	('计算机', 666,-55.155),
	('计算机', 789,-15.593),
	('计算机', 928,-53.963),
	('计算机', 666,-54.555),
	('计算机', 666,-55.555),
	('数据库', 666,-55.155),
	('数据库', 789,-15.593),
	('数据库', 928,-53.963),
	('数据库', 666,-54.555),
	('数据库', 666,-55.555);
	
	--GaussDB：
	m_db=# SELECT name, std(c1/c2) c5 FROM t1 GROUP BY name WITH rollup;
	name | c5
	--------+--------------------
	数据库 | 15.02396266299967
	计算机 | 15.023962662999669
	| 15.02396266299967

函数名	与MySQL的差异

	(3 rows)
	
	--MySQL
	mysql> SELECT name, std(c1/c2) c5 FROM t1 GROUP BY name WITH rollup;
	+-----------+--------------------+
	| name | c5 |
	+-----------+--------------------+
	| 数据库 | 15.023962662999669 |
	| 计算机 | 15.023962662999669 |
	| NULL | 15.02396266299967 |
	+-----------+--------------------+
	3 rows in set (0.00 sec)
	
	--删除基表：
	DROP TABLE t1;
	DROP TABLE
	● 聚合函数与GROUP BY同时存在的场景下，存在中间结果为DECIMAL数据类型参与运算时，MySQL存在数据失真问题，GaussDB保留完整精度的数据。
	--基表准备：
	CREATE TABLE IF NOT EXISTS fun_op_case_tb_1 (id int,name varchar(20),col_znumeric2 DECIMAL(3,2) zerofill,col_znumeric3 DEC(5,3) zerofill, col_bit1 BIT(3), col_time2 time);
	
	INSERT INTO fun_op_case_tb_1 VALUES
	(1, '计算机', 0.01, 3.130, b'101', '08:30:23.01'),
	(2, '计算机', 1.20, 30.990, b'101', '08:30:23.01'),
	(3, '计算机', 1.33, 43.500, b'101', '08:30:23.01'),
	(4, '计算机', 2.24, 30.990, b'101', '08:30:23.01'),
	(5, '计算机', 1.25, 43.600, b'101', '08:30:23.01'),
	(6,'计算机',2.20,'20.900',b'101','08:30:23.01'),
	(7,'计算机',2.20,'20.900',b'101','08:30:23.01'),
	(8,'计算机',2.20,'20.900',b'101','08:30:23.01'),
	(9,'计算机',2.29,'22.780',b'101','08:30:23.01'),
	(10,'计算机',2.02,'20.900',b'101','08:30:23.01');
	
	--GaussDB：
	m_db=# SET m_format_behavior_compat_options='enable_precision_decimal';
	m_db=# SELECT avg(col_znumeric3/col_znumeric2) FROM fun_op_case_tb_1 WHERE id<=10 GROUP BY name;
	avg
	---------------- 	
	46.90407212526
	(1 row)
	m_db=# SELECT sum(col_bit1/col_time2) FROM fun_op_case_tb_1 WHERE id<=10 GROUP BY name;
	sum
	-------- 	
	0.0006
	(1 rows)
	
	--MySQL：
	mysql> SELECT avg(col_znumeric3/col_znumeric2) FROM fun_op_case_tb_1 WHERE id<=10 GROUP BY name;
	+----------------------------------+
	| avg(col_znumeric3/col_znumeric2) |
	+----------------------------------+
	| 46.90407213000 |
	+----------------------------------+
	1 row in set (0.00 sec)
	mysql> SELECT sum(col_bit1/col_time2) FROM fun_op_case_tb_1 WHERE id<=10 GROUP BY name;

函数名	与MySQL的差异
	+--------------------------+ | sum(col_bit1/col_time2) | +--------------------------+ | 0.0010 | +--------------------------+ 1 row in set (0.00 sec) --删除基表： DROP TABLE fun_op_case_tb_1; DROP TABLE

3.2.9 JSON 函数

JSON函数差异说明：对于JSON函数和其他字符入参函数，如果输入中包含转义字符，默认情况下会与MySQL有一定差异。要实现与MySQL的兼容，需要设置GUC参数standard_conforming_strings取值为off，在这种情况下，转义字符的处理将与MySQL兼容，但是使用转义字符\f、\Z、\0和\uxxxx的场景会与MySQL存在差异。

表 3-19 JSON 函数列表

函数名	与MySQL的差异
JSON_APPEND()	-
JSON_ARRAY()	-
JSON_ARRAY_APPEND()	-
JSON_ARRAY_INSERT()	-
JSON_CONTAINS()	-
JSON_CONTAINS_PATH()	-
JSON_DEPTH()	-
JSON_EXTRACT()	-
JSON_INSERT()	-
JSON_KEYS()	-
JSON_LENGTH()	-
JSON_MERGE()	-
JSON_MERGE_PATCH()	-
JSON_MERGE_PRESERVE()	-

函数名	与MySQL的差异
JSON_OBJECT()	-
JSON_QUOTE()	-
JSON_REMOVE()	-
JSON_REPLACE()	-
JSON_SEARCH()	-
JSON_SET()	-
JSON_TYPE()	-

JSON_UNQUOTE()	在转义字符中\0和\uxxxx的场景与MySQL有差异：
	SELECT json_unquote('"\0"');
	
	mysql> SELECT json_unquote('"\0"');
	ERROR 3141 (22032): Invalid JSON text in argument 1 to function json_unquote: "Missing a closing quotation mark in string." at position 1.
	
	m_db=# SELECT json_unquote('"\0"');
	ERROR: invalid byte sequence for encoding "UTF8": 0x00

JSON_VALID()	-

3.2.10 窗口函数

表 3-20 窗口函数列表

函数名	与MySQL的差异
LAG() LEAD()	● 偏移量N的取值范围不同： MySQL中，N只允许是在范围[0, 2^{63}-1]整数值。 GaussDB中，N只允许是在范围[0, 2^{31}-1]整数值。 ● 偏移量N的取值形式不同： – MySQL中，取值形式如下： – 常量字面量的无符号整数。 – prepare语句中使用?声明的标记参数。 – 用户自定义的变量。 – 存储过程中的局部变量。 – GaussDB中，取值形式如下： – 常量字面量的无符号整数。 – 不支持在prepare语句中使用?声明的标记参数（prepare语句当前有差异）。 – 用户自定义的变量。 – 不支持使用存储过程中的局部变量（PLSQL当前不支持）。 ● 该函数作为子查询结合CREATE TABLE AS使用时，单独执行该函数的子查询语句没有报错或告警时： – GaussDB在严格模式和宽松模式下，CREATE TABLE AS语句执行成功，建表成功。 – MySQL在严格模式下，CREATE TABLE AS语句执行可能报错，建表失败。
ROW_NUMBER()	-
RANK()	-
DENSE_RANK()	-
FIRST_VALUE()	-
LAST_VALUE()	-
PERCENT_RANK()	-
NTILE()	-

函数名	与MySQL的差异

窗口函数	● ORDER BY子句排序时，对于NULL值的排序不同：
	– MySQL中，NULL值默认升序排在前面。
	– GaussDB中，NULL值默认升序排在后面。
	● OVER子句中的ORDER BY子句与PARTITION BY子句中使用列别名：
	– MySQL不支持使用列别名。
	– GaussDB支持使用列别名。
	● 当入参为表达式（如1 / col1）形式时，结果的精度存在差异：
	– MySQL会先计算表达式的结果并对表达式结果进行四舍五入，导致最终结果精度下降。
	– GaussDB不会对表达式的结果进行四舍五入。
	● 二进制字符串显示存在差异：
	– MySQL中，将二进制字符串编码为16进制后的编码值进行显示（例如'-4'会显示成编码后的0x2D34）。
	– GaussDB中，显示原字符串的值（例如'-4'会保持显示'-4'）。
	● 使用CREATE TABLE AS语法创建表，指定DESC查看表结构时，存在以下差异：
	– MySQL8.0中：
	– 表中的列类型（Type）为BIGINT类型或INT类型时不显示宽度。
	– 表中的列类型（Type）的宽度为0时，显示宽度，如binary(0)。
	– GaussDB中：
	– 表中的列类型（Type）为BIGINT类型或INTEGER类型时会显示宽度。
	– 表中的列类型（Type）的宽度为0时，不显示宽度，如binary(0)只显示成binary。
	● 窗口函数的执行结果依赖于表数据的顺序，在一些场景下（如存在GROUP BY、WHERE、HAVING的场景），GaussDB与MySQL的表数据顺序有差异，会影响窗口函数的执行结果，例如：
	– GaussDB的行为：
	`-- 预置表数据。`
	`m_db=# CREATE TABLE t1(id int,name varchar(20),age int);`
	`CREATE TABLE`
	`m_db=# INSERT INTO t1(id, name,age) VALUES (1, 'zwt',90), (2, 'dda',85), (3, 'aab',90), (4, 'aac',78), (5, 'aad',85), (6, 'aae',92), (7, 'aaf',78);`
	`INSERT 0 7`
	`m_db=# INSERT INTO t1(id, name,age) VALUES (1, 'zwt',90), (2, 'dda',85), (3, 'aab',90), (4, 'aac',78), (5, 'aad',85), (6, 'aae',92), (7, 'aaf',78);`
	`INSERT 0 7`

函数名	与MySQL的差异

	-- 表数据的顺序与MySQL有差异导致last_value的取值结果有差异。
	m_db=# SELECT age, last_value(age) over() FROM t1 WHERE id > 0
	GROUP BY age HAVING age > 10;
	age | last_value
	-----+------------
	78 | 85
	90 | 85
	92 | 85
	85 | 85
	(4 rows)
	
	m_db=# DROP TABLE IF EXISTS t1;
	DROP TABLE
	– MySQL的行为：
	# 预置表数据。
	mysql> CREATE TABLE t1(id int,name varchar(20),age int);
	
	Query OK, 0 rows affected (0.12 sec)
	
	mysql> INSERT INTO t1(id, name,age) VALUES (1, 'zwt',90), (2, 'dda',85), (3, 'aab',90), (4, 'aac',78), (5, 'aad',85), (6, 'aae',92), (7, 'aaf',78);
	Query OK, 7 rows affected (0.01 sec)
	Records: 7 Duplicates: 0 Warnings: 0
	
	mysql> INSERT INTO t1(id, name,age) VALUES (1, 'zwt',90), (2, 'dda',85), (3, 'aab',90), (4, 'aac',78), (5, 'aad',85), (6, 'aae',92), (7, 'aaf',78);
	Query OK, 7 rows affected (0.01 sec)
	Records: 7 Duplicates: 0 Warnings: 0
	
	# 表数据的顺序与GaussDB有差异导致last_value的取值结果有差异。
	mysql> SELECT age, last_value(age) over() FROM t1 WHERE id > 0
	GROUP BY age HAVING age > 10;
	+------+------------------------+
	| age | last_value(age) over() |
	+------+------------------------+
	| 90 | 92 |
	| 85 | 92 |
	| 78 | 92 |
	| 92 | 92 |
	+------+------------------------+
	4 rows in set (0.00 sec)
	
	mysql> DROP TABLE IF EXISTS t1;
	Query OK, 0 rows affected (0.10 sec)

3.2.11 数字操作函数

表 3-21 数字操作函数列表

函数名	与MySQL的差异
ABS()	-
ACOS()	-
ASIN()	-

函数名	与MySQL的差异
ATAN()	-
ATAN2()	-
CEILING() CEIL() FLOOR()	部分场景下函数的返回类型与MySQL不一致，进而导致CREATE TABLE AS生成的表字段与MySQL不一致。 ● 入参为BIGINT类型或BIGINT UNSIGNED类型，入参值大于等于20位时（包括符号位），GaussDB返回整型，MySQL 5.7返回DECIMAL。例如： SET m_format_behavior_compat_options='enable_precision_decimal'; CREATE TABLE tt AS SELECT ceiling(-9223372036854775808); DESC tt; MySQL表字段返回类型为：DECIMAL(16,0)。 GaussDB表字段返回类型为：BIGINT(17)。 ● 入参为NUMERIC类型时，返回类型可能与MySQL不一致。当参数为常量、表字段时返回类型和MySQL 5.7保持一致。对于其他类型的入参如嵌套情况，结果会有差异，GaussDB返回NUMERIC类型，MySQL可能返回整型。例如： SET m_format_behavior_compat_options='enable_precision_decimal'; CREATE TABLE t AS SELECT ceiling(abs(5.5)); DESC t; MySQL表字段返回类型为：INT(4)。 GaussDB表字段返回类型为：DECIMAL(3,0)。
COS()	-
DEGREES()	-
EXP()	-
LN()	-
LOG()	-
LOG10()	-
LOG2()	-
PI()	精度传递开关关闭的情况下，也即m_format_behavior_compat_options中的enable_precision_decimal未设置时，PI函数的返回值精度与MySQL的有差异：MySQL中PI函数的结果仅保留四舍五入之后的小数后6位，而GaussDB的结果会保留四舍五入之后的小数后15位。
POW()	-
POWER()	-
RAND()	-
SIGN()	-
SIN()	-
SQRT()	-

函数名	与MySQL的差异
TAN()	-
TRUNCATE()	-
CRC32()	当BINARY类型插入字符串长度小于目标长度时，GaussDB填充符和MySQL不同；因此入参为BINARY类型时，函数结果和MySQL不一致。
CONV()	-
COT()	-
RADIANS()	-

3.2.12 网络地址函数

表 3-22 网络地址函数列表

函数名	与MySQL的差异
INET_ATON()	-
INET_NTOA()	-
INET6_ATON()	-
INET6_NTOA()	-
IS_IPV6()	-
IS_IPV4()	-

3.2.13 其他函数

表 3-23 其他函数列表

函数名	与MySQL的差异
DATABASE()	-
UUID()	-
UUID_SHORT()	-

函数名	与MySQL的差异

ANY_VALUE()	• 作为分组的第一条数据是不确定的，与底层算子相关。例如同一条sql语句，GaussDB返回5和4，MySQL返回5和2。
	CREATE TABLE t1(a INT, b INT);
	INSERT INTO t1 VALUES(1, 5);
	INSERT INTO t1 VALUES(2, 4);
	INSERT INTO t1 VALUES(2, 2);
	CREATE TABLE t2(a INT, b INT);
	INSERT INTO t2 VALUES(2, 7);
	INSERT INTO t2 VALUES(3, 9);
	m_db=# SELECT ANY_VALUE(t1.b) FROM t1 LEFT JOIN t2 ON t1.a=t1.b
	GROUP BY t1.a;
	any_value
	----------- 	
	5
	4
	(2 rows)
	mysql> SELECT ANY_VALUE(t1.b) FROM t1 LEFT JOIN t2 ON t1.a=t1.b
	GROUP BY t1.a;
	+-----------------+
	| ANY_VALUE(t1.b) |
	+-----------------+
	| 5 |
	| 2 |
	+-----------------+
	2 rows in set (0.04 sec)
	DROP TABLE t1;
	DROP TABLE t2;
	• 与DISTINCT关键字一起使用的情况下，ORDER BY中排序的列没有包括在SELECT语句所检索的结果集的列中时，GaussDB目前不支持使用ANY_VALUE函数避免报错。
	CREATE TABLE t1(a INT, b INT);
	INSERT INTO t1 VALUES(1, 2);
	INSERT INTO t1 VALUES(1, 3);
	m_db=# SELECT DISTINCT a FROM t1 ORDER BY ANY_VALUE(b);
	ERROR: For SELECT DISTINCT, ORDER BY expressions must appear in select list.
	LINE 1: SELECT DISTINCT a FROM t1 ORDER BY ANY_VALUE(b);
	^
	mysql> SELECT DISTINCT a FROM t1 ORDER BY ANY_VALUE(b);
	+------+
	| a |
	+------+
	| 1 |
	+------+
	1 row in set (0.00 sec)
	DROP TABLE t1;
	• ANY_VALUE函数入参为NULL、字符串类型时，返回值类型与MySQL存在差异。例如：
	– 入参为NULL时，GaussDB返回值类型为BIGINT，MySQL返回值类型为binary。
	– 入参为VARCHAR类型时，GaussDB返回值类型为VARCHAR类型，MySQL返回值类型可能是VARCHAR类型，也可能是TEXT类型。

函数名	与MySQL的差异
SLEEP()	• 调用SLEEP函数的过程中，若使用Ctrl+C提前结束调用，GaussDB只显示Cancel request sent信息，与MySQL显示不一致。 • 除上述情况外，当SLEEP函数在其他SQL语句中被调用时，使用Ctrl+C提前结束语句，若操作恰好被SLEEP函数内部获取，则不会报错，若被系统中其他函数获取则报错，与MySQL表现不一致。 • 若在SLEEP函数执行的过程中，其所在的进程被相关命令结束（如：SELECT PG_TERMINATE_BACKEND(xxx);），GaussDB目前行为为报错处理，与MySQL不一致。
COLLATION()	GaussDB仅支持utf8、utf8mb4、gbk、gb18030和latin1字符集下的字符序。
FOUND_ROWS()	-
ROW_COUNT()	• GaussDB没有SIGNAL语句，MySQL支持SIGNAL语句。 • GaussDB中，因为不存在连接参数CLIENT_FOUND_ROWS（设置之后返回匹配行数而不是影响行数），不受此参数的影响，统一返回影响行数，MySQL受此参数影响。 • GaussDB中INSERT ON DUPLICATE KEY UPDATE中触发冲突的场景，返回受影响行数和MySQL存在差异，具体请参见**DML**章节中INSERT ... ON DUPLICATE KEY UPDATE语法的差异说明。
SYSTEM_USER()	MySQL的配置文件中配置skip-name-resolve时，不会将127.0.0.1或::1解析为localhost，GaussDB没有相关参数，始终将127.0.0.1和::1解析为localhost。
DEFAULT()	GaussDB支持使用列别名，MySQL不支持。
BENCHMARK()	• 因为MySQL和GaussDB执行层框架存在差异，所以MySQL和GaussDB使该函数评估同一个表达式的执行时间不具有可比性。该函数仅用于评估GaussDB不同表达式的执行效率对比。 • 执行时间较长时，当在客户端输入Ctrl+C时，MySQL返回0后结束任务，GaussDB统一显示Cancel request sent后结束任务。
LAST_INSERT_ID()	-
CONNECTION_ID()	-

3.3 操作符

GaussDB数据库兼容绝大多数MySQL的操作符，但存在部分差异。如未列出，操作符行为默认为GaussDB原生行为，目前存在MySQL不支持但是GaussDB支持的语句，在GaussDB数据库下，这类语句通常为系统内部使用，因此不建议使用。

操作符差异

- ORDER BY排序对NULL值处理的差异。MySQL在排序时会将NULL值排序在前面；GaussDB默认将NULL值默认排在最后面。GaussDB可以通过NULLS FIRST和NULLS LAST设置NULL值排序顺序。
- 有ORDER BY时，除上述特殊场景外，GaussDB输出顺序与MySQL一致。没有ORDER BY时，GaussDB不保证结果顺序与MySQL一致。
- 使用MySQL操作符时需要用括号表达式严格确保表达式的结合性，否则执行报错。例如：SELECT 1 regexp ('12345' regexp '123');。

  GaussDB数据库中操作符不用括号严格表达的结合性也能成功执行。
- NULL值显示不同。MySQL会将NULL显示为“NULL”；GaussDB将NULL值显示为空值。

  MySQL输出结果：

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

  GaussDB输出结果：

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

  (1 row)
  ```
- 操作符执行后，列名显示不一致。MySQL会将NULL显示为“NULL”；GaussDB将NULL值显示为空值。
- 字符串转double遇到非法字符串时，告警信息不一致。MySQL在常量非法字符串报错，字段非法字符串不报错；GaussDB在常量非法字符串和字段非法字符串都报错。
- 比较操作符返回结果显示不同。MySQL返回1/0；GaussDB返回t/f。

表 3-24 操作符

操作符	与MySQL的差异
<>	MySQL支持索引，GaussDB不支持索引。
<=>	MySQL支持索引，GaussDB不支持索引、hash连接和合并连接。

操作符	与MySQL的差异

行表达式	• MySQL支持<=>操作符行比较、GaussDB不支持<=>操作符行比较。
	• MySQL不支持行表达式与NULL比较。GaussDB支持<、<=、=、>=、>、<>操作符对行表达式与NULL值比较。
	• MySQL不支持IS NULL、ISNULL对行表达式的操作。GaussDB支持。
	• 操作符对于行表达式的不支持的操作，GaussDB错误信息与MySQL不一致。
	• MySQL不支持ROW(values)其中values只有一列数据，GaussDB支持。
	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
	m_db=# SELECT ROW(NULL) AS x;
	x
	---- 	
	()
	(1 row)
	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> SELECT ISNULL((1, 2));
	ERROR 1241 (21000): Operand should contain 1 column(s)
	mysql> SELECT NULL BETWEEN NULL AND ROW(2,2);

操作符	与MySQL的差异
	ERROR 1241 (21000): Operand should contain 1 column(s) mysql> SELECT ROW(NULL) AS x; 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 ') as x' at line 1
--	MySQL表示对一个操作数进行两次取反，结果等于原操作数；GaussDB表示注释。
!!	MySQL：!!含义同!，表示取非。 GaussDB：!表示取非操作，当!与!中间存在空格时，表示连续两次取非（! !）；当!与!中间没有空格时，表示阶乘（!!）。 **说明** • GaussDB中，当同时使用阶乘（!!）和取非（!）时，阶乘（!!）和取非（!）中间需要添加空格，否则会报错。 • GaussDB中，当需要多次取非操作时，!与!之间需使用空格隔开。
[NOT] REGEXP	• GaussDB和MySQL在正则表达式中支持的元字符有所不同。例如，GaussDB支持“\d”表示数字，“\w”表示字母、数字和下划线，“\s”表示空格，而MySQL不支持这些元字符，MySQL会把这些字符当成正常字符串。 • GaussDB中“\b”可以与“\\b”匹配，MySQL会匹配失败。 • GaussDB中使用“\”表示转义字符，而MySQL中使用“\\”。 • MySQL不支持2个操作符连在一起使用。 • 模式字符串pattern非法入参，只存在右单括号“)”时，GaussDB会报错，MySQL 5.7会报错，MySQL 8.0不会报错。 • 在de|abc匹配序列de或abc的匹配规则，当|左右存在空值时，GaussDB不会报错，MySQL 5.7会报错，MySQL 8.0不会报错。 • 制表符“\t”正则匹配字符类[:blank:]，GaussDB可匹配，MySQL 5.7不能匹配，MySQL 8.0可匹配。 • GaussDB支持非贪婪模式匹配，即尽可能少的匹配字符，在部分特殊字符后加“?”问号字符，例如：“??, *?, +?, {n}?, {n,}?, {n,m}?”。MySQL 5.7版本不支持非贪婪模式匹配，并报错：Got error 'repetition-operator operand invalid' from regexp。MySQL 8.0版本已经支持。 • 在BINARY字符集下，text类型、blob类型均会转换成bytea类型，由于REGEXP操作符不支持bytea类型，因此无法匹配。
LIKE	MySQL：LIKE的左操作数只能是位运算或者算术运算或者由括号组成的表达式，LIKE的右操作数只能是单目运算符(不含NOT)或者括号组成的表达式。 GaussDB：LIKE的左右操作数可以是任意表达式。

操作符	与MySQL的差异
[NOT] BETWEEN AND	MySQL：[NOT] BETWEEN AND嵌套使用时从右到左结合。[NOT] BETWEEN AND的第1个操作数和第2个操作数只能是位运算或者算术运算或者由括号组成的表达式。 GaussDB：[NOT] BETWEEN AND嵌套使用时从左到右结合。[NOT] BETWEEN AND的第1个操作数和第2个操作数可以是任意表达式。

IN	MySQL：IN的左操作数只能是位运算或者算术运算或者由括号组成的表达式。
	GaussDB：IN的左操作数可以是任意表达式。不支持ROW IN (ROW,ROW....)形式的查询。
	在开启精度传递的场景下，对表中的数据使用in操作符时，若表中数据为FLOAT类型、DOUBLE类型且包括相应精度和标度（如float(4,2)，double(4,2)）,GaussDB会根据精度和标度对值进行比较，MySQL会读取内存值（失真数值，导致比较不相等）。
	--GaussDB：
	m_db=# CREATE TABLE test1(t_float float(4,2));
	CREATE TABLE
	m_db=# INSERT INTO test1 VALUES(1.42),(2.42);
	INSERT 0 2
	m_db=# SELECT t_float, t_float in (1.42,2.42) FROM test1;
	t_float | ?column?
	---------+----------
	1.42 | t
	2.42 | t
	(2 rows)
	--MySQL：
	mysql> CREATE TABLE test1(t_float float(4,2));
	Query OK, 0 rows affected (0.01 sec)
	mysql> INSERT INTO test1 VALUES(1.42),(2.42);
	Query OK, 2 rows affected (0.00 sec)
	Records: 2 Duplicates: 0 Warnings: 0
	mysql> SELECT t_float, t_float in (1.42,2.42) FROM test1;
	+---------+------------------------+
	| t_float | t_float in (1.42,2.42) |
	+---------+------------------------+
	| 1.42 | 0 |
	| 2.42 | 0 |
	+---------+------------------------+
	2 rows in set (0.00 sec)

!	MySQL：!的操作数只能是单目运算符（不含not）或者括号组成的表达式。 GaussDB：!的操作数可以是任意表达式。
#	MySQL支持#注释，GaussDB不支持#注释。
BINARY	GaussDB中支持的表达式和MySQL并不完全一致（包括一些函数、操作符等）。GaussDB独有的表达式“~”、“IS DISTINCT FROM”等，由于BINARY关键字优先级更高，使用BINARY expr会优先将BINARY与“~”、“IS DISTINCT FROM”的左参数合并，导致报错。

操作符	与MySQL的差异

取负（-）	取反结果类型和精度与MySQL不一致：
	`CREATE TABLE t AS SELECT - -1;`
	● MySQL表字段返回类型为：decimal(2,0)。
	● GaussDB表字段返回类型为：integer(1)。
	在开启精度传递（m_format_behavior_compat_options = 'enable_precision_decimal'）的情况下，负号加常量数据类型的精度可能与MySQL存在差异，MySQL 5.7中，当表达式中包含取反操作符时，结果精度中的最大长度（max_length）会根据表达式中取反操作符的数量增加，而GaussDB不会。例如：
	● GaussDB：
	`m_db=# DROP TABLE IF EXISTS test;`
	`NOTICE:  table "test" does not exist, skipping`
	`DROP TABLE`
	`m_db=# CREATE TABLE test as`
	`m_db-# SELECT format(-4.4600e3,1) f9;`
	`INSERT 0 1`
	`m_db=# DESC test;`
	`Field |  Type  | Null | Key | Default | Extra`
	`-------+-------------+------+-----+---------+-------`
	`f9  | varchar(45) | YES |  |  |`
	`(1 row)`
	
	`m_db=# DROP TABLE IF EXISTS t1;`
	`NOTICE:  table "t1" does not exist, skipping`
	`DROP TABLE`
	`m_db=# CREATE TABLE t1 AS SELECT cast(- -4.46 AS BINARY) c4,convert(- - -002.2600,BINARY) c14;`
	`INSERT 0 1`
	`m_db=# DESC t1;`
	`Field |  Type  | Null | Key | Default | Extra`
	`-------+---------------+------+-----+---------+-------`
	`c4  | varbinary(5)  | YES |  |  |`
	`c14  | varbinary(10) | YES |  |  |`
	`(2 rows)`
	
	`m_db=# DROP VIEW IF EXISTS v2;`
	`NOTICE:  view "v2" does not exist, skipping`
	`DROP VIEW`
	`m_db=# CREATE VIEW v2 AS SELECT cast(- -4.46 AS BINARY) c4,convert(- - -002.2600,BINARY) c14;`
	`CREATE VIEW`
	`m_db=# DESC v2;`
	`Field |  Type  | Null | Key | Default | Extra`
	`-------+--------------+------+-----+---------+-------`
	`c4  | varbinary(5) | YES |  |  |`
	`c14  | varbinary(8) | YES |  |  |`
	`(2 rows)`
	● MySQL：
	`mysql> DROP TABLE IF EXISTS test;`
	`Query OK, 0 rows affected (0.01 sec)`
	
	`mysql> CREATE TABLE test AS`
	`  -> SELECT format(-4.4600e3,1) f9;`
	`Query OK, 1 row affected (0.01 sec)`
	`Records: 1  Duplicates: 0  Warnings: 0`
	
	`mysql> DESC test;`
	`+-------+-------------+------+-----+---------+-------+`
	`| Field | Type  | Null | Key | Default | Extra |`
	`+-------+-------------+------+-----+---------+-------+`
	`| f9  | varchar(63) | YES |  | NULL  |  |`

操作符	与MySQL的差异
	+-------+-------------+------+-----+---------+-------+ 1 row in set (0.00 sec) mysql> DROP TABLE IF EXISTS t1; Query OK, 0 rows affected, 1 warning (0.00 sec) mysql> CREATE TABLE t1 AS SELECT cast(- -4.46 AS BINARY) c4,convert(- - -002.2600,BINARY) c14; Query OK, 1 row affected (0.02 sec) Records: 1 Duplicates: 0 Warnings: 0 mysql> DESC t1; +-------+---------------+------+-----+---------+-------+ | Field | Type | Null | Key | Default | Extra | +-------+---------------+------+-----+---------+-------+ | c4 | varbinary(7) | YES | | NULL | | | c14 | varbinary(12) | YES | | NULL | | +-------+---------------+------+-----+---------+-------+ 2 rows in set (0.00 sec) mysql> DROP VIEW IF EXISTS v2; Query OK, 0 rows affected, 1 warning (0.00 sec) mysql> CREATE VIEW v2 AS SELECT cast(- -4.46 AS BINARY) c4,convert(- - -002.2600,BINARY) c14; Query OK, 0 rows affected (0.03 sec) mysql> DESC v2; +-------+---------------+------+-----+---------+-------+ | Field | Type | Null | Key | Default | Extra | +-------+---------------+------+-----+---------+-------+ | c4 | varbinary(7) | YES | | NULL | | | c14 | varbinary(10) | YES | | NULL | | +-------+---------------+------+-----+---------+-------+ 2 rows in set (0.00 sec)
/**/	GaussDB中语句中不支持/**/注释。

操作符	与MySQL的差异

xor	GaussDB的xor和MySQL的行为不同。GaussDB优化器会有常数优化，会出现先计算表示结果是常数的情况。
	GaussDB:
	m_db=# SELECT 1 xor null xor pow(200, 2000000) FROM dual;
	ERROR: value out of range: overflow
	m_db=# CREATE TABLE t1(a int, b int);
	CREATE TABLE
	m_db=# INSERT INTO t1 VALUES(2,2), (200, 2000000000);
	INSERT 0 2
	m_db=#
	m_db=#
	m_db=# SELECT 1 xor null xor pow(a, b) FROM t1;
	?column?
	---------- 	
	
	(2 rows)
	MySQL:
	mysql> SELECT 1 xor null xor pow(200, 2000000) FROM dual;
	+----------------------------------+
	| 1 xor null xor pow(200, 2000000) |
	+----------------------------------+
	| NULL |
	+----------------------------------+
	1 row in set (0.00 sec)
	ysql> CREATE TABLE t1(a int, b int);
	Query OK, 0 rows affected (0.04 sec)
	
	mysql> INSERT INTO t1 VALUES(2,2), (200, 2000000000);
	Query OK, 2 rows affected (0.01 sec)
	Records: 2 Duplicates: 0 Warnings: 0
	
	mysql> SELECT 1 xor null xor pow(a, b) FROM t1;
	+--------------------------+
	| 1 xor null xor pow(a, b) |
	+--------------------------+
	| NULL |
	| NULL |
	+--------------------------+
	2 rows in set (0.00 sec)

IS NULL和IS NOT NULL	MySQL的优先级小于逻辑运算符，GaussDB的优先级大于逻辑运算符。

操作符	与MySQL的差异

AND（&&）、OR（||）、XOR、|、&、<、>、<=、>=、=、!=	MySQL执行机制为执行左操作数后，对结果进行判断是否为空，进而决定是否需要执行右操作数。
	GaussDB执行机制是执行左右操作数后，对结果再进行判断是否为空。
	当左操作数结果为空，右操作数执行报错时，MySQL不会报错直接返回，GaussDB会执行报错。
	MySQL行为：
	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)
	GaussDB行为：
	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) | (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)

操作符	与MySQL的差异

+、-、*、 /、%、mod、div	CREATE VIEW AS SELECT算数操作符（'+', '-', '*', '/', '%', 'mod', 'div'）内嵌b''常量时，MySQL 5.7返回类型可能存在unsigned标识，GaussDB不会。
	MySQL输出结果：
	mysql> CREATE VIEW v22 as SELECT b'101' / b'101' c22;
	Query OK, 0 rows affected (0.00 sec)
	
	mysql> DESC v22;
	+-------+------------------------+------+-----+---------+-------+
	| Field | Type | Null | Key | Default | Extra |
	+-------+------------------------+------+-----+---------+-------+
	| c22 | decimal(5,4) unsigned | YES | | NULL | |
	+-------+------------------------+------+-----+---------+-------+
	1 row in set (0.01 sec)
	GaussDB输出结果：
	m_db=# CREATE VIEW v22 AS SELECT b'101' / b'101' c22;
	CREATE VIEW
	m_db=# DESC v22;
	Field | Type | Null | Key | Default | Extra
	-------+--------------+------+-----+---------+-------
	c22 | decimal(5,4) | YES | | |
	(1 row)

表 3-25 操作符组合差异

操作符组合示例	MySQL数据库	GaussDB数据库	说明
SELECT 1 LIKE 3 & 1;	不支持	支持	LIKE的右操作数不能是位运算符组成的表达式。
SELECT 1 LIKE 1 +1;	不支持	支持	LIKE的右操作数不能是算术运算符组成的表达式。
SELECT 1 LIKE NOT 0;	不支持	支持	LIKE的右操作数只能是+、-、!等单目操作符或者括号组成的表达式，NOT除外。
SELECT 1 BETWEEN 1 AND 2 BETWEEN 2 AND 3;	从右到左结合	从左到右结合	建议加上括号明确运算的优先级，防止由于运算顺序的差异导致运算结果产生偏差。
SELECT 2 BETWEEN 1=1 AND 3;	不支持	支持	BETWEEN的第2个操作数不能是比较操作符组成的表达式。
SELECT 0 LIKE 0 BETWEEN 1 AND 2;	不支持	支持	BETWEEN的第1个操作数不能是模式匹配操作符组成的表达式。
SELECT 1 IN (1) BETWEEN 0 AND 3;	不支持	支持	BETWEEN的第1个操作数不能是IN操作符组成的表达式。
SELECT 1 IN (1) IN (1);	不支持	支持	第2个IN表达式左操作数不能是IN组成的表达式。

操作符组合示例	MySQL数据库	GaussDB数据库	说明
SELECT ! NOT 1;	不支持	支持	!的操作数只能是+、-、!等单目操作符或者括号组成的表达式，NOT除外。

索引差异

- GaussDB当前仅支持UBTree和B-tree索引。
- 在执行LIKE模糊匹配，并通过EXPLAIN打印执行计划时，执行计划中会显示当前索引列对应字符序的最小/最大字符权重编码。该编码在不同的客户端字符集设置下的显示可能存在差异，但不影响LIKE模糊匹配查询结果的正确性。
- B-tree/UBTree索引场景保持原生GaussDB原有逻辑，即同一操作符族内的类型比较，支持索引扫描，其余索引类型暂未支持。
- 在使用GaussDB JDBC连接数据库时，GaussDB的YEAR类型在含有绑定参数的PBE场景下无法利用索引。
- WHERE子句中，索引字段类型和常量类型操作场景下，GaussDB中索引与MySQL索引支持存在差异，如表3-26所示。例如以下语句GaussDB不支持索引：

```
CREATE TABLE t(_int int);
CREATE INDEX idx ON t(_int) USING BTREE;
SELECT * FROM t WHERE _int >  2.0;
```

说明

- WHERE子句里索引字段类型和常量类型操作场景中，可以使用cast函数将常数类型显式转换为字段类型，以便实现索引。

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

- 在执行LIKE模糊匹配时，单字节字符集场景下，GaussDB创建前缀索引长度最长为2676字节（MySQL为3072字节），超过最大长度建议创建非前缀索引。

表 3-26 索引支持差异

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
BIT类型	BIT类型	否	是
	整型	否	是
	浮点型	否	是
	字符串类型	否	是
	二进制类型	否	是
	带日期的时间类型	否	否
	TIME类型	否	是
	YEAR类型	否	是

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
SET/ENUM类型	字符串类型	否	否
	二进制类型	否	否
	整型	否	否
	浮点型	否	否
	定点型	否	否
	带日期的时间类型	否	否
	TIME类型	否	否
	YEAR类型	否	否
TIME类型	TIME类型	是	是
	带日期的时间类型	是	是
	YEAR类型	是	是
	整型（可转为TIME类型）	是	是
	整型（不可转为TIME类型）	否	否
	字符串类型（可转为TIME类型）	是	是
	字符串类型（不可转为TIME类型）	否	否
	二进制类型（可转为TIME类型）	是	是
	二进制类型（不可转为TIME类型）	否	否
	浮点型（可转为TIME类型）	是	是
	浮点型（不可转为TIME类型）	否	否
	定点型（可转为TIME类型）	是	是

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
	定点型（不可转为TIME类型）	否	否
YEAR类型	YEAR类型	是	是
	整型（可转为YEAR类型）	是	是
	整型（不可转为YEAR类型，负数）	否	是
	浮点型（可转为YEAR类型）	是	是
	浮点型（不可转为YEAR类型）	否	否
	定点型（可转为YEAR类型）	是	是
	定点型（不可转为YEAR类型）	否	否
	字符串类型（可转为YEAR类型）	是	是
	字符串类型（不可转为YEAR类型）	否	否
	二进制类型（可转为YEAR类型）	是	是
	二进制类型（不可转为YEAR类型）	否	否
	带日期的时间类型	否	是
	TIME类型	否	是
带日期的时间类型	带日期的时间类型	是	是
	TIME类型	是	是
	YEAR类型	是	否

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
	整型（可转为带日期的时间类型）	是	是
	整型（不可转为带日期的时间类型）	否	否
	浮点型（可转为带日期的时间类型）	是	是
	浮点型（不可转为带日期的时间类型）	否	否
	定点型（可转为带日期的时间类型）	是	是
	定点型（不可转为带日期的时间类型）	否	否
	字符串类型（可转为带日期的时间类型）	是	是
	字符串类型（不可转为带日期的时间类型）	否	否
	二进制类型（可转为带日期的时间类型）	是	是
	二进制类型（不可转为带日期的时间类型）	否	否
定点型	定点型	是	是
	整型	是	是
	浮点型	否	是
	字符串类型	否	是
	二进制类型	否	是

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
	带日期的时间类型	否	是
	TIME类型	否	是
	YEAR类型	是	是
浮点型	浮点型	是	是
	整型	是	是
	定点型	是	是
	字符串类型	是	是
	二进制类型	是	是
	带日期的时间类型	是	是
	TIME类型	是	是
	YEAR类型	是	是
整型	整型	是	是
	浮点型	否	是
	定点型	是	是
	字符串类型(可转为整型）	是	是
	字符串类型(不可转为整型）	否	是
	二进制类型(可转为整型）	是	是
	二进制类型(不可转为整型）	否	是
	带日期的时间类型	是	是
	TIME类型	是	是
	YEAR类型	是	是
二进制类型	二进制类型	是	是
	字符串类型	是	是

索引字段类型	常量类型	GaussDB是否支持走索引	MySQL是否支持走索引
	带日期的时间类型	否	否
	TIME类型	是	否
	YEAR类型	否	否
	整型	否	否
	浮点型	否	否
	定点型	否	否
字符串类型	字符串类型	是	是
	带日期的时间类型	否	否
	TIME类型	是	否
	YEAR类型	否	否
	二进制类型	否	否
	整型	否	否
	浮点型	否	否
	定点型	否	否

说明

- 仅在GUC参数m_format_behavior_compat_options未开启disable_int_cmp_num_index选项时，支持整型索引字段与定点型常量比较走索引。
- 整型索引字段与定点型常量比较走索引场景中，不支持定点型常量为用户自定义变量、case when表达式或子查询返回值。

3.4 字符集

GaussDB数据库支持指定数据库、模式、表或列的字符集，默认的字符集是utf8。支持的范围如下。

表 3-27 字符集列表

MySQL数据库	GaussDB数据库
utf8mb4	支持
utf8	支持
gbk	支持

MySQL数据库	GaussDB数据库
gb18030	支持
binary	支持
latin1	支持

说明

- GaussDB数据库将utf8和utf8mb4视为同一个字符集，编码最大长度为4字节。当字符串字符集为utf8，指定其字符序为utf8mb4_bin/utf8mb4_general_ci/utf8mb4_unicode_ci/utf8mb4_0900_ai_ci时（例如SELECT _utf8'a' collate utf8mb4_bin），MySQL中会发生报错，GaussDB不报错。当字符串字符集为utf8mb4，指定其字符序为utf8_bin/utf8_general_ci/utf8_unicode_ci时，存在同样差异。
- 词法语法解析按照字节流解析，当多字节字符中包含与'\', '\'', '\\'等符号一致的编码时，会导致与MySQL行为不一致，建议暂时关闭转义符开关（请参见《管理员指南》中“配置运行参数 > GUC参数说明 > 版本和平台兼容性 > 平台和客户端兼容性”章节中GUC参数m_format_behavior_compat_options的enable_escape_string选项）进行规避。
- GaussDB数据库对不属于当前字符集的非法字符未执行严格的编码逻辑校验，可能导致此类非法字符成功输入，而MySQL会校验报错。

3.5 排序规则

GaussDB数据库支持指定模式、表或列的排序规则，支持的范围如下。

说明

排序规则差异说明：

- GaussDB数据库仅字符串类型、部分二进制类型支持指定排序规则，其他类型不支持指定排序规则，可以通过查询pg_type系统表中类型的typcollation属性是否为0来判断该类型是否支持字符序。MySQL中所有类型均可以指定字符序，但除字符串、二进制类型外，其他类型指定排序规则无实际意义。
- GaussDB数据库中排序规则（除binary外）仅支持在其对应字符集与库级字符集一致时可以指定，字符集必须与数据库的字符集一致，且不支持表内多种字符集混合使用。
- utf8mb4字符集下默认字符序为utf8mb4_general_ci，与MySQL 5.7保持一致。
- 使用latin1字符序需要设置兼容性参数m_format_dev_version = 's2'。

表 3-28 排序规则列表

MySQL数据库	GaussDB数据库
utf8mb4_general_ci	支持
utf8mb4_unicode_ci	支持
utf8mb4_bin	支持
gbk_chinese_ci	支持
gbk_bin	支持
gb18030_chinese_ci	支持

MySQL数据库	GaussDB数据库
gb18030_bin	支持
binary	支持
utf8mb4_0900_ai_ci	支持
utf8_general_ci	支持
utf8_bin	支持
utf8_unicode_ci	支持
latin1_swedish_ci	支持
latin1_bin	支持

3.6 事务

GaussDB数据库兼容MySQL的事务，但存在部分差异。本章节介绍GaussDB数据库中事务相关的差异。

事务默认隔离级别

GaussDB数据库默认隔离级别为READ COMMITTED，MySQL默认隔离级别为REPEATABLE-READ。

```
-- 查看当前事务隔离级别。
m_db=# SHOW transaction_isolation;
 transaction_isolation
-----------------------
 read committed
(1 row)
```

子事务

GaussDB数据库中，可使用SAVEPOINT用于在当前事务里建立一个新的保存点（子事务），使用ROLLBACK TO SAVEPOINT回滚到指定保存点（子事务），子事务回滚后父事务可以继续运行，子事务的回滚不影响父事务的事务状态。

MySQL不存在创建保存点（子事务）。

嵌套事务

嵌套事务指在事务块中开启新事务。

GaussDB数据库中，正常事务块中开启新事务会警告存在一个进行中的事务，忽略开启命令；异常事务块中开启新事务将报错，必须在执行ROLLBACK/COMMIT回滚之前的语句后才可以执行。

MySQL中，正常事务块中开启新事务会先把之前事务提交，然后开启新事务；异常事务块中开启新事务会忽略错误，提交之前无错误的语句并开启新事务。

```
-- GaussDB数据库正常事务块中，开启新事务会警告并忽略。
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;   -- 会警告there is already a transaction in progress。
m_db=# SELECT * FROM test_t ORDER BY 1;
m_db=# COMMIT;

-- GaussDB数据库异常事务块中，开启新事务会报错，必须ROLLBACK/COMMIT之后才可以执行。
m_db=# BEGIN;
m_db=# ERROR sql; -- 错误语句。
m_db=# BEGIN; -- 报错。
m_db=# COMMIT; -- ROLLBACK/COMMIT之后才可以执行。
```

隐式提交的语句

GaussDB数据库使用GaussDB存储，继承GaussDB事务机制，事务中执行DDL、DCL不会自动提交。

MySQL在DDL、DCL、管理类语句，锁相关语句会自动提交。

```
-- GaussDB数据库创建表和设置GUC参数可以回滚操作。
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; -- 不存在该表。
Did not find any relation named "test_table_rollback".
```

SET TRANSACTION 差异

GaussDB数据库中，SET TRANSACTION同时设置多次隔离级别/事务访问模式时，只有最后一个会生效；多个事务特性支持使用空格和逗号分隔。

MySQL中SET TRANSACTION不允许设置多次隔离级别/事务访问模式；多个事务特性只支持使用逗号分隔。

表 3-29 SET TRANSACTION 差异

语法	功能	差异
SET TRANSACTION	设置事务特性。	GaussDB数据库中，不开启m_format_dev_version='s2'参数时，SET TRANSACTION在会话级别生效，功能与SET SESSION TRANSACTION一致；开启m_format_dev_version='s2'参数时，SET TRANSACTION是设置下一个事务特性；MySQL中SET TRANSACTION在下一个事务生效。
SET SESSION TRANSACTION	设置会话级事务特性。	-

语法	功能	差异
SET GLOBAL TRANSACTION	设置全局会话级事务特性，该特性适用于后续会话，对当前会话无影响。	GaussDB数据库中，GLOBAL是全局会话级别生效，只针对当前数据库实例，其它数据库不影响。 MySQL中，会使所有数据库生效。

```
-- SET TRANSACTION会话级生效。
m_db=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
m_db=# SHOW transaction_isolation;
m_db=# SHOW transaction_read_only;
-- GaussDB数据库同时设置多次隔离级别/事务访问模式，最后一个生效。
m_db=# SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED, ISOLATION LEVEL
REPEATABLE READ, READ WRITE, READ ONLY;
m_db=# SHOW transaction_isolation; -- repeatable read
 transaction_isolation
-----------------------
 repeatable read
(1 row)
m_db=# SHOW transaction_read_only; -- on
 transaction_read_only
-----------------------
 on
(1 row)
```

START TRANSACTION 差异

GaussDB数据库中，START TRANSACTION开启事务时，同时支持设置隔离级别；同时设置多次隔离级别/事务访问模式时，只有最后一个会生效；当前版本不支持立即开启一致性快照；多个事务特性支持空格和逗号分隔。

MySQL的START TRANSACTION开启事务时，不支持设置隔离级别，不支持设置多次事务访问模式；多个事务特性只支持逗号分隔。

在MySQL中，可重复读隔离级别下的事务，只有在执行第一个SELECT语句后才开始快照读。在GaussDB中，事务一旦开启，不仅第一个SELECT语句会进行快照读，第一个执行的DDL、DML或DCL语句也会建立事务的一致性读快照。

```
-- 开启事务设置隔离级别。
m_db=# START TRANSACTION ISOLATION LEVEL READ COMMITTED;
m_db=# COMMIT;
-- 多次设置访问模式。
m_db=# START TRANSACTION READ ONLY, READ WRITE;
m_db=# COMMIT;
```

事务相关的 GUC 参数

表 3-30 事务相关的 GUC 参数差异

GUC参数	功能	差异
autocommit	设置事务自动提交模式。	-

GUC参数	功能	差异
transaction_isolation	在GaussDB中是设置当前事务的隔离级别。 在MySQL中是设置会话级事务的隔离级别。	• GaussDB中，通过使用SET transaction_isolation = value命令，只能改变当前事务的隔离级别。如果想要改变会话级的隔离级别，可以使用default_transaction_isolation。在MySQL中，通过使用SET命令，可以改变会话级的事务隔离级别。 • 支持范围差异。 MySQL中，支持以下隔离级别设置，对大小写不敏感，对空格敏感： – READ-COMMITTED – READ-UNCOMMITTED – REPEATABLE-READ – SERIALIZABLE GaussDB中，支持以下隔离级别设置，对大小写和空格敏感： – read committed – read uncommitted – repeatable read – serializable – default（设置和会话中默认隔离级别一样） • 在GaussDB中，新事务的transaction_isolation值将被初始化为default_transaction_isolation的值。 • 设置m_format_dev_version = 's2'时： – 以下语句是设置下一个事务特性：set @@transaction_isolation = value; set transaction isolation level value。 – 以下语句修改会话级事务特性：set [local|session|@@session.] transaction_isolation = value。 – 下一个事务特性不允许在事务内使用，如果隐式事务报错，即单个SQL语句报错，继续保持下一个事务特性。
tx_isolation	设置事务的隔离级别； tx_isolation和transaction_isolation是同义词。	GaussDB中只支持通过select @@语法查询，不支持通过show语法查询，不支持修改。
default_transaction_isolation	设置事务的隔离级别。	GaussDB中通过SET设置会改变会话级事务隔离级别。 MySQL中不支持该系统参数。

GUC参数	功能	差异
transaction_read_only	在GaussDB中是设置当前事务的访问模式。 在MySQL中是设置会话级事务的访问模式。	• 在GaussDB中，通过使用SET命令，只能改变当前事务的访问模式。如果想要改变会话级的访问模式，可以使用default_transaction_read_only。在MySQL中，通过使用SET命令，可以改变会话级的事务隔离级别。 • 在GaussDB中，新事务的transaction_read_only值将被初始化为default_transaction_read_only的值。 • 设置m_format_dev_version = 's2'时： – 以下语句是设置下一个事务特性：set @@transaction_read_only = value; set transaction {read write | read only}。 – 以下语句修改会话级事务特性：set [local| session|@@session.] transaction_read_only = value。 – 下一个事务特性不允许在事务内使用，如果隐式事务报错，即单个SQL语句报错，继续保持下一个事务特性。
tx_read_only	设置事务的访问模式。tx_read_only和transaction_read_only是同义词。	GaussDB中只支持通过select @@语法查询，不支持通过show语法查询，不支持修改。
default_transaction_read_only	设置事务的访问模式。	GaussDB中通过SET设置会改变会话级事务访问模式；MySQL中不支持该系统参数。

3.7 SQL

3.7.1 关键字

约束差异：

- 当关键字在GaussDB数据库中为保留关键字，在MySQL中为非保留关键字，其差异为：在GaussDB数据库中不可作为表名、列名、列别名、AS列别名、AS表别名、表别名、函数名和变量名，在MySQL中支持。
- 当关键字在GaussDB数据库中为非保留关键字，在MySQL中为保留关键字，其差异为：在GaussDB数据库中可作为表名、列名、列别名、AS列别名、AS表别名、表别名、函数名和变量名，在MySQL中不支持。
- 当关键字在GaussDB数据库中为保留关键字（可以是函数或类型），在MySQL中为保留关键字，其差异为：在GaussDB数据库中可作为列别名、AS列别名、函数名和变量名，在MySQL中不支持。

- 当关键字在GaussDB数据库中为保留关键字（可以是函数或类型），在MySQL中为非保留关键字，其差异为：在GaussDB数据库中不可作为表名、列名、AS表别名和表别名，在MySQL中支持。
- 当关键字在GaussDB数据库中为非保留关键字（不能是函数或类型），在MySQL中为保留关键字，其差异为：在GaussDB数据库中可作为表名、列名、列别名、AS列别名、AS表别名、表别名和变量名，在MySQL中不支持。
- 当关键字在GaussDB数据库中为非保留关键字（不能是函数或类型），在MySQL中为非保留关键字，其差异为：在GaussDB数据库中不可作为函数名，在MySQL中支持。

  说明

  在GaussDB数据库中的非保留关键字、保留关键字（可以是函数或类型）以及非保留关键字（不能是函数或类型）之中，以下关键字不能作为列别名进行使用：

  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、WITHOUT

  其中，SIGNED和WITHOUT在MySQL中可以作为列别名进行使用。

3.7.2 标识符

GaussDB中的标识符存在以下差异：

- GaussDB无引号标识符中不支持以美元符号（$）开头，MySQL无引号标识符中支持。
- GaussDB无引号标识符中的支持大小写敏感的数据库对象。
- GaussDB标识符支持U+0080~U+00FF扩展字符，MySQL标识符支持U+0080~U+FFFF的扩展字符。
- 无引号标识符中，GaussDB不支持创建以数字开头包含一个e或E结尾作为标识符的表，例如：

```
-- GaussDB报错不支持，MySQL支持
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);
                        ^
```

- 有引号标识符中，GaussDB对于创建了列名为纯数字或科学计算法的表，不支持直接使用，需要在引号中使用；对于点操作符（.）场景，列名为纯数字或科学计算法的表也需要在引号中使用。例如：

```
-- 创建列名为纯数字或科学计算法的表
m_db=# CREATE TABLE t1(`123` int, `1e3` int, `1e` int);
CREATE TABLE

-- 向表中插入数据
m_db=# INSERT INTO t1 VALUES(7, 8, 9);
INSERT 0 1

-- 结果非预期，但与MySQL结果一致
m_db=# SELECT 123 FROM t1;
 ?column?
----------
```



```
    123
(1 row)

-- 结果非预期，但与MySQL结果一致
m_db=# SELECT 1e3 FROM t1;
 ?column?
----------
    1000
(1 row)

-- 结果非预期，并且与MySQL结果不一致
m_db=# SELECT 1e FROM t1;
 e
---
 1
(1 row)

-- 正确用法
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)

-- 点操作符的场景，GaussDB不支持，MySQL支持
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;
                 ^
-- 点操作符的场景，正确用法：
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
```

- GaussDB分区名使用双引号（需要设置SQL_MODE包含ANSI_QUOTES）或反引号时区分大小写，MySQL不区分。
- MySQL标识符长度限制为64字符，而GaussDB标识符长度限制为63字节。超过标识符的长度限制后，MySql报错，GaussDB会对标识符截断并告警。
- GaussDB不支持可执行注释。

3.7.3 DDL

表 3-31 DDL 语法兼容介绍

概述	详细语法说明	差异
主键、索引	ALTER TABLE、CREATE TABLE、CREATE INDEX 、DROP INDEX	• 在GaussDB中，当约束关联的表为ustore，且SQL语句中指定为using btree时，底层会建立为ubtree。 • 索引名、约束名、key名GaussDB在同一SCHEMA下不可重复；MySQL在同一表下不可重复。 • MySQL和GaussDB主键上支持列的个数上限不同。 • GaussDB中的主键指定索引名后创建的索引名为用户所指定的索引名，MySQL索引名为PRIMARY。 • 在MySQL 5.7中，索引升降序ASC | DESC被解析但是被忽略，默认行为为ASC；在MySQL 8.0及GaussDB中，ASC | DESC被解析且生效。 • GaussDB中前缀长度不得超过2676，键值的实际长度受内部页面限制，若字段中含有多字节字符或者一个索引上有多个键，索引行长度可能会超限报错。 • GaussDB中主键索引中不支持前缀键，创建或添加主键时不支持指定前缀长度。 • GaussDB数据库的CREATE/DROP INDEX语句中INDEX选项algorithm_option和lock_option目前只在语法上支持，创建时不报错，但实际不起作用。

概述	详细语法说明	差异
支持自增列	ALTER TABLE、CREATE TABLE	• GaussDB的自动增长列建议为索引的第一个字段，否则建表时产生警告，MySQL自动增长列必须为索引第一个字段，否则建表时会报错。GaussDB含有自动增长列的表进行某些操作时会产生错误，例如：ALTER TABLE EXCHANGE PARTITION。 • GaussDB的AUTO_INCREMENT = value语法，value必须为小于2^127的正数。MySQL可以为0，GaussDB不可以。 • GaussDB中当自增值已经达到字段数据类型的最大值时，继续自增将产生错误。MySQL有些场景产生错误或警告，有些场景仍自增为最大值。 • 不支持innodb_autoinc_lock_mode系统变量，GaussDB的GUC参数auto_increment_cache=0时，批量插入自动增长列的行为与MySQL系统变量innodb_autoinc_lock_mode=1相似。 • GaussDB的自动增长列在导入数据或者进行Batch Insert执行计划的插入操作时，对于混合0、NULL和确定值的场景，如果产生错误，后续插入自增值不一定与MySQL完全一致。可以使用GUC参数auto_increment_cache控制预留自增值的数量。 • GaussDB的并行导入或插入自动增长列触发自增时，每个并行线程预留的缓存值也只在其线程中使用，未完全使用完毕的话，也会出现表中自动增长列的值不连续的情况。并行插入产生的自增值结果无法保证与MySQL完全一致。

概述	详细语法说明	差异
		• GaussDB的本地临时表中的自动增长列批量插入时不会预留自增值，正常场景不会产生不连续的自增值。MySQL临时表与普通表中的自动增长列自增结果一致。 • GaussDB的SERIAL数据类型为原有的自增列，与AUTO_INCREMENT自增列有差异。MySQL的SERIAL数据类型就是AUTO_INCREMENT自增列。 • GaussDB不允许auto_increment_offset的值大于auto_increment_increment的值，会产生错误。MySQL允许，并说明auto_increment_offset会被忽略。 • 在表有主键或索引的情况下，ALTER TABLE命令重写表数据的顺序与MySQL不一定相同，GaussDB按表数据存储顺序重写，MySQL会按主键或索引顺序重写，导致自增值的顺序可能不同。 • GaussDB使用ALTER TABLE命令添加或修改自增列时，第一次预留自增值的数量是表统计信息中的行数，统计信息的行数不一定与MySQL一致。 • GaussDB的last_insert_id函数返回值为128位的整型。 • GaussDB在触发器或用户自定义函数中自增时，刷新last_insert_id返回值。MySQL不刷新。 • GaussDB的对GUC参数auto_increment_offset和auto_increment_increment设置超出范围的值会产生错误。MySQL会自动改为边界值。

概述	详细语法说明	差异
		• sql_mode设置no_auto_value_on_zero参数，表定义的自动增长列为非NOT NULL约束，向表中插入数据不指定自动增长列的值时，GaussDB中自动增长列插入NULL值，且不触发自增；MySQL中自动增长列插入NULL值，触发自增。
支持指定字符集与排序规则	ALTER SCHEMA、ALTER TABLE、CREATE SCHEMA、CREATE TABLE	• 指定库级字符集时，除BINARY字符集外，暂不支持创建新库/模式的字符集与数据库的server_encoding不同。 • 指定表级、列级字符集和字符序时，MySQL支持指定与库级字符集、字符序不同的字符集和字符序。在GaussDB中，表级、列级字符集和字符序仅支持BINARY字符集、字符序或者与库级字符集、字符序相同的字符集、字符序。 • 若重复指定字符集或字符序，仅最后一个生效。

概述	详细语法说明	差异
基础表定义语法	CREATE TABLE、ALTER TABLE	• GaussDB不支持以下选项：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、STATS_SAMPLE_PAGES。 • 以下选项在GaussDB中不报错，但实际上也不生效：ENGINE、ROW_FORMAT。 • GaussDB中ALTER TABLE暂不支持DROP INDEX | DROP KEY | ORDER BY子项。 • ALTER TABLE新增列时，MySQL指定字段为NOT NULL时，会将NULL值转为对应类型的默认值插入该列，GaussDB会检查NULL值。 • GaussDB中“ALTER TABLE tablename;”语法不支持tablename为空。 • 在MySQL宽松模式下，会将NULL进行类型转换，并成功插入数据；在MySQL严格模式下不允许插入NULL值。在GaussDB不支持此特性，在宽松模式和严格模式下均不允许插入NULL值。 • CREATE TABLE带CHECK约束的时候，MySQL 8.0会生效，MySQL 5.7只解析语法但不生效。GaussDB在此功能上同步MySQL 8.0版本，

概述	详细语法说明	差异
		且GaussDB CHECK约束可以引用其他列，而MySQL不能。 • GaussDB一个表中最多只能加32767个CHECK约束。

概述	详细语法说明	差异
创建/修改分区表语法	CREATE TABLE PARTITION、CREATE TABLE SUBPARTITION、ALTER TABLE	• MySQL在以下场景支持表达式，不支持多个分区键： – 使用LIST分区/RANGE分区策略，不指定COLUMNS关键字。 – 使用HASH分区策略。 • MySQL在以下场景不支持表达式，支持多个分区键： – 使用LIST分区/RANGE分区策略，指定COLUMNS关键字。 – 使用KEY分区策略。 • GaussDB不支持使用表达式作为分区键。 • GaussDB仅在以下场景支持使用多个分区键：使用LIST分区/RANGE分区策略，且不指定二级分区。 • GaussDB分区表不支持用虚拟生成列作为分区键。 • GaussDB中分区键暂不支持key的column_list为空。 • GaussDB分区表不支持LINEAR/KEY hash。 • GaussDB的CREATE TABLE语句中hash分区表和二级分区表所使用的hash函数与MySQL不一致，因此hash分区表和二级分区表的存储与MySQL有区别。 • MySQL支持修改分区表的分区键信息，GaussDB中不支持。 • GaussDB的CREATE/ALTER TABLE语句中分区表为KEY分区，不支持指定algorithm。 不支持表达式入参的语法： – PARTITION BY HASH() – PARTITION BY KEY() – VALUES LESS THAN()

概述	详细语法说明	差异
交换普通表和分区表分区的数据	ALTER TABLE PARTITION	ALTER TABLE EXCHANGE PARTITION的差异点： ● 对于自增列，MySQL执行ALTER TABLE EXCHANGE PARTITION后，自增列会被重置；GaussDB则不会被重置，自增列则按照旧的自增值递增。 ● MySQL表或分区使用tablespace时，则无法进行分区和普通表数据的交换；GaussDB表或分区使用不同的tablespace时，仍可进行分区和普通表数据的交换。 ● 对于列默认值，MySQL不会校验默认值，因此默认值不同时也可进行分区和普通表数据的交换；GaussDB会校验默认值，如果默认值不同，则无法进行分区和普通表数据的交换。 ● MySQL在分区表或普通表上进行DROP列操作后，表结构仍然一致，则可进行分区和普通表数据的交换；GaussDB需要保证普通表和分区表的被删除列严格对齐才能进行分区和普通表数据的交换。 ● MySQL和GaussDB的哈希算法不同，所以两者在相同的hash分区存储的数据可能不一致，导致最后交换的数据也可能不一致。 ● MySQL的分区表不支持外键，普通表包含外键或其他表引用普通表的外键，则无法进行分区和普通表数据的交换；GaussDB的分区表支持外键，在两个表的外键约束一致时，则可进行分区和普通表数据的交换，GaussDB的分区表不带外键，普通表有其他表引用，如果分区表和普通表表一致，则可进行分区和普通表数据的交换。

概述	详细语法说明	差异
支持CREATE TABLE ... LIKE语法	CREATE TABLE ... LIKE	• 在MySQL 8.0.16之前的版本中，CHECK约束会被语法解析但功能会被忽略，表现为不复制CHECK约束，GaussDB支持复制CHECK约束。 • 对于主键约束名称，在建表时，MySQL所有主键约束名称固定为PRIMARY KEY，GaussDB不支持复制。 • 对于唯一键约束名称，在建表时，MySQL支持复制，GaussDB不支持复制。 • 对于CHECK约束名称，在建表时，MySQL 8.0.16 之前的版本无CHECK约束信息，GaussDB支持复制。 • 对于索引名称，在建表时，MySQL支持复制，GaussDB不支持复制。 • 在跨sql_mode模式建表时，MySQL受宽松模式和严格模式控制，GaussDB可能存在严格模式失效的情况。 例如：源表存在默认值“0000-00-00”，在“no_zero_date”严格模式下，GaussDB建表成功，且包含默认值“0000-00-00”，严格模式失效；而MySQL建表失败，受严格模式控制。
TRUNCATE子分区语法	ALTER TABLE [IF EXISTS] table_name truncate_clause;	支持子项有差异，对于truncate_clause： • GaussDB： TRUNCATE PARTITION { { ALL | partition_name [, ...] } | FOR (partition_value [, ...]) } [UPDATE GLOBAL INDEX] • MySQL： TRUNCATE PARTITION {partition_names | ALL}
删除有依赖的对象	DROP drop_type name CASCADE;	在GaussDB中，删除有依赖的对象需要加CASCADE，MySQL不需要。

概述	详细语法说明	差异
分区表索引	CREATE INDEX	• GaussDB的分区表索引分为LOCAL和GLOBAL两种。LOCAL索引与某个具体分区绑定，而GLOBAL索引则对应整个分区表。 • LOCAL和GLOBAL索引的创建方法和默认规则具体说明参见《M-Compatibility开发指南》的“SQL参考 > SQL语法 > C > CREATE INDEX”章节，例如：在非分区键上创建唯一索引，会默认创建为GLOBAL索引。 • MySQL无GLOBAL索引的概念。在GaussDB中，当分区表索引为GLOBAL索引时，对表分区进行DROP、TRUNCATE、EXCHANGE等操作不会默认更新GLOBAL索引，进而导致GLOBAL索引失效，导致后续语句无法选中该索引。为了避免这种场景，建议用户在使用分区操作语法时在最后显指定UPDATE GLOBAL INDEX子句，或配置全局GUC参数enable_gpi_auto_update为true（推荐），使得在进行分区操作时自定更新GLOBAL索引。 • MySQL支持在相同列上同时创建唯一索引和普通索引。GaussDB在顺序不同的索引列上，支持同时创建分区LOCAL和GLOBAL索引；顺序相同时，则不支持同时创建。

概述	详细语法说明	差异
新增外键约束、修改外键约束的参考字段、被参考字段	CREATE TABLE、ALTER TABLE	• GaussDB外键约束对类型不敏感，如果主表和从表对应的字段数据类型存在隐式类型转换就可以建成。MySQL外键类型敏感。如果两个表对应的列类型不同外键无法建成。 • MySQL不支持通过MODIFY COLUMN或CHANGE COLUMN方式修改表列外键所在列的数据类型或列名等，GaussDB可以。 • 在GaussDB中，允许将外键作为分区键。 • GaussDB创建外键时支持指定MATCH FULL和MATCH SIMPLE选项，如果用户指定了MATCH PARTIAL选项，会提示报错信息。MySQL中支持指定以上选项，但并不生效，行为与MATCH SIMPLE一致。 • GaussDB创建外键时可以指定ON [UPDATE | DELETE] SET DEFAULT选项。MySQL创建外键时指定ON [UPDATE | DELETE] SET DEFAULT选项会报错。 • GaussDB创建外键时，必须在被参考表的被参考列上创建唯一索引。MySQL创建外键时，需要在被参考表的被参考列上创建索引，可以不是唯一索引。 • GaussDB创建外键时，不需要在参考表的参考列上创建索引。MySQL创建外键时，需要在参考表的参考列上创建索引，如果在参考表的参考列上没有索引，则会自动添加一条对应的索引，在删除外键时，不会删除自动添加的索引。 • GaussDB的参考表和被参考表可以都是临时表，不可以在临时表和非临时表之间创建外键。MySQL无法使用

概述	详细语法说明	差异
		临时表作为参考表或被参考表。MySQL在创建外键指定被参考表时，不会匹配当前会话已创建的临时表。 ● GaussDB创建外键时可以不指定被参考表的被参考字段名，此时会将被参考表中的主键作为外键的被参考字段。MySQL中，必须指定被参考表的被参考字段。 ● GaussDB无论foreign_key_checks是否关闭，都可以修改参考字段或被参考字段的数据类型。MySQL仅当foreign_key_checks为off时，才可以修改参考字段或被参考字段的数据类型。 ● GaussDB可以删除参考表的参考字段，此时会级联删除相关外键约束。MySQL在删除参考表的参考字段时，会发生删除失败。 ● GaussDB当删除包含被参考表的模式，且参考表在其他模式中时，foreign_key_checks为on时，会级联删除参考表上的外键约束。在MySQL中，如果foreign_key_checks为on，则删除失败。
CMK密钥轮转，轮换加密COLUMN ENCRYPTION KEY的CLIENT MASTER KEY，对COLUMN ENCRYPTION KEY明文进行重加密。	ALTER COLUMN ENCRYPTION KEY	M-Compatibility模式不支持全密态。因此不支持该语法。
密态等值查询特性使用多级加密模型，主密钥加密列密钥，列密钥加密数据。本语法用于创建主密钥对象。	CREATE CLIENT MASTER KEY	M-Compatibility模式不支持全密态。因此不支持该语法。
创建一个列加密密钥，该密钥可用于加密表中的指定列。	CREATE COLUMN ENCRYPTION KEY	M-Compatibility模式不支持全密态。因此不支持该语法。

概述	详细语法说明	差异
全密态功能，传输密钥到服务端缓存，只在开启内存解密逃生通道的情况下使用。	\send_token	M-Compatibility模式不支持全密态。因此不支持该语法。
全密态功能，传输密钥到服务端缓存，只在开启内存解密逃生通道的情况下使用。	\st	M-Compatibility模式不支持全密态。因此不支持该语法。
全密态功能，销毁服务端缓存的密钥，只在开启内存解密逃生通道的情况下使用。	\clear_token	M-Compatibility模式不支持全密态。因此不支持该语法。
全密态功能，销毁服务端缓存的密钥，只在开启内存解密逃生通道的情况下使用。	\ct	M-Compatibility模式不支持全密态。因此不支持该语法。
在全密态数据库特性中,用于设置访问外部密钥管理者的参数。	\key_info KEY_INFO	M-Compatibility模式不支持全密态。因此不支持该语法。
全密态功能，用于开启三方动态库功能与加载三方动态库时的参数设置。	\crypto_module_info MODULE_INFO	M-Compatibility模式不支持全密态。因此不支持该语法。
全密态功能，用于开启三方动态库功能与加载三方动态库时的参数设置。	\cmi MODULE_INFO	M-Compatibility模式不支持全密态。因此不支持该语法。

概述	详细语法说明	差异
支持更改表名语法	ALTER TABLE tbl_name RENAME [TO | AS | =] new_tbl_name; 或RENAME {TABLE | TABLES} tbl_name TO new_tbl_name [, tbl_name2 TO new_tbl_name2, ...];	• GaussDB的ALTER RENAME语法仅支持修改表名称功能操作，不能耦合其它功能操作。 • GaussDB中，仅旧表名字段支持使用schema.table_name格式，且新表名与旧表名将属于同一个Schema。 • GaussDB不支持新旧表跨Schema重命名操作；但如有权限，则可在当前Schema下修改其他Schema下表名称。 • GaussDB的RENAME多组表的语法支持全为本地临时表的重命名，不支持本地临时表和非本地临时表组合的场景。

概述	详细语法说明	差异

支持CREATE VIEW AS SELECT语法	CREATE VIEW table_name AS query;	• 当不开启精度传递开关（m_format_behavior_compat_options不开启enable_precision_decimal选项）时，针对以下类型，不支持CREATE VIEW view_name AS query语法中query包含计算操作（如函数调用、使用操作符计算），仅允许直接的字段调用（如 SELECT col1 FROM table1）。精度传递开关打开（m_format_behavior_compat_options开启enable_precision_decimal选项）时可以使用。BINARY[(n)]、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
		• 在query为简单查询场景下，GaussDB针对上述类型的计算操作进行报错提示，例如:
		m_db=# CREATE TABLE TEST (salary int(10));
		CREATE TABLE
		
		m_db=# INSERT INTO TEST VALUES(8000);

概述	详细语法说明	差异

		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.
		● 在query为复合查询，子查询等非简单查询场景下，GaussDB针对上述类型的计算操作与MySQL存在差异，GaussDB下新创建表的数据类型列精度属性不保留。
		● CREATE VIEW AS SELECT场景。当UNION嵌套子查询时，MySQL会对内层子查询新建临时表。若临时表的返回类型为tinytext、text、mediumtext、longtext时，MySQL会按照所属类型默认的最大字节长度进行计算。而GaussDB会以创建临时表的实际字节长度进行计算。因此最终GaussDB聚合结果的文本类型有可能小于MySQL的聚合结果。如MySQL返回longtext，GaussDB返回mediumtext。如：
		MySQL 5.7行为:
		mysql> CREATE TABLE IF NOT EXISTS tb_1 (id int,col_text2 text);
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> CREATE TABLE IF NOT EXISTS tb_2 (id int,col_text2 text);
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> CREATE VIEW v1 AS SELECT * FROM (SELECT cast(col_text2 AS char) c37 FROM tb_1) t1
		-> UNION ALL SELECT * FROM (SELECT cast(col_text2 as char) c37 FROM tb_2) t2;
		Query OK, 0 rows affected (0.00 sec)

概述	详细语法说明	差异

		mysql> DESC v1;
		+-------+----------+------+-----
		+---------+-------+
		| Field | Type | Null | Key |
		Default | Extra |
		+-------+----------+------+-----
		+---------+-------+
		| c37 | longtext | YES | |
		NULL | |
		+-------+----------+------+-----
		+---------+-------+
		1 row in set (0.00 sec)
		GaussDB行为：
		mysql_regression=# CREATE
		TABLE IF NOT EXISTS tb_1 (id
		int,col_text2 text);
		CREATE TABLE
		mysql_regression=# CREATE
		TABLE IF NOT EXISTS tb_2 (id
		int,col_text2 text);
		CREATE TABLE
		mysql_regression=# CREATE VIEW
		v1 AS SELECT * FROM (SELECT
		cast(col_text2 AS char) c37 FROM
		tb_1) t1
		mysql_regression-# UNION ALL
		SELECT * FROM (SELECT
		cast(col_text2 AS char) c37 FROM
		tb_2) t2;
		CREATE VIEW
		mysql_regression=# DESC v1;
		Field | Type | Null | Key |
		Default | Extra
		-------+------------+------+-----
		+---------+-------
		c37 | mediumtext | YES |
		| |
		(1 row)
		● 使用bitstring常量进行视图创建时，与MySQL不一致，MySQL转换成hexstring进行创建，GaussDB使用bitstring直接创建。由于bitstring常量为unsigned值，因此GaussDB创建视图的属性为unsigned。
		– MySQL 5.7行为：
		mysql> SELECT version();
		+-----------------+
		| version() |
		+-----------------+
		| 5.7.44-debug-log |
		+-----------------+
		1 row in set (0.00 sec)
		
		mysql> DROP VIEW IF EXISTS v1;
		Query OK, 0 rows affected, 1
		warning (0.00 sec)

概述	详细语法说明	差异

		mysql> CREATE VIEW v1 AS SELECT b'101'/b'101' AS c22;
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> DESC v1;
		+-------+----------------------+------+-----+---------+-------+
		| Field | Type | Null | Key | Default | Extra |
		+-------+----------------------+------+-----+---------+-------+
		| c22 | decimal(5,4) unsigned | YES | | NULL | |
		+-------+----------------------+------+-----+---------+-------+
		1 row in set (0.00 sec)
		
		mysql> SHOW CREATE VIEW v1;
		+------
		+----------
		---------- 	
		---------- 	
		---------- 	
		+----------
		+----------+
		| View | Create View
		
		| character_set_client | collation_connection |
		+------
		+----------
		---------- 	
		---------- 	
		---------- 	
		+----------
		+----------+
		| v1 | CREATE ALGORITHM=UNDEFINED DEFINER=`omm`@`%` SQL SECURITY DEFINER VIEW `v1` AS SELECT (0x05 / 0x05) AS `c22` | utf8mb4 | utf8mb4_general_ci |
		+------
		+----------
		---------- 	
		---------- 	
		---------- 	
		+----------
		+----------+
		1 row in set (0.00 sec)
		– GaussDB行为：
		m_db=# DROP VIEW IF EXISTS v1;
		DROP VIEW
		
		m_db=# CREATE VIEW v1 AS SELECT b'101'/b'101' AS c22;
		CREATE VIEW
		
		m_db=# DESC v1;
		Field | Type | Null | Key | Default | Extra
		-------+--------------+------+-----+---------+-------
		c22 | decimal(5,4) | YES |

概述	详细语法说明	差异
		| | (1 row)
视图依赖差异	CREATE VIEW、ALTER TABLE	MySQL中视图存储，只记录目标表的表名、列名、数据库名信息，不记录目标表的唯一标识；GaussDB会将创建视图时的SQL解析，并存储目标表的唯一标识。因此存在如下差异： ● 修改存在视图依赖的列的数据类型，MySQL中对应的视图不感知目标表的修改，因此可以修改成功；GaussDB视图中的列禁止修改数据类型，因此无法修改该列的数据类型。 ● 重命名存在视图依赖的列，MySQL中对应的视图不感知目标表的修改，因此可以修改成功，但是后续无法查询该视图；GaussDB视图中，每个列精确存储了其对应的表和列的唯一标识，因此表中的列名可以修改成功，视图中的列名不被修改，且后续可以查询该视图。
修改视图定义	CREATE OR REPLACE VIEW、ALTER VIEW	● MySQL可以对视图的任何属性进行修改；GaussDB禁止修改不可更新视图当中的列名、列类型以及删除列，允许修改可更新视图列名、列类型以及删除列。 ● MySQL对嵌套视图的底层视图执行修改列操作后，只要列名存在，上层视图就可以正常使用；GaussDB对嵌套视图中底层视图做修改列名、列类型以及删除列操作，会导致上层视图不可用。

概述	详细语法说明	差异
ANALYZE分区语法	ALTER TABLE tbl_name ANALYZE PARTITION {partition_names | ALL}	• GaussDB下该语法仅支持分区统计信息收集功能。 • MySQL下分区名partition_names 大小写不区分，GaussDB下分区名不带反引号的情况下不区分大小写，带反引号时区分大小写。 • GaussDB下执行成功回显ALTER TABLE，执行报错由已有错误码报错信息决定，MySQL下执行结果以表格回显形式显示。

概述	详细语法说明	差异
支持虚拟生成列语法	[GENERATED ALWAYS] AS（generation_expr）[STORED | VIRTUAL]	• MySQL数据库中虚拟生成列支持创建索引，GaussDB数据库中不支持。 • MySQL数据库中虚拟生成列支持作为分区键，GaussDB数据库中不支持。 • GaussDB数据库中生成列的CHECK约束兼容MySQL 8.0数据库的行为，即CHECK约束检查生效。 • MySQL数据库中存储生成列作为分区键时，支持ALTER TABLE修改存储生成列，GaussDB数据库中不支持。 • MySQL数据库中向可被更新的视图中更新生成列的数据时，支持指定关键字DEFAULT，GaussDB数据库中不支持。 • MySQL数据库中虚拟生成列支持IGNORE特性，GaussDB数据库中不支持。 • 在GaussDB数据库中查询虚拟生成列等价于查询虚拟生成列的表达式（在表达式与列定义的数据类型、字符集或字符序不一致时，会将表达式向列定义的类型做隐式转换处理），此行为在查询虚拟生成列用于建表或建视图等其他行为，可能导致数据类型与MySQL数据库存在差异。例如：在使用CREATE TABLE AS建表时，如果源表中虚拟生成列定义为FLOAT类型，在目标表中其对应列的数据类型可能为DOUBLE，与MySQL数据库存在差异。 • GaussDB的generated always as语句不能再引用由generated always as生成的列，MySQL可以。

概述	详细语法说明	差异
通过CREATE TABLE SELECT新建表并插入数据	CREATE TABLE [AS] SELECT	• 不支持创建分区表。 • 不支持replace/ignore。 • 如果SELECT列非直接表列，则默认允许NULL且无默认值。如：CREATE TABLE t1 SELECT unix_timestamp('2008-01-02 09:08:07.3465') AS a，创建出来的新表的字段a允许为NULL且无默认值。 • 如果需要使用完整的功能，需要将GUC参数m_format_behavior_compat_options开启enable_precision_decimal选项，否则由于版本兼容性问题，涉及数据类型精度相关的类型将导致行为报错。比如：如果SELECT列包含非直接表列（表达式、函数、常量等），以及union场景（因为涉及结果类型推导），都会报错。 • 使用CREATE TABLE AS SELECT建表，表中列名最大长度为63，超过时超过部分将会被截断；MySQL超过最大长度64时报错。

概述	详细语法说明	差异
UTF8字符集编码最大长度不同，导致创建表/视图的字段长度产生差异	CREATE TABLE [AS] SELECT; CREATE VIEW [AS] SELECT	当MySQL的字符集为utf8，GaussDB的字符集为utf8（utf8mb4）时，由于MySQL的 UTF8编码最大为3字节，GaussDB的utf8（utf8mb4）的编码最大为4字节，开启GUC参数 m_format_behavior_compat_options = 'enable_precision_decimal' 时，CREATE TABLE AS (ctas)和CREATE VIEW AS (cvas)创建文本类型时（包括二进制文本）可能存在差异： 由于ctas和cvas的场景会依赖字符集的最长字节长度，例如某一节点返回的最大字节长度GaussDB与MySQL均为1024，那么最终返回的字符长度，MySQL为341（1024/3），GaussDB为256（1024/4）。如： MySQL 5.7行为： mysql> CREATE TABLE t1 AS SELECT (CASE WHEN true THEN min(521.2312) ELSE group_concat(115.0414) END) res1; Query OK, 1 row affected (0.06 sec) Records: 1 Duplicates: 0 Warnings: 0 mysql> DESC t1; +-------+--------------+------+-----+---------+-------+ | Field	Type	Null	Key	Default	Extra | +-------+--------------+------+-----+---------+-------+ | res1	varchar(341)	YES		NULL	| +-------+--------------+------+-----+---------+-------+ 1 row in set (0.01 sec) GaussDB行为： mysql_regression=# CREATE TABLE t1 AS SELECT (CASE WHEN true THEN min(521.2312) ELSE group_concat(115.0414) END) res1; INSERT 0 1 mysql_regression=# DESC t1; Field	Type	Null	Key	Default	Extra -------+--------------+------+-----+---------+------- res1	varchar(256)	YES			 (1 row)

概述	详细语法说明	差异

指定字段的默认值	CREATE TABLE、ALTER TABLE	• MySQL 5.7指定默认值时，仅支持不带括号形式的默认值。MySQL 8.0与GaussDB支持带括号形式的默认值。
		-- 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)
		• MySQL给BLOB、TEXT、JSON数据类型指定默认值时必须给默认值添加括号，GaussDB给上述数据类型指定默认值时可以不添加括号。
		• GaussDB指定默认值时不会校验默认值是否溢出，但是如果是varbianry类型的话会校验是否溢出；MySQL

概述	详细语法说明	差异

		指定不带括号形式的默认值时会校验是否溢出，指定带括号形式的默认值时不会校验是否溢出。
		• GaussDB支持使用DATE/TIME/TIMESTAMP开头的时间常量给字段指定默认值，MySQL使用DATE/TIME/TIMESTAMP开头的时间常量给字段指定默认值时必须在默认值外添加括号。
		-- 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 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

概述	详细语法说明	差异
		00:00:00')); Query OK, 0 rows affected (0.19 sec)

3.7.4 DML

表 3-32 DML 语法兼容介绍

概述	详细语法说明	差异
DELETE支持从多个表中删除数据	DELETE	• 在多表删除执行过程中，若发现将要删除的元组被其他会话并发修改，会取该条会话匹配中所有元组的最新值重新进行匹配，若依然满足条件再对这条元组进行删除。这个过程中MySQL对所有目标表的删除是一致的，而GaussDB仅会对涉及并发更新的目标表的元组进行重新匹配，可能产生数据不一致。 • 多表操作语法中目标表和范围表的校验规则与MySQL会存在差异，设置GUC兼容性参数m_format_dev_version为's2'后，检验规则保持一致。
UPDATE支持从多个表中更新数据	UPDATE	在多表更新执行过程中，若发现将要更新的元组被其他会话并发修改，会取该条会话匹配中所有元组的最新值重新进行匹配，若依然满足条件再对这条元组进行更新。这个过程中MySQL对所有目标表的更新是一致的，而GaussDB仅会对涉及并发更新的目标表的元组进行重新匹配，可能产生数据不一致。
SELECT INTO语法	SELECT	• GaussDB可以使用SELECT INTO根据查询结果创建一个新表，MySQL不支持。 • GaussDB的SELECT INTO语法不支持将多个查询进行集合运算后的结果作为查询结果。

概述	详细语法说明	差异

REPLACE INTO语法	REPLACE	时间类型初始值的差异。例如:
		• MySQL不受严格模式和宽松模式的影响，可向表中插入时间0值，即：
		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)
		• GaussDB在宽松模式下才可以成功插入时间0值，即
		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)
		在严格模式下，则报错：
		The date, time, datetime, timestamp, or year is incorrect.

概述	详细语法说明	差异
LOAD DATA导入数据功能	LOAD DATA	在使用LOAD DATA导入数据功能时，GaussDB与MySQL相比有如下差异： ● LOAD DATA语法执行结果与MySQL严格模式一致，宽松模式暂未适配。 ● IGNORE与LOCAL参数功能仅为当导入数据与表中数据存在冲突时，忽略当前冲突行数据功能和当文件中字段数小于指定表中列数时自动为其余列填充默认值功能，其余功能暂未适配。 ● [(col_name_or_user_var [, col_name_or_user_var] ...)]指定列参数不支持重复指定列。 ● [FIELDS TERMINATED BY 'string']指定换行符不能与[LINES TERMINATED BY 'string']分隔符相同。 ● 执行LOAD DATA语法写入表中的数据若无法转换为表中数据类型格式时报错。 ● LOAD DATA SET表达式中不支持指定列名计算。 ● LOAD DATA只能用于表，不能用于视图。 ● Windows下的文件与Linux环境下文件默认换行符存在差异，LOAD DATA无法识别此场景会报错，建议用户导入时检查导入文件行尾换行符。 ● GaussDB不设置GUC参数m_format_behavior_compat_options值时，LOAD DATA无论是否指定LOCAL参数都仅支持从服务端导入数据；MySQL在指定LOCAL参数时支持从客户端环境导入数据，不指定LOCAL时则从服务端环境导入数据；GaussDB设置GUC参数值包含enable_load_data_remote_transmission后，LOAD DATA LOCAL参数行为与MySQL一致。

概述	详细语法说明	差异
LIMIT子句差异	DELETE、SELECT、UPDATE	各个语句的limit子项与MySQL的limit项当前存在差异。 GaussDB中limit参数最大值为BIGINT类型限制（超过92233720368547758O7报错）。在MySQL中，limit最大值为unsigned LONGLONG类型限制（超过18446744073709551615报错）。 limit可以设置小数值，实际执行时四舍五入。MySQL不能取小数。
反斜杠(\)用法差异	INSERT	反斜杠（\）的用法在GaussDB和MySQL中都可以由参数控制但当前默认用法不同： MySQL中使用参数NO_BACKSLASH_ESCAPES控制字符串和标识符中的反斜杠（\）被解析为普通字符还是转义字符，默认反斜杠字符（\）作为字符串和标识符中的转义字符。使用“SET sql_mode='NO_BACKSLASH_ESCAPES';”语句可以禁用反斜杠字符（\）作为字符串和标识符中的转义字符。 GaussDB中使用参数standard_conforming_strings控制字符串和标识符中的反斜杠（\）被解析为普通字符还是转义字符。默认值为on，在普通字符串文本中按照SQL标准把反斜杠（\）当普通文本。使用“SET standard_conforming_strings=off;”语句将反斜杠字符（\）作为字符串和标识符中的转义字符。
插入值少于字段数目时，MySQL报错，GaussDB补充空值	INSERT	GaussDB不指定列的列表时，如果插入值少于字段数目，默认按建表时的字段顺序赋值。字段上有非空约束时报错，没有非空约束时，如果指定了默认值则缺省部分补充默认值，若未指定默认值则补充空。

概述	详细语法说明	差异
ORDER BY中排序的列必须包括在结果集的列中	SELECT	在GaussDB中，在与GROUP BY子句一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。在与DISTINCT关键字一起使用的情况下，ORDER BY中排序的列必须包括在SELECT语句所检索的结果集的列中。
外键数据类型是timestamp/datetime时，UPDATE/DELETE外表报错	UPDATE/DELETE	外键数据类型是timestamp/datetime时，UPDATE/DELETE外表报错，MySQL成功。

概述	详细语法说明	差异

NATURAL JOIN语法	SELECT	• 在GaussDB中，NATURAL [[LEFT | RIGHT] OUTER] JOIN允许不指定LEFT | RIGHT，不指定时NATURAL OUTER JOIN为NATURAL JOIN。允许连续使用多次JOIN。
		• GaussDB join的顺序严格按照从左往右，MySQL可能会调整顺序。
		• GaussDB和MySQL在natural join与using时均不允许左表或右表参与join的字段出现歧义（一般由左或右临时表中重名字段造成）。因为两者join的顺序有差别，故行为上可能有差别。
		– GaussDB的行为：
		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, 因为:列a,b在t1 join t2 得到的临时表中存在重复，故nature join存在歧义。
		ERROR: common column name "a" appears more than once in left table
		– MySQL的行为：
		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

JOIN语法	SELECT	GaussDB JOIN不支持使用逗号“,”的连接方式，MySQL支持。 GaussDB不支持USE INDEX FOR JOIN。 GaussDB中STRAIGHT_JOIN多表关联场景下生成的执行计划，与MySQL可能存在差异。

概述	详细语法说明	差异

SELECT语句显示的列名	SELECT	• 为了使SELECT语句显示的列名与MySQL一致，GaussDB需要打开列名回显控制开关：
		`SET m_format_behavior_compat_options = 'select_column_name'`
		• GaussDB不设置此配置项时：
		– SELECT系统函数：回显为系统函数名。
		– SELECT表达式：回显为?column?。
		– SELECT布尔值：回显为bool。
		• GaussDB设置此配置项时，列名回显为全部的函数或表达式输入。
		– 对于普通注释，MySQL客户端会将注释忽略，gsql客户端和pymysql不会忽略。
		– 对于如/*!形式开头的注释，MySQL服务端会将其转为可执行语句，M兼容暂不支持识别此类注释，因此作为普通注释处理。
		– 对于包含--且后面无空格的表达式，M兼容不支持将其识别为两个-号，当前识别为注释；MySQL服务端将其识别为两个-号。
		– 如果显示的列名字符串中含有转义字符，只有在设置了m_format_behavior_compat_options参数包含enable_escape_string项后才会显示转义后的字符，否则会显示转义字符本身，比如“SELECT"abc\tdef";”M兼容在未开启上述设置时显示为abc\tdef。
		`m_db=# SET m_format_behavior_compat_options='select_column_name,enable_escape_string';`
		`SET`
		`m_db=# SELECT "abc\tdef";`
		` abc  def`
		`----------`
		` abc  def`
		`(1 row)`

概述	详细语法说明	差异

		m_db=# SET m_format_behavior_compat_options='select_column_name';
		SET
		m_db=# SELECT "abc\tdef";
		 abc\tdef
		---------- 	
		 abc\tdef
		(1 row)
		– 列名超过63个字符时，会截断后面部分。
		– 表达式最后的部分为注释时，则不会显示最后的注释以及与注释相连的空格。
		m_db=# SELECT 123 /* 456 */;
		 123
		----- 	
		 123
		(1 row)
		– 表达式为布尔值时，无论输入大小写，回显为TRUE或FALSE。
		m_db=# SELECT true;
		 TRUE
		------ 	
		 t
		(1 row)
		– 表达式为null时，无论输入大小写，回显为NULL。
		m_db=# SELECT null;
		 NULL
		------ 	
		
		(1 row)
		– 表达式包含-时，会将全部的输入作为列名输出。
		m_db=# SELECT (+-+1);
		 (+-+1)
		-------- 	
		 -1
		(1 row)
		
		m_db=# SELECT -true;
		 -true
		------- 	
		 -1
		(1 row)
		
		m_db=# SELECT -null;
		 -null
		------- 	
		
		(1 row)
		● 当使用pymysql执行SELECT语句，查询的字符串前缀字符不是ASCII字符，且数据库的编

概述	详细语法说明	差异
		码不是UTF-8时，显示的列名会与MySQL存在差异。
SELECT导出文件（into outfile）	SELECT ... INTO OUFILE ...	SELECT INTO OUTFILE语法，导出文件中FLOAT、DOUBLE、REAL类型的值显示精度和MySQL存在差异，不影响COPY导入和导入后的值。

概述	详细语法说明	差异

SELECT/UPDATE/INSERT/REPLACE指定模式名、表名	SELECT/UPDATE/INSERT/REPLACE	• SELECT语句指定投影列时，MySQL支持“模式名.表别名.列名”的三段式用法，GaussDB不支持。
		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
		• UPDATE/REPLACE SET中，MySQL的三段式用法为database.table.column；GaussDB的三段式用法为table.column.field，其中field为指定复合类型中的属性。
		• INSERT ... SET中，MySQL支持使用column、table.column和database.table.column；GaussDB只支持使用column，不支持使用table.column和database.table.column。
		• INSERT ... SET中，MySQL支持SET子句中等式右边引用列名以及包含列名的表达式，GaussDB不支持：
		– GaussDB的行为：
		m_db=# CREATE TABLE t2 (a int default 3, b int default 5);
		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

概述	详细语法说明	差异

		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
		– MySQL的行为：
		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;
		+------+------+
		| 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 |

概述	详细语法说明	差异

		<pre>| 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)</pre>

UPDATE SET执行顺序与MySQL存在差异	UPDATE ... SET	MySQL中，UPDATE SET的顺序是从前往后依次UPDATE，前面UPDATE的结果会影响后面的结果，且允许多次设置同一列；GaussDB中为先取出原来的所有相关的数据，再一次性UPDATE，且不允许多次设置同一列，二者存在差异。设置GUC兼容性参数m_format_dev_version为's2'后，仅在单表场景可保持与MySQL行为一致，既支持同一列设置多次，且引用更新后的结果。
IGNORE特性	UPDATE/DELETE/INSERT	MySQL数据库和GaussDB执行过程的差异，因此产生的WARNING条数和WARNING信息可能存在不同。

概述	详细语法说明	差异
SHOW COLUMNS语法	SHOW	• 用户权限验证与MySQL存在差异。 – GaussDB中需要拥有指定表所在Schema的USAGE权限，同时还需要拥有指定表的任意表级权限或列级权限，仅显示拥有SELECT、INSERT、UPDATE、REFERENCES和COMMENT权限的列信息。 – MySQL中需要拥有指定表的任意表级权限或列级权限，仅显示拥有SELECT、INSERT、UPDATE、REFERENCES和COMMENT权限的列信息。 • LIKE和WHERE子句中涉及到字符串比较操作时，Field、Collation、Null、Extra、Privileges字段使用字符集utf8mb4、字符序utf8mb4_general_ci，Type、Key、Default、Comment字段使用字符集utf8mb4、字符序utf8mb4_bin。 • GaussDB中建议用户在WHERE子句中，不要对返回字段以外的列进行选择，否则可能会出现非预期的报错。 -- 预期报错 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'; ^ -- 非预期报错 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'; ^

概述	详细语法说明	差异
SHOW CREATE DATABASE语法	SHOW	用户权限验证与MySQL存在差异。 ● GaussDB中需要拥有指定Schema的USAGE权限。 ● MySQL中需要拥有任意库级权限（除GRANT OPTION和USAGE）、任意表级权限（除GRANT OPTION）或任意列级权限。

概述	详细语法说明	差异
SHOW CREATE TABLE 语法	SHOW	• 用户权限验证与MySQL存在差异。 – GaussDB中需要拥有指定表所在Schema的USAGE权限和指定表的任意表级权限。 – MySQL中需要拥有指定表的任意表级权限（除GRANT OPTION）。 • 返回的建表语句与MySQL存在差异。 – GaussDB中索引以CREATE INDEX语句的形式返回。MySQL中表的索引在CREATE TABLE语句中返回。主要因为GaussDB中CREATE INDEX语法支持的可选参数范围与CREATE TABLE语法中创建索引不同，因此某些索引无法在CREATE TABLE语句中创建。 – GaussDB中CREATE TABLE语法的ENGINE和ROW_FORMAT选项仅做了语法适配，实际不生效，因此在返回的建表语句中不予显示。 • 设置兼容性参数m_format_dev_version为's2'后，返回的建表语句才兼容MySQL。兼容的内容包括：列注释位置变更、表注释位置变更、全局临时表ON COMMIT选项位置变更、主键与唯一约束位置变更、主键与唯一约束中的USING INDEX TABLESPACE选项不再显示以及索引注释位置变更。

概述	详细语法说明	差异
SHOW CREATE VIEW 语法	SHOW	• 用户权限验证与MySQL存在差异。 – GaussDB中需要拥有指定视图所在Schema的USAGE权限和指定视图的任意表级权限。 – MySQL中需要拥有指定视图的表级SELECT和表级SHOW VIEW权限。 • 返回的视图创建语句与MySQL存在差异。以SELECT * FROM tbl_name形式创建的视图，GaussDB中*不会被展开，而MySQL中会展开。 • 返回结果中的character_set_client和collation_connection字段与MySQL存在差异。 – MySQL中显示视图创建时系统变量character_set_client和collation_connection的会话值 – GaussDB中未记录相关元数据，显示为NULL。
SHOW PROCESSLIST 语法	SHOW	GaussDB中该命令的查询结果中的字段内容和大小写与information_schema.processlist视图内字段内容与大小写保持一致，MySQL中可能存在差异。 • GaussDB中用户只能访问自己的线程信息，拥有SYSADMIN权限的用户可以访问所有用户的线程信息。 • MySQL中用户只能访问自己的线程信息，拥有PROCESS权限的用户可以访问所有用户的线程信息。
SHOW [STORAGE] ENGINES	SHOW	GaussDB中该命令的查询结果中的字段内容和大小写与information_schema.engines视图内字段内容与大小写保持一致，MySQL中可能存在差异。因为MySQL与GaussDB的存储引擎不同，所以该指令查询的结果不同。

概述	详细语法说明	差异
SHOW [SESSION] STATUS	SHOW	GaussDB中该命令的查询结果中的字段内容和大小写与information_schema.session_status视图内字段内容与大小写保持一致，MySQL中可能存在差异。GaussDB中当前仅支持Threads_connected和Uptime。
SHOW [GLOBAL] STATUS	SHOW	GaussDB中该命令的查询结果中的字段内容和大小写与information_schema.global_status视图内字段内容与大小写保持一致，MySQL中可能存在差异。GaussDB中当前仅支持Threads_connected和Uptime。
SHOW INDEX	SHOW	• 用户权限验证与MySQL存在差异。 – GaussDB中需要拥有指定SCHEMA的USAGE权限和指定表的任意表级权限或者任意列级权限。 – MySQL中需要拥有指定表的任意表级权限（除GRANT OPTION）或者任意列级权限。 • GaussDB中临时表存储于独立的临时Schema中，在使用FROM或IN db_name条件来展示指定临时表索引信息时，须指明db_name为临时表所在的Schema才能展示临时表索引信息，否则会提示不存在该临时表，这一点和MySQL在部分情况下存在差异。 • GaussDB中，查询结果中Table、Index_type、Index_comment字段使用字符集utf8mb4、字符序utf8mb4_bin；字段Key_name，Column_name、Collation、Null、Comment字段使用字符集utf8mb4、字符序utf8mb4_general_ci。

概述	详细语法说明	差异
SHOW SESSION VARIABLES	SHOW	GaussDB中查询结果中字段内容及大小写与information_schema.session_variables视图内字段内容及大小写保持一致，与MySQL可能存在差异。 GaussDB中对查询结果中字段使用LIKE和WHERE进行选择时，排序规则与information_schema.session_variables视图内对应字段保持一致。
SHOW GLOBAL VARIABLES	SHOW	GaussDB中查询结果中字段内容及大小写与information_schema.global_variables视图内字段内容及大小写保持一致，与MySQL可能存在差异。 GaussDB中对查询结果中字段使用LIKE和WHERE进行选择时，排序规则与information_schema.global_variables视图内对应字段保持一致。
SHOW CHARACTER SET	SHOW	GaussDB中查询结果中字段内容及大小写与information_schema.character_sets视图内字段内容及大小写保持一致，与MySQL可能存在差异。 GaussDB中对查询结果中字段使用LIKE和WHERE进行选择时，排序规则与information_schema.character_sets视图内对应字段保持一致。
SHOW COLLATION	SHOW	GaussDB中查询结果中字段内容及大小写与information_schema.collations视图内字段内容及大小写保持一致，与MySQL可能存在差异。 GaussDB中对查询结果中字段使用LIKE和WHERE进行选择时，排序规则与information_schema.collations视图内对应字段保持一致。

概述	详细语法说明	差异
SHOW TABLES	SHOW	• LIKE行为存在差异，具体请参见**操作符**章节的“LIKE”。 • WHERE表达式行为存在差异，具体行为请参见GaussDB数据库的“WHERE表达式”。 • GaussDB中：表和数据库的权限需要分开赋予用户，查询的数据库必须是用户在SHOW SCHEMAS上可以查询到，不能仅仅有表的权限，必须还需要有数据库的权限。MySQL中只要拥有表权限即可访问。 • GaussDB中：校验逻辑中优先校验Schema是否存在，后校验当前用户是否对Schema具有权限，与MySQL存在差异。 • GaussDB中：查询结果中字段使用字符集utf8mb4、字符序utf8mb4_bin。 • GaussDB中：LIKE子句中，当目标database是information_schema时，pattern被转为小写再进行匹配。在MySQL 8.0中，当目标database是information_schema时，pattern被转为大写再进行匹配。

概述	详细语法说明	差异
SHOW TABLE STATUS	SHOW	• GaussDB中：该语法展示数据依赖information_schema下的tables视图。MySQL中tables指定的是表。 • GaussDB中：表和数据库的权限需要分开赋予用户，查询的数据库必须是用户在SHOW SCHEMAS上可以查询到，不能仅仅有表的权限，必须还需要有数据库的权限。MySQL中只要拥有表权限即可访问。 • GaussDB中：校验逻辑中优先校验Schema是否存在，后校验当前用户是否对Schema具有权限，与MySQL存在差异。 • GaussDB中：对查询结果中字段使用LIKE和WHERE进行选择时，排序规则与information_schema.tables视图内对应字段保持一致。 • GaussDB中：LIKE子句中，当目标database是information_schema时，pattern被转为小写再进行匹配。在MySQL 8.0中，当目标database是information_schema时，pattern被转为大写再进行匹配。
SHOW DATABASES	SHOW	GaussDB中：查询结果中字段使用字符集utf8mb4、字符序utf8mb4_bin。
支持SQL_MODE中ONLY_FULL_GROUP_BY选项	SELECT	SELECT列表中非聚合函数列与GROUP BY字段不一致时，非聚合函数列都需要出现GROUP BY列表或WHERE列表中，且WHERE子句中的列需要等于某一常量时，不报错。其中WHERE子句中的列，GaussDB支持入参数为1的函数列表达式，MySQL不支持函数列表达式。 GaussDB只支持GROUP BY后字段为正整数。

概述	详细语法说明	差异
使用SELECT查询系统参数、用户变量	SELECT @variable、SELECT @@variable	• MySQL支持查询用户变量时，不添加具体的变量名（即SELECT @），GaussDB不支持。 MySQL的行为： mysql> SELECT @; +------------+ | @ | +------------+ | NULL | +------------+ 1 row in set (0.00 sec) GaussDB的行为： m_db=# SELECT @; ERROR: syntax error at or near "@" LINE 1: SELECT @; ^ • 当查询的系统变量类型为BOOLEAN时，GaussDB中输出结果为t/f，MySQL为1/0，BOOLEAN类型实际映射为TINYINT类型。

概述	详细语法说明	差异

子查询	SELECT	• GaussDB不支持子查询结果包含多列，包含多列时执行会报错；MySQL支持子查询包含多列。
		MySQL的行为：
		mysql> SELECT row(1,2) = (SELECT 1,2);
		+-------------------------+
		| row(1,2) = (SELECT 1,2) |
		+-------------------------+
		| 1 |
		+-------------------------+
		1 row in set (0.00 sec)
		GaussDB的行为：
		m_db=# SELECT row(1,2) = (SELECT 1,2);
		ERROR: subquery must return only one column
		LINE 1: SELECT row(1,2) = (SELECT 1,2); ^
		• 在开启精度传递的场景下，MySQL对于子查询FROM子句中返回类型为numeric类型时，如果满足以下条件之一：
		– SELECT子句中存在GROUP BY；
		– SELECT子句中存在HAVING；
		– SELECT子句中存在DISTINCT；
		– SELECT子句中存在LIMIT；
		– SELECT子句中不存在FROM table；
		– SELECT子句中存在用户自定义变量赋值语句；
		将可能发生精度截断，将此类子查询作为下一步运算的中间计算值时，会导致GaussDB的精度结果比MySQL高。
		MySQL的行为：
		mysql> SELECT greatest((SELECT * FROM (SELECT DISTINCT c2/1.61 FROM t_time) t4), 1.00000000000000000);
		+--+
		| greatest((SELECT * FROM (SELECT DISTINCT c2/1.61 FROM t_time) t4), 1.00000000000000000) |
		+--+
		|

概述	详细语法说明	差异

		39144.72670800000000000 |
		+----------
		---------- 	
		--+
		1 row in set (0.00 sec)
		GaussDB的行为：
		m_db=# SELECT greatest((SELECT * FROM (SELECT DISTINCT c2/1.61 FROM t_time) t4), 1.00000000000000000);
		greatest
		---------- 	
		39144.72670807453416149
		(1 row)
		另外，PBE与用户自定义变量一起使用的场景。如果满足上述条件，MySQL会按照30位精度输出；否则，MySQL会按照原数据精度输出，而GaussDB始终按照30位精度输出，例如：
		MySQL的行为：
		-- 满足上述条件：
		mysql> SET @var6=12.1234567891;
		Query OK, 0 rows affected (0.00 sec)
		mysql> PREPARE p1 FROM "SELECT * FROM (SELECT @var6) t";
		Query OK, 0 rows affected (0.00 sec)
		Statement prepared
		mysql> EXECUTE p1;
		+----------
		| @var6 |
		+----------
		|
		12.123456789100000000000000000000 |
		+----------
		1 row in set (0.00 sec)
		-- 不满足上述条件：
		mysql> PREPARE p1 FROM "SELECT * FROM (SELECT @var6 FROM (SELECT 1) v1) t";
		Query OK, 0 rows affected (0.00 sec)
		Statement prepared
		mysql> EXECUTE p1;
		+----------
		| @var6 |
		+----------
		| 12.1234567891 |
		+----------
		1 row in set (0.00 sec)
		GaussDB的行为：
		-- 满足上述条件：
		m_db=# SET @var6=12.1234567891;
		SET
		m_db=# PREPARE p1 FROM "SELECT * FROM (SELECT @var6) t";
		PREPARE
		m_db=# EXECUTE p1;
		@var6
		---------- 	

概述	详细语法说明	差异
		12.12345678910000000000000000000 00 (1 row) -- 不满足上述条件： m_db=# PREPARE p1 FROM "SELECT * FROM (SELECT @var6 FROM (SELECT 1) v1) t"; PREPARE m_db=# EXECUTE p1; @var6 ---------- 12.12345678910000000000000000000 00 (1 row)

SELECT后跟行表达式	SELECT	MySQL不支持SELECT后跟行表达式，GaussDB支持SELECT后跟行表达式。
		MySQL的行为：
		mysql> SELECT row(1,2);
		ERROR 1241 (21000): Operand should
		contain 1 column(s)
		GaussDB的行为：
		m_db=# SELECT row(1,2);
		row(1,2)
		---------- 	
		(1,2)
		(1 row)

概述	详细语法说明	差异

SELECT视图查询、子查询或UNION涉及NUMERIC转TIME/DATETIME进位差异	SELECT	SELECT部分场景下输出TIME/DATETIME类型结果与MySQL存在差异。
		差异场景：视图查询、子查询或UNION；涉及NUMERIC转TIME/DATETIME。
		差异行为：GaussDB的SELECT行为统一，NUMERIC转TIME/DATETIME类型时，只对最大精度位（6）作进位处理。MySQL的视图查询、子查询和UNION场景，对实际的结果精度位作进位处理。
		MySQL的行为：
		-- 简单查询，只对最大精度位6作进位，所以输出11:11:00.00002。
		mysql> SELECT maketime(11, 11, 2.2/time '08:30:23.01');
		+--+
		| maketime(11, 11, 2.2/time '08:30:23.01') |
		+--+
		| 11:11:00.00002 |
		+--+
		1 row in set (0.01 sec)
		
		-- 子查询，对实际的结果精度位作进位，所以输出11:11:00.00003。
		mysql> SELECT * FROM (SELECT maketime(11, 11, 2.2/time '08:30:23.01')) f1;
		+--+
		| maketime(11, 11, 2.2/time '08:30:23.01') |
		+--+
		| 11:11:00.00003 |
		+--+
		1 row in set (0.00 sec)
		GaussDB的行为：
		m_db=# SET m_format_behavior_compat_options='enable_precision_decimal';
		SET
		
		-- 简单查询，只对最大精度位6作进位，所以输出11:11:00.00002。
		m_db=# SELECT maketime(11, 11, 2.2/time '08:30:23.01');
		maketime
		---------------- 	
		11:11:00.00002
		(1 row)
		
		-- 子查询，也只对最大精度位6作进位，结果精度是5，所以输出。11:11:00.00002
		m_db=# SELECT * FROM (SELECT maketime(11, 11, 2.2/time '08:30:23.01')) f1;
		maketime
		---------- 	

概述	详细语法说明	差异
		11:11:00.00002 (1 row)

SELECT在数值类型和子查询日期与时间函数的运算处理差异	SELECT	SELECT在数值类型和子查询日期与时间函数的运算场景，GUC参数m_format_behavior_compat_options开启enable_precision_decimal选项，GaussDB会先将函数返回的日期与时间转换为数值类型，然后按照数值类型运算，结果也为数值类型。MySQL在子查询条件查询、分组查询等场景会截断日期与时间函数返回值。
		MySQL的行为：
		mysql> SELECT 1.5688 * (SELECT adddate('2020-10-20', interval 1 day) WHERE true GROUP BY 1 HAVING true);
		+----------+
		| 1.5688 * (SELECT adddate('2020-10-20', interval 1 day) WHERE true HAVING true) |
		+----------+
		|
		3168.976 |
		GaussDB的行为：
		m_db=# SELECT 1.5688 * (SELECT adddate('2020-10-20', interval 1 day) WHERE true GROUP BY 1 HAVING true);
		?column?
		---------- 	
		31691361.744799998
		(1 row)

概述	详细语法说明	差异

SELECT在嵌套子查询时unsigned类型差异	SELECT	SELECT在嵌套子查询时unsigned类型不会被覆盖，与MySQL 5.7存在差异。
		MySQL 5.7的行为：
		mysql> DROP TABLE IF EXISTS t1;
		Query OK, 0 rows affected (0.02 sec)
		
		mysql> CREATE TABLE t1 (
		    -> c10 real(10, 4) zerofill
		    ->);
		Query OK, 0 rows affected (0.03 sec)
		
		mysql> INSERT INTO t1 VALUES(123.45);
		Query OK, 1 row affected (0.00 sec)
		
		mysql> DESC t1;
		+-------+---------------------------------
		+------+-----+---------+-------+
		| Field | Type | Null |
		Key | Default | Extra |
		+-------+---------------------------------
		+------+-----+---------+-------+
		| c10 | double(10,4) unsigned zerofill |
		YES | | NULL | |
		+-------+---------------------------------
		+------+-----+---------+-------+
		1 row in set (0.01 sec)
		
		mysql> CREATE TABLE t1_sub_1 AS
		SELECT (SELECT * FROM t1);
		Query OK, 1 row affected (0.03 sec)
		Records: 1 Duplicates: 0 Warnings: 0
		
		mysql> DESC t1_sub_1;
		+--------------------+--------------+------
		+-----+---------+-------+
		| Field | Type | Null | Key |
		Default | Extra |
		+--------------------+--------------+------
		+-----+---------+-------+
		| (SELECT * FROM t1) | double(10,4) |
		YES | | NULL | |
		+--------------------+--------------+------
		+-----+---------+-------+
		1 row in set (0.00 sec)
		GaussDB的行为：
		test=# DROP TABLE IF EXISTS t1;
		DROP TABLE
		test=# CREATE TABLE t1 (
		test(# c10 real(10, 4) ZEROFILL
		test(#);
		CREATE TABLE
		test=# INSERT INTO t1 VALUES(123.45);
		INSERT 0 1
		test=# DESC t1;
		| Field | Type | Null | Key
		| Default | Extra
		-------+---------------------------------+------
		+-----+---------+-------
		| c10 | double(10,4) unsigned zerofill |
		YES | | |
		(1 row)
		test=# CREATE TABLE t1_sub_1 AS
		SELECT (SELECT * FROM t1);

概述	详细语法说明	差异
		INSERT 0 1 test=# DESC t1_sub_1; Field | Type | Null | Key | Default | Extra -------+----------+------+-----+---------+------- c10 | double(10,4) unsigned | YES | | | (1 row)

概述	详细语法说明	差异

SELECT FOR SHARE/FOR UPDATE/ LOCK IN SHARE MODE	SELECT	• GaussDB不支持FOR SHARE/FOR UPDATE/LOCK IN SHARE MODE子句和UNION/EXCEPT/DISTINCT/ GROUP BY/HAVING子句一起使用，MySQL 5.7部分支持（FOR SHARE/EXCEPT语法不支持），MySQL 8.0均支持。
		• 当将锁子句与LEFT/RIGHT [OUTER] JOIN子句连用时，LEFT JOIN不支持给右表上锁，RIGHT JOIN不支持给左表上锁；MySQL可以给JOIN两侧的表同时上锁。
		• MySQL不支持给同一张表指定多次锁子句；GaussDB支持给同一张表指定多次锁子句，实际生效按照最强的锁处理。
		-- 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)
		
		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)

概述	详细语法说明	差异

SELECT语法支持范围	SELECT	• GaussDB的HAVING必须且只能引用GROUP BY子句中的列或聚合函数中使用的列。MySQL支持对此行为的扩展，并允许HAVING引用列表中的SELECT列和外部子查询中的列。
		• 当查询表为空表时，GaussDB在使用指定WITH ROLLUP语句进行查询时，查询结果为一条空行数据，而MySQL查询结果为空。
		• GaussDB指定FROM子句中的表别名时，支持带字段名称。MySQL 5.7不支持指定表别名时带字段名称，MySQL 8.0仅支持给子查询指定表别名时带字段名称。
		-- 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 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 *

概述	详细语法说明	差异

		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 line 1
		mysql> SELECT * FROM (SELECT * FROM t1) t2(a, b);
		+------+------+
		| a | b |
		+------+------+
		| 1 | 2 |
		+------+------+
		1 row in set (0.00 sec)
		• 当查询语句不带FROM子句时，GaussDB支持带WHERE子句，与MySQL 8.0保持一致，MySQL 5.7不支持。
		-- 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)

概述	详细语法说明	差异

不携带ORDER BY子句的UNION、GROUP BY等语句在数据合并或聚合的时候，由于使用执行器算子存在差异，不保证输出数据顺序和MySQL顺序一致。	SELECT	以GROUP BY场景为例，使用hashagg算子时和原始顺序不同，建议需要保证数据顺序的场景添加ORDER BY子句。
		-- 数据初始化
		DROP TABLE IF EXISTS test;
		CREATE TABLE test(id INT);
		INSERT INTO test VALUES (1),(2),(3),(4),(5);
		-- GaussDB
		-- 不开精度传递，id顺序为(1 3 2 4 5)
		m_db=# SET m_format_behavior_compat_options= '';
		SET
		m_db=# SELECT /*+ use_hash_agg*/ id, pi() FROM test GROUP BY 1,2;
		id | pi
		----+-------------------
		1 | 3.141592653589793
		3 | 3.141592653589793
		2 | 3.141592653589793
		4 | 3.141592653589793
		5 | 3.141592653589793
		(5 rows)
		-- 打开精度传递，由于值变化导致顺序变化，id顺序为(5 4 2 3 1)
		m_db=# SET m_format_behavior_compat_options= 'enable_precision_decimal';
		SET
		m_db=# SELECT /*+ use_hash_agg*/ id, pi() FROM test GROUP BY 1,2;
		id | pi
		----+----------
		5 | 3.141593
		4 | 3.141593
		2 | 3.141593
		3 | 3.141593
		1 | 3.141593
		(5 rows)
		-- MySQL，id顺序为原始顺序
		mysql> SELECT id, pi() FROM test GROUP BY 1,2;
		+------+----------+
		| id | pi() |
		+------+----------+
		| 1 | 3.141593 |
		| 2 | 3.141593 |
		| 3 | 3.141593 |
		| 4 | 3.141593 |
		| 5 | 3.141593 |
		+------+----------+
		5 rows in set (0.00 sec)

概述	详细语法说明	差异
支持WITH AS语句	SELECT UPDATE DELETE	• GaussDB的WITH递归场景下：当subquery中非递归部分的列与subquery最终获得的结果列的类型、typmod、字符序不一致时，会产生语法错误，暂不支持该场景使用。 • GaussDB的WITH递归部分不支持聚合函数，窗口函数，FOR UPDATE/SHARE，LIMIT与OFFSET；MySQL支持FOR UPDATE/SHARE，MySQL 8.0.19以后的版本支持LIMIT与OFFSET。 • GaussDB的WITH递归部分支持DISTINCT与GROUP BY，MySQL不支持。 • WITH递归场景下：当WITH递归部分列生成的值比非递归部分更宽时，MySQL宽松模式数据截断，严格模式报错。GaussDB数据不截断，与MySQL数据长度加宽后的结果一致。 • GaussDB的WITH递归部分进行外连接时，与MySQL支持范围存在差异。 -- 例如以下差异： -- 数据初始化 DROP TABLE IF EXISTS t2; CREATE TABLE t2(c INT); INSERT INTO t2 VALUES (5); -- GaussDB，非递归部分使用UNION SELECT 1::bigint为了强转非递归部分的列类型。 m_db=# WITH RECURSIVE cte AS (SELECT 1 AS a UNION SELECT 1::bigint UNION SELECT a+1 FROM cte RIGHT JOIN t2 ON t2.c>cte.a WHERE cte.a<3) SELECT * FROM cte; ERROR: recursive reference to query "cte" must not appear within an outer join LINE 1: ...AS a UNION SELECT 1::bigint UNION SELECT a+1 FROM cte RIGHT ... ^ -- MySQL 8.0 mysql> WITH RECURSIVE cte AS (SELECT 1 AS a UNION SELECT a+1 FROM cte RIGHT JOIN t2 ON t2.c>cte.a WHERE cte.a<3) SELECT * FROM cte; +------+ | a | +------+

概述	详细语法说明	差异
		| 1 | | 2 | | 3 | +------+ 3 rows in set (0.00 sec) DROP TABLE IF EXISTS t2;

概述	详细语法说明	差异
INSERT ... ON DUPLICATE KEY UPDATE语法	INSERT	• GaussDB的ON DUPLICATE KEY UPDATE子句中的VALUES()不支持表名.列名格式，MySQL支持。 • INSERT ... query ON DUPLICATE KEY UPDATE语句中，query为子查询，且子查询为UNION查询或EXCEPT查询时，MySQL 5.7支持UPDATE子句引用子查询中的列名；MySQL 8.0不支持UPDATE子句引用子查询中的列名；GaussDB与MySQL 8.0保持一致，不支持UPDATE子句引用子查询中的列名。 • ON DUPLICATE KEY UPDATE子句更新多列时，MySQL前面UPDATE的结果会影响后面的结果，且允许同一列设置多次。GaussDB中，前面UPDATE的结果不会影响后面的结果，且不支持同一列设置多次，二者存在差异。设置GUC兼容性参数m_format_dev_version为's2'后，可保持与MySQL行为一致，既支持同一列设置多次，且引用更新后的结果。 • 插入数据违反唯一约束执行update操作时，GaussDB和MySQL返回的受影响行数存在差异。更新一条数据时，GaussDB返回1，MySQL返回2；如果将现有行更新为其当前值，GaussDB返回1，MySQL返回0。 • ON DUPLICATE KEY UPDATE子句更新自增列为NULL，且自增列含有NOT NULL约束时，GaussDB报错The null value in column xxx violates the not-null constraint.；MySQL执行成功，更新对应自增列为0。 • 对于表字段为VARCHAR类型，隐式转换为数值数据类型的场景，MySQL在字符串转换失败的部分比字符串完整长度

概述	详细语法说明	差异

		少2的情况下（如："AA"没有字符被转换成功，长度差为2-0=2；"4XY"的第一个字符转换成功，长度差为3-1=2），不会报错。此时MySQL在长度差不等于2的情况下，仍然会报错。GaussDB中对应场景字符串转换失败时会始终报错。
		m_db=# CREATE TABLE t1(a INT PRIMARY KEY, b VARCHAR(10));
		NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for table "t1"
		CREATE TABLE
		m_db=# INSERT INTO t1 values(1, 'A');
		INSERT 0 1
		m_db=# INSERT INTO t1 VALUES(1, 'X') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		ERROR: The double value 'X' is incorrect.
		CONTEXT: referenced column: b
		m_db=# INSERT INTO t1 VALUES(1, 'YY') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		ERROR: The double value 'YY' is incorrect.
		CONTEXT: referenced column: b
		m_db=# INSERT INTO t1 VALUES(1, 'ZZZ') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		ERROR: The double value 'ZZZ' is incorrect.
		CONTEXT: referenced column: b
		m_db=# DROP TABLE t1;
		DROP TABLE
		
		mysql> CREATE TABLE t1(a INT PRIMARY KEY, b VARCHAR(10));
		Query OK, 0 rows affected (0.00 sec)
		
		mysql> TRUNCATE TABLE t1;
		Query OK, 0 rows affected (0.01 sec)
		
		mysql> INSERT INTO t1 values(1, 'A');
		Query OK, 1 row affected (0.00 sec)
		
		mysql> INSERT INTO t1 VALUES(1, 'X') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		ERROR 1292 (22007): Truncated incorrect DOUBLE value: 'X'
		
		mysql> INSERT INTO t1 VALUES(1, 'YY') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		Query OK, 2 rows affected, 2 warnings (0.00 sec)
		
		mysql> INSERT INTO t1 VALUES(1, 'ZZZ') ON DUPLICATE KEY UPDATE b=values(a)+values(b);
		ERROR 1292 (22007): Truncated incorrect DOUBLE value: 'ZZZ'

概述	详细语法说明	差异
		mysql> DROP TABLE t1; Query OK, 0 rows affected (0.00 sec)

3.7.5 DCL

表 3-33 DCL 语法兼容介绍

概述	详细语法说明	差异
SET NAMES指定COLLATE子句	SET [SESSION | LOCAL] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};	GaussDB中SQL_ASCII库下暂不支持指定charset_name与数据库字符集不同。具体请参考《M-Compatibility开发指南》中“SQL参考 > SQL语法 > SQL语句 > S > SET ”章节。 不指定字符集时，MySQL会报错但GaussDB不报错。
支持DESCRIBE语句	{DESCRIBE | DESC} tbl_name [col_name | wild]	• 用户权限验证与MySQL存在差异。 – GaussDB中需要拥有指定表所在Schema的USAGE权限，同时还需要拥有指定表的任意表级权限或列级权限，仅显示拥有SELECT、INSERT、UPDATE、REFERENCES和COMMENT权限的列信息。 – MySQL中需要拥有指定表的任意表级权限或列级权限，仅显示拥有SELECT、INSERT、UPDATE、REFERENCES和COMMENT权限的列信息。 • 模糊匹配时涉及到字符串比较操作时，Field字段使用字符集utf8mb4、字符序utf8mb4_general_ci。

概述	详细语法说明	差异
SET设置用户变量	SET @var_name := expr	• MySQL用户变量名支持使用转义字符或双重引号转义，GaussDB用户变量名不支持。 单引号括起的变量名，变量名不能出现单引号，如@'''、@''''、@'\''不支持，解析时匹配不到'或解析报错，如： -- 解析报错 db_mysql=# SET @'''' = 1; ERROR: syntax error at or near "@" LINE 1: SET @'''' = 1; -- 解析时匹配不到' db_mysql=# SET @'\'' = 1; db_mysql'# 双引号括起的变量名，变量名不能出现双引号，如@"""、@"""""、@"\""不支持，解析时匹配不到"或解析报错，如： -- 解析报错 db_mysql=# SET @"""""" = 1; ERROR: syntax error at or near "@" LINE 1: SET @"""""" = 1; -- 解析时匹配不到" db_mysql=# SET @"\"" = 1; db_mysql"# 反引号括起的变量名，变量名不能出现反引号，如@```、@````、@`\``不支持，解析时匹配不到`或解析报错，如： -- 解析报错 db_mysql=# SET @```` = 1; ERROR: syntax error at or near "@" LINE 1: SET @```` = 1; -- 解析时匹配不到` db_mysql=# SET @`\`` = 1; db_mysql`# • 形如set @var_name1 = @var_name2 := @var_name3 = @var_name4 := expr; 连续赋值，MySQL支持，GaussDB不支持。 db_mysql=# set @a := @b := @c = @d := 1; ERROR: user_defined variables cannot be set, such as

概述	详细语法说明	差异
		@var_name := expr is not supported. ● expr在GaussDB中可以为聚集函数，在MySQL中不支持。
SET设置系统参数	SET [SESSION | @@SESSION. | @@ | LOCAL | @@LOCAL.] {config_parameter { TO | = } { expr | DEFAULT } | FROM CURRENT }};	● config_parameter为BOOLEAN类型系统参数时： – 参数值直接设置为字符串形式的'1'/'0'、'true'/'false'时，GaussDB数据库设置成功，MySQL设置失败。 – 参数值设置为子查询的查询结果，当查询结果为'true'/'false'、非整数类型1/0时，GaussDB数据库设置成功，MySQL设置失败；当查询结果为NULL时，GaussDB数据库设置失败，MySQL设置成功。

USE切换当前模式	USE schema_name	使用USE语句指定模式，当用户没有对应模式的USAGE权限时，MySQL产生报错，GaussDB会将当前模式指定为空。
		-- 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

3.7.6 其他语句

表 3-34 其他语法兼容介绍

概述	详细语法说明	差异
锁机制	锁机制	• GaussDB数据库锁机制只能在事务块中使用，MySQL无限制。 • MySQL获取read锁后，当前会话无法进行写操作，GaussDB数据库获取read锁后，当前会话可以进行写操作。 • MySQL给表上锁后，读取其他表报错，GaussDB数据库无限制。 • MySQL同一会话中获取同一个表的锁，会自动释放上一个锁，并提交事务，GaussDB数据库无该机制。 • GaussDB数据库中LOCK TABLE只能在一个事务块的内部有用，且无UNLOCK TABLE命令，锁总是在事务结束时释放。
PBE	PBE	• 重复创建同名的PREPARE语句，GaussDB数据库会报已经存在的错误，需要先删除已有statement，MySQL会覆盖旧的statement。 • GaussDB数据库和MySQL在SQL语句执行过程中对异常场景的报错阶段不同，例如解析层、执行层等；而PREPARE语句对预备语句只处理到解析层。因此PBE下对于异常场景，报错位置在PREPARE阶段还是EXECUTE阶段，GaussDB数据库和MySQL存在可能差异。
单行注释语法	单行注释语法	单行注释语法仅在设置参数m_format_behavior_compat_options包含'forbid_none_space_comment'项后注释行为与MySQL一致。

3.7.7 用户与权限

概述

GaussDB数据库用户与权限管理的行为请参见《开发指南》中的“数据库安全 > 用户及权限”章节。

用户与权限的语法说明请参见《M-Compatibility开发指南》中的“ SQL参考 > SQL语法 > SQL语句”章节。

差异说明

- 语法格式差异

  GaussDB数据库的授权语法请参见《M-Compatibility开发指南》中的“ SQL参考 > SQL语法 > SQL语句 > G > GRANT”章节。

  MySQL中的授权语法如下：

```
-- 全局级、数据库级、表级、存储过程级赋权语法
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} ...]

-- 用户代理赋权语法
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
}
```

- 赋权类型差异

  MySQL支持的赋权类型如下：

  表 3-35 MySQL 支持的赋权类型

权限类型	释义及权限级别
ALL [PRIVILEGES]	授予指定访问级别的所有权限，除了 **GRANT OPTION**和 **PROXY**。
ALTER	启用**ALTER TABLE**。级别：全局、数据库、表。
ALTER ROUTINE	允许更改或删除存储过程。级别：全局、数据库、例程。
CREATE	启用数据库和表创建。级别：全局、数据库、表。
CREATE ROUTINE	启用存储过程创建。级别：全局、数据库。
CREATE TABLESPACE	允许创建、更改或删除表空间和日志文件组。级别：全局。
CREATE TEMPORARY TABLES	启用**CREATE TEMPORARY TABLE**。 级别：全局、数据库。
CREATE USER	启用**CREATE USER**、 **DROP USER**、 **RENAME USER**和**REVOKE ALL PRIVILEGES**。级别：全局。
CREATE VIEW	允许创建或更改视图。级别：全局、数据库、表。
DELETE	启用**DELETE**. 级别：全局、数据库、表。
DROP	允许删除数据库、表和视图。级别：全局、数据库、表。
EVENT	启用定时任务。级别：全局、数据库。
EXECUTE	使用户能够执行存储过程。级别：全局、数据库、存储过程。
FILE	使用户能够使服务器读取或写入文件。级别：全局。
GRANT OPTION	允许向其他账户授予权限或从其他账户删除权限。级别：全局、数据库、表、存储过程、代理。
INDEX	允许创建或删除索引。级别：全局、数据库、表。

权限类型	释义及权限级别
INSERT	启用INSERT。级别：全局、数据库、表、列。
LOCK TABLES	在具有SELECT权限的表上启用LOCK TABLES 。级别：全局、数据库。
PROCESS	使用户能够通过SHOW PROCESSLIST查看所有正在运行的线程. 级别：全局。
PROXY	启用用户代理。级别：从用户到用户。
REFERENCES	启用外键创建。级别：全局、数据库、表、列。
RELOAD	启用FLUSH操作的使用。级别：全局。
REPLICATION CLIENT	使用户能够查询源服务器或副本服务器的位置。级别：全局。
REPLICATION SLAVE	允许副本从源读取二进制日志。级别：全局。
SELECT	启用使用SELECT。级别：全局、数据库、表、列。
SHOW DATABASES	启用SHOW DATABASES以显示所有数据库。级别：全局。
SHOW VIEW	启用SHOW CREATE VIEW。级别：全局、数据库、表。
SHUTDOWN	启用mysqladmin shutdown的使用。级别：全局。
SUPER	启用其他管理操作，例如 CHANGE MASTER TO、 KILL、 PURGE BINARY LOGS、 SET GLOBAL和mysqladmin debug命令。级别：全局。
TRIGGER	启用触发器操作。级别：全局、数据库、表。
UPDATE	启用UPDATE。 级别：全局、数据库、表、列。
USAGE	等价于“没有特权”。

GaussDB数据库以对象划分支持以下权限：

表 3-36 GaussDB 数据库支持的赋权类型

授权对象	支持授予的权限
数据库	CREATE、CONNECT、TEMPORARY、TEMP、ALTER、DROP、COMMENT
模式	CREATE、USAGE、ALTER、DROP、COMMENT

授权对象	支持授予的权限
表、视图	SELECT、INSERT、UPDATE、DELETE、TRUNCATE、REFERENCES、TRIGGER、ALTER、DROP、COMMENT、INDEX、VACUUM
列	SELECT、INSERT、UPDATE、REFERENCES、COMMENT
序列	SELECT、USAGE、UPDATE、ALTER、DROP、COMMENT

- MySQL通过'*.*'表示全局层级的授权对象；GaussDB通过'{DATABASE} db_name'表示数据库层级的授权对象。GaussDB的数据库层级对应MySQL的全局层级。
- MySQL通过'schema_name.*'表示数据库/模式层级的授权对象；GaussDB通过'{SCHEMA} schema_name'表示模式层级的授权对象。GaussDB的模式层级对应MySQL的数据库/模式层级。
- MySQL中用户名为两部分：用户名@主机名；GaussDB数据库当前仅支持用户名。
- MySQL支持在GRANT赋权语法中修改用户验证，安全连接，资源参数属性，即auth_option、tls_option和resource option；GaussDB数据库赋权语法中不支持以上特性，需使用CREATE USER、ALTER USER设置用户相关属性。
- MySQL支持用户代理赋权，GRANT PROXY ON主要用于对多个用户进行统一的权限管理。MySQL 5.7未提供角色机制，而在MySQL 8.0和GaussDB数据库中都提供了角色机制。角色能满足用户对于多个用户权限统一管控的目标，可以替代GRANT PROXY ON。
- GaussDB数据库拥有public的概念，所用用户都拥有public的权限，部分系统表、系统视图可供所有用户查询。用户可以对public所拥有的权限进行grant和revoke；MySQL中，新创建的用户只拥有全局的usage权限，这个权限很小，几乎为0，只有连接数据库和查询information_schema 数据库的权限。
- GaussDB数据库中，对象的所有者缺省具有该对象上的所有权限，出于安全考虑所有者可以舍弃部分权限，但ALTER、DROP、COMMENT、INDEX、VACUUM以及对象的可再授予权限属于所有者固有的权限，隐式拥有；MySQL中，没有owner的概念，即使用户创建了表，如果没赋予用户对应权限，那么用户也不能对其创建的表进行IUD等操作。
- 在MySQL中，USAGE实际上表示无权限，所用用户都拥有该权限，当执行revoke或grant usage时，实际上不会进行任何修改；在GaussDB数据库中，USAGE权限如下：
  - 对于模式，USAGE允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。
  - 对于序列，USAGE允许使用nextval函数。
- 在GaussDB数据库中，支持给用户设置管理员角色，包括系统管理员（SYSADMIN）、安全管理员（CREATEROLE）、审计管理员（AUDITADMIN）、监控管理员（MONADMIN）、运维管理员（OPRADMIN）、安全策略管理员（POLADMIN）。默认情况下拥有SYSADMIN属性的系统管理员，具备系统最高权限。三权分立后，系统管理员将不再具有CREATEROLE属性（安全管理员）和AUDITADMIN属性（审计管理员）能力，即不再拥有创建角色和用户的权限，也不再拥有查看和维护数据库审计日志的权限；在MySQL中，不支持该用户设置管理员角色，也没有三权分立相关设计。

- 在GaussDB数据库中，可以给用户赋予ANY权限，表示用户能够在非系统模式下拥有对应的权限，包括CREATE ANY TABLE、SELECT ANY TABLE、CREATE ANY INDEX等；在MySQL中，不支持ANY权限的赋予。
- MySQL中提供SHOW GRANTS查询用户权限；GaussDB数据库中，可以通过gsql客户端元命令'\l+'、'\dn+' 、'\dp'查询权限信息，也可以通过查询pg_namespace、pg_class、pg_attribute等系统表的权限相关字段查询权限信息。
- MySQL中数据库、表、列被删除时，相关的授权信息在系统表中依然保留，如果重新创建同名对象用户依然拥有权限；GaussDB数据库中当数据库、表、列被删除时，相关的授权信息会被删除，在重新创建同名对象后需要重新授权。
- MySQL在授予数据库层级的权限时，支持‘_’和‘%’对数据库名进行模糊匹配；GaussDB数据库不支持对象名模糊匹配，‘_’或‘%’等特殊字符被识别为普通字符。
- MySQL中，GRANT语句中指定用户不存在时默认会创建该账户（此特性已在MySQL 8.0中移除）；GaussDB数据库不支持给未创建用户赋权。

3.7.8 系统表和系统视图

表 3-37 GaussDB 数据库与 MySQL 的系统表或系统视图差异

系统表或系统视图	差异列	GaussDB数据库与MySQL的差异
information_schema.columns	generation_expression	该字段输出结果因涉及GaussDB数据库与MySQL的表达式的字符串拼接逻辑的不同而存在差异。
	data_type	该字段输出结果因涉及GaussDB数据库的数据类型format_type输出，目前未修改，与MySQL存在差异。
	column_type	该字段输出结果因涉及GaussDB数据库的数据类型format_type输出，目前未修改，与MySQL存在差异。
information_schema.tables	engine	GaussDB数据库中： ● ENGINE对齐information_schema.engines数据。 ● 部分系统表ENGINE为空。 ● 在默认为ASTORE表且未指定STORAGE_TYPE时ENGINE为空。
	version	GaussDB数据库中不支持该字段。
	row_format	GaussDB数据库中不支持该字段。
	avg_row_length	GaussDB数据库下表示使用数据文件除以所有元组数（包括活元组和死元组）的结果。表中没有元组，值为null。
	max_data_length	GaussDB数据库中不支持该字段。

系统表或系统视图	差异列	GaussDB数据库与MySQL的差异
	data_free	GaussDB数据库中表示死元组在总元组中的比例乘以数据文件大小。如果表中没有元组，则为null。
	check_time	GaussDB数据库中不支持该字段。
	create_time	在GaussDB数据库下，此字段与MySQL行为表现有差异，对于创建视图的情形MySQL中该字段置null，GaussDB数据库则显示实际的创建表时间。数据库自带的表，视图设置null。
	update_time	GaussDB数据库自带的表，视图设置null。
	table_collation	在GaussDB数据库下，此字段与MySQL行为表现有差异。如果表指定的是视图，则为null。如果指定的表在创建时，未使用COLLATE子句指定列的排序规则，则为null。
information_schema.statistics	collation	GaussDB数据库只有值A、D，不会是NULL。
	packed	GaussDB数据库中不支持该字段。
	sub_part	GaussDB数据库中不支持该字段。
	comment	GaussDB数据库中不支持该字段。
information_schema.partitions	subpartition_name	GaussDB数据库中，如果分区不是子分区，则为null。
	subpartition_ordinal_position	GaussDB数据库中，如果分区不是子分区，则为null。
	partition_method	GaussDB数据库中，分区策略。如果分区不是一级分区，则为null。 ● 'r'：范围分区。 ● 'i'：间隔分区。 ● 'l'：list分区。 ● 'h'：hash分区。
	subpartition_method	GaussDB数据库中，子分区策略。如果分区不是二级分区，则为null。 ● 'r'：范围分区。 ● 'i'：间隔分区。 ● 'l'：list分区。 ● 'h'：hash分区。
	partition_description	GaussDB数据库中，区分一级分区和二级分区。
	partition_expression	GaussDB数据库中不支持该字段。

系统表或系统视图	差异列	GaussDB数据库与MySQL的差异
	subpartition_expression	GaussDB数据库中不支持该字段。
	data_length	GaussDB数据库中不支持该字段。
	max_data_length	GaussDB数据库中不支持该字段。
	index_length	GaussDB数据库中不支持该字段。
	data_free	GaussDB数据库中不支持该字段。
	create_time	GaussDB数据库中不支持该字段。
	update_time	GaussDB数据库中不支持该字段。
	check_time	GaussDB数据库中不支持该字段。
	checksum	GaussDB数据库中不支持该字段。
	partition_comment	GaussDB数据库中不支持该字段。
	nodegroup	GaussDB数据库中不支持该字段。

说明

- 视图中对于整型的类型回显，不支持指定精度范围。如MySQL的bigint(1)，GaussDB数据库下对应的是bigint类型，MySQL中bigint(21) unsigned，在GaussDB数据库下对应的是bigint unsigned类型。
- MySQL中int类型，在GaussDB数据库中是integer类型。
- m_schema.columns_priv视图的Column_priv字段、m_schema.tables_priv视图的Table_priv,Column_priv字段、m_schema.procs_priv视图的Routine_type,Proc_priv字段、m_schema.proc视图的type,language,sql_data_access,is_deterministic,security_type,sql_mode字段、m_schema.func视图的type字段均不支持，在此版本中不予显示。
- 由于information_schema.tables、information_schema.statistics、information_schema.partitions中部分字段基于统计信息获取，查看前请先执行ANALYZE，更新统计信息后再查看（如果数据库中更新数据，建议延迟执行ANALYZE）。
- information_schema.statistics包含的索引列需要是创建索引中索引列是完整的表列，如果索引列是表达式，则不在这个视图中。
- information_schema.partitions中一级分区和二级分区分开呈现。
- 视图中的grantee字段，MySQL的格式是'*user_name*'@'*host_name*'，在GaussDB数据库中是被授予权限的用户或角色的名称。
- 视图中的host字段，在GaussDB数据库中返回当前节点的hostname。
- 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、m_schema.procs_priv 在MySQL下需要授权后才能查看视图内容，GaussDB数据库可以根据默认权限查看到对应的内容。如对于表t1，在MySQL下需要先对t1给对应的用户授权，才能在权限视图中看到对应的权限信息，GaussDB数据库下则可以直接在视图中看到t1表相关的权限信息。
- m_schema中的系统视图，但在MySQL是系统表。
- information_schema.views中的VIEW_DEFINITION以及information_schema.routines中的ROUTINE_DEFINITION不做字符序控制。
- 《M-Compatibility开发指南》中“Schema”章节所列举的字符类型的视图字段，字符集使用utf8mb4，字符序使用utf8mb4_bin或utf8mb4_general_ci，字符序优先级为《M-Compatibility开发指南》中“SQL参考 > 字符集与字符序 > 字符集和字符序合并规则”所描述的“支持字符序的数据类型的列”的优先级，和MySQL存在差异。

3.8 驱动

GaussDB中，驱动PBE接口通过文本模式传参时，如果兼容性参数m_format_behavior_compat_options中不包含disable_zero_chars_conversion选项，服务端会将参数中的“\0”字符替换成空格，与MySQL行为存在差异。如果兼容性参数m_format_behavior_compat_options中包含disable_zero_chars_conversion选项，则禁止将“\0”字符转换成空格，与MySQL行为一致。

3.8.1 ODBC

3.8.1.1 ODBC 接口参考

获取参数描述信息

SQLDescribeParam接口是ODBC API中的一个函数，用于获取与预处理SQL语句（如调用SQLPrepare）相关参数的描述信息。它可以返回参数的类型、大小、是否允许NULL值等元数据，这对于动态构建SQL语句和绑定参数非常有用。

原型

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

表 3-38 SQLDescribeParam 参数说明

参数名	参数说明	差异
StatementHandle	语句句柄。	-
ParameterNumber	参数序号，起始为1，依次递增。	-
DataTypePtr	指向返回参数数据类型的指针。	MySQL ODBC对于任意类型均返回SQL_VARCHAR。 GaussDB ODBC的会根据内核返回的不同类型判断返回给应用相应的DataType类型。
ParameterSizePtr	指向返回参数大小的指针。	MySQL ODBC若允许ODBC驱动程序使用更大的数据包进行数据传输，则返回24M，否则返回255。 GaussDB ODBC根据实际类型返回参数大小。
DecimalDigitsPtr	指向返回参数十进制位数的指针。	-
NullablePtr	指向返回参数是否允许NULL值的指针。	MySQL ODBC直接返回SQL_NULLABLE_UNKNOWN。 GaussDB ODBC直接返回SQL_NULLABLE。