

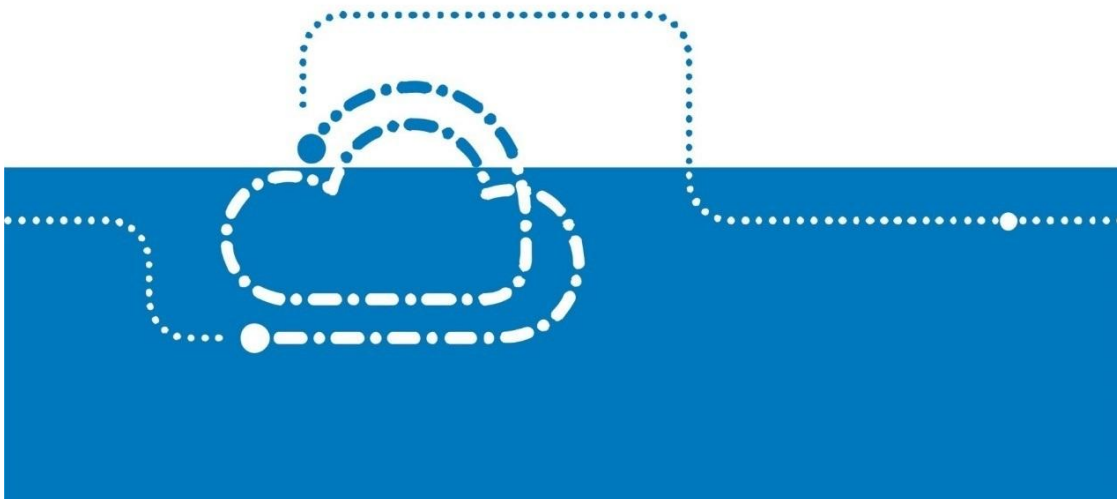


金篆信科

ZXCLOUD GoldenDB



分布式数据库系统

SQL规范



产品版本：V6.1.02.01

法律声明

若接收金篆信科股份有限公司（以下称为"金篆信科"）的此份文档，即表示您已同意以下条款。若不同意以下条款，请停止使用本文档。本文档版权所有金篆信科股份有限公司。保留任何未在本文档中明示授予的权利。文档中涉及金篆信科的专有信息。未经金篆信科事先书面许可，任何单位和个人不得复制、传递、分发、使用和泄漏该文档以及该文档包含的任何图片、表格、数据及其他信息。和是金篆信科的注册商标。金篆信科产品的名称和标志是金篆信科的商标或注册商标。在本文档中提及的其他产品或公司名称可能是其各自所有者的商标或注册商标。在未经金篆信科或第三方权利人事先书面同意的情况下，阅读本文档并不表示以默示、不可反言或其他方式授予阅读者任何使用本文档中出现的任何标记的权利。本产品符合有关环境保护和人身安全方面的设计要求，产品的存放、使用和弃置应遵照产品手册、相关合同或相关国法律、法规的要求进行。本文档按"现状"和"仅此状态"提供。本文档中的信息随着金篆信科产品和技术的进步将不断更新，金篆信科不再通知此类信息的更新。

金篆信科股份有限公司

地址：北京市北京经济技术开发区科谷一街 10 号院 8 号楼

邮编：100176

网站：

邮箱：

- 产品版本: V6.1.02.01
- 1 文档概述
 - 1.1 文档用户
 - 1.2 术语
- 2 分布式 SQL 语法通用说明
- 3 DATA TYPES
 - 3.1 GoldenDB 支持数据类型列表
 - 3.2 使用说明
 - 3.3 GoldenDB 支持数据类型举例
 - 3.3.1 CLOB、BLOB 大对象插入
 - 3.3.2 JSON
 - 3.3.2.1 建表
 - 3.3.2.2 插入数据
 - 3.3.2.3 GoldenDB 计算节点支持的 JSON 函数
 - 3.3.2.3.1 计算节点支持的 JSON 函数列表
 - JSON_VALID JSON 值是否有效 - 3.3.2.3.2 支持的 JSON 函数示例 - 3.3.2.3.3 JSON 函数数据类型说明
- 4 OPERATORS
 - XOR 异或
- 5 FUNCTIONS
 - 5.1 GoldenDB 支持函数列表
 - 5.2 函数解析模式说明
 - 5.3 复杂函数说明
- 6 PSEUDO COLUMNS
 - 6.1 SEQUENCE
 - 6.1.1 语法说明
 - 6.1.2 用例
 - 6.1.3 使用说明
 - 6.1.4 限制说明
 - 6.1.5 最佳实践
 - 6.2 伪列 ROWNUM
 - 6.2.1 语法说明
 - 6.2.2 用例
 - 6.2.3 限制说明

- 6.2.4 最佳实践
- 7 DDL
 - 7.1 CREATE DATABASE
 - 7.1.1 语法说明
 - 7.1.2 用例
 - 7.1.3 使用说明
 - 7.1.4 最佳实践
 - 7.2 DROP DATABASE
 - 7.2.1 语法说明
 - 7.2.2 用例
 - 7.2.3 使用说明
 - 7.2.4 最佳实践
 - 7.3 ALTER DATABASE
 - 7.3.1 语法说明
 - 7.3.2 用例
 - 7.3.3 使用说明
 - 7.3.4 最佳实践
 - 7.4 CREATE TABLE
 - 7.4.1 语法说明
 - 7.4.1.1 distributed_option 说明
 - 7.4.1.2 数据分发策略原则
 - 7.4.1.3 分发选项说明
 - GroupNo 存储安全组定义
 - 7.4.1.4 不同分区不同分片表
 - 7.4.1.5 多级分片表（该功能为企业版特性）
 - 7.4.1.6 使用 DEFAULT 值的 LIST 表
 - 7.4.1.7 单个分片下支持多个区间段的 RANGE 表
 - 7.4.1.8 自定义 HASH 算法
 - 7.4.1.9 无主键表
 - 7.4.2 用例
 - 7.4.3 使用说明
 - 7.4.3.1 主键的限制
 - 7.4.3.2 分发键限制
 - 7.4.3.3 表限制
 - 7.4.3.4 数据类型限制
 - 7.4.3.5 自增列相关限制
 - 7.4.3.6 字段属性限制

- 7.4.3.7 自增列有关配置项
 - 7.4.4 最佳实践
 - 7.4.4.1 表级别定义原则
 - 7.4.4.2 表的设计原则
- 7.5 DROP TABLE
 - 7.5.1 语法说明
 - 7.5.2 用例
 - 7.5.3 使用说明
 - 7.5.4 最佳实践
- 7.6 ALTER TABLE
 - 7.6.1 语法说明
 - 7.6.2 用例
 - 7.6.3 不同分片不同分区管理
 - 7.6.4 使用说明
 - 7.6.5 最佳实践
- 7.7 TRUNCATE TABLE
 - 7.7.1 语法说明
 - 7.7.2 用例
 - 7.7.3 使用说明
 - 7.7.4 最佳实践
- 7.8 DROP INDEX
 - 7.8.1 语法说明
 - 7.8.2 用例
 - 7.8.3 使用说明
 - 7.8.4 最佳实践
- 7.9 CREATE INDEX
 - 7.9.1 语法说明
 - 7.9.2 用例
 - 7.9.3 使用说明
 - 7.9.4 最佳实践
- 7.10 COPY TABLE
 - 7.10.1 语法说明
 - 7.10.2 使用说明
 - 7.10.3 最佳实践
- 7.11 CHECK TABLE
 - 7.11.1 语法说明
 - 7.11.2 用例
 - 7.11.3 使用说明

- 7.11.4 最佳实践
- 8 DML
 - 8.1 INSERT
 - 8.1.1 语法说明
 - 8.1.2 用例
 - 8.1.3 使用说明
 - 8.1.4 最佳实践
 - 8.2 DELETE
 - 8.2.1 语法说明
 - 8.2.2 用例
 - 8.2.3 使用说明
 - 8.2.4 限制说明
 - 8.2.5 最佳实践
 - 8.3 UPDATE
 - 8.3.1 语法说明
 - 8.3.2 用例
 - 8.3.3 使用说明
 - 8.3.4 最佳实践
 - 8.3.4.1 关联方式的多表优化
 - 8.3.4.2 非关联方式的多表优化
 - 8.3.4.3 其他优化
 - 8.4 REPLACE
 - 8.4.1 语法说明
 - 8.4.2 用例
 - 8.4.3 使用说明
 - 8.4.4 最佳实践
 - 8.5 MERGE INTO
 - 8.5.1 语法说明
 - 8.5.2 用例
 - 8.5.3 限制说明
 - 8.5.4 最佳实践
- 9 DQL
 - 9.1 SELECT
 - 9.1.1 语法说明
 - 9.1.2 用例
 - 9.1.3 使用说明
 - 9.1.3.1 SELECT LIST
 - 9.1.3.2 FUNCTION

- 9.1.3.3 WHERE
 - 9.1.3.4 data type
 - 9.1.3.5 多表
 - 9.1.3.6 select ... readslave
 - 9.1.3.7 其它
 - 9.1.4 最佳实践
 - 9.1.4.1 子查询的优化
 - 9.1.4.2 JOIN 的优化
 - 9.1.4.3 分组聚合的优化
 - 9.1.4.4 分页的优化
 - 9.2 UNION
 - 9.2.1 语法说明
 - 9.2.2 用例
 - 9.2.3 使用说明
 - 9.2.4 最佳实践
- 10 VIEW
 - 10.1 语法说明
 - 10.1.1 CREATA VIEW
 - 10.1.2 ALTER VIEW
 - 10.1.3 DROP VIEW
 - 10.2 用例
 - 10.3 使用说明
 - 10.4 最佳实践
- 11 TRIGGER
 - 11.1 语法说明
 - 11.2 用例
 - 11.3 使用说明
 - 11.4 最佳实践
- 12 TRANSACTION AND LOCKING STATEMENTS
 - 12.1 TRANSACTION
 - 12.1.1 语法说明
 - 12.1.2 使用说明
 - 12.1.3 最佳实践
 - 12.2 SAVEPOINT
 - 12.2.1 语法说明
 - 12.2.1.1 SAVEPOINT
 - 12.2.1.2 ROLLBACK TO SAVEPOINT
 - 12.2.1.3 RELEASE SAVEPOINT

- 12.2.2 用例
 - 12.2.3 使用说明
 - 12.2.4 最佳实践
- 13 PREPARE
 - 13.1 语法说明
 - 13.2 用例
 - 13.3 使用说明
 - 13.4 最佳实践
- 14 SYNONYM
 - 14.1 语法说明
 - 14.1.1 CREATE SYNONYM
 - 14.1.1 DROP SYNONYM
 - 14.2 用例
 - 14.3 使用说明
 - 14.4 最佳实践
- 15 数据库管理语句
 - 15.1 语法说明
 - 15.1.1 账户管理语法
 - 15.1.2 账户管理用例
 - 15.1.3 SHOW 语法
 - 15.1.4 SHOW 用例
 - 15.1.5 SET 命令
 - 15.1.6 SET 用例
 - 15.1.7 DESC 命令
 - 15.1.8 DESC 用例
 - 15.1.9 碎片整理
 - 15.2 使用说明
 - 15.3 最佳实践
- 16 实用语法
 - 16.1 EXPLAIN
 - 16.1.1 语法说明
 - 16.1.2 用例
 - 16.1.3 使用说明
 - 16.1.4 最佳实践
 - 16.2 USE
 - 16.2.1 语法说明
 - 16.2.2 用例

- 16.2.3 使用说明
 - 16.2.4 最佳实践
 - 16.3 外部流水号设置
 - 16.3.1 语法说明
 - 16.3.2 用例
 - 16.3.3 使用说明
 - 16.3.4 最佳实践
 - 16.4 IGNORE_SPACE
 - 16.4.1 语法说明
 - 16.4.2 用例
 - 16.4.3 最佳实践
 - 16.5 PURGE
 - 16.5.1 语法说明
 - 16.5.1.1 PURGE TABLE
 - 16.5.1.2 PURGE RECYCLEBIN
 - 16.5.2 用例
 - 16.5.3 限制说明
 - 16.5.4 最佳实践
 - 16.6 FLASHBACK
 - 16.6.1 语法说明
 - 16.6.1.1 FLASHBACK TABLE
 - 16.6.2 用例
 - 16.6.3 限制说明
 - 16.6.4 最佳实践
 - 16.7 INITCAP
 - 16.7.1 语法说明
 - 16.7.2 用例
 - 16.7.3 限制说明
 - 16.7.4 最佳实践
- 17 USER-DEFINED FUNCTION
 - 17.1 语法说明
 - 17.1.1 CREATE FUNCTION
 - 17.1.2 DROP FUNCTION
 - 17.2 用例
 - 17.3 使用说明
 - 17.4 最佳实践
- 18 AP 节点(MPP)的使用
 - 18.1 hint 注释控制是否走 MPP

- 18.2 代价估算自适应配置
 - 18.3 MPP 的限制
- 19 存储过程
 - 19.1 单机存储过程
 - 19.1.1 语法说明
 - 19.1.1.1 CREATE PROCEDURE
 - 19.1.1.2 DROP PROCEDURE
 - 19.1.1.3 CALL PROCEDURE
 - 19.1.2 用例
 - 19.1.3 使用说明
 - 19.1.4 最佳实践
 - 19.2 分布式存储过程
 - 19.2.1 语法说明
 - 19.2.1.1 存储过程创建
 - 19.2.1.2 存储过程执行
 - 19.2.1.3 存储过程修改
 - 19.2.1.4 存储过程删除
 - 19.2.1.5 存储过程查看
 - 19.2.1.6 存储过程中支持的 SQL 列表
 - 19.2.1.7 存储过程中明确不支持的 SQL 列表
 - 19.2.2 使用说明
- 20 XA 事务语法
 - 20.1 语法说明
 - 20.1.1 XA START/BEGIN
 - 20.1.2 XA END
 - 20.1.3 XA PREPARE
 - 20.1.4 XA COMMIT
 - 20.1.5 XA ROLLBACK
 - 20.1.6 XA RECOVER
 - 20.2 用例
 - 20.3 使用说明
- 21 保留字
 - WINDOW
- 22 函数限制
 - 22.1 随机值函数和时间函数
 - 22.1.1 复制表
 - 22.1.2 分发表

1 文档概述

1.1 文档用户

本规范文档介绍分布式数据库支持的语法、分布式下的应用限制、分布式下的最佳应用方式，用于帮助业务开发人员了解如何基于分布式数据库编写 SQL 开发业务程序。

1.2 术语

本规范文档涉及术语以及说明参见下表。

表 1-1 术语说明表

术语	英文	说明
脏读	UR	查询时，读取的数据中允许包含 脏数据，或者用于查询时不存在其他事务修改的情况。
强一致性读	CR	查询时，读取的数据是已提交的。
单节点写	SW	不 考虑分布式事务下的对部分已提交事务的数据的修改。 下面两种情况使用 SW： <ul style="list-style-type: none">修改的数据不会在两个及以上事务中同时修改。

- 涉及本数据的所有事务都只会修改单条记录。

强一致 性写 CW 需要考虑分布式事务下的对部分已提交事务的数据的修改。

2 分布式 SQL 语法通用说明

- **CONSISTENCY_OPTION**

数据一致性级别：脏读(UR)、强一致性读(CR)、单节点写(SW)、强一致性写(CW)；默认值为强一致性读(CR)和强一致性写(CW)。

- **STORAGEDB**

表示 SQL 语句执行所涉及到的 DB 节点，当指定了 STORAGEDB 时，计算节点以指定的 STORAGEDB 为准，而不再根据表的定义来判断。

- **SAMEDB**

该提示可以出现在 START TRANSACTION 或事务中的任一条 SQL 的末尾。

含义为从该条语句开始，后续语句涉及下发的 DB 节点均和该语句保持一致，该提示可以提升数据库的处理性能。

- **table_name 说明**

本规范中 table_name 可以为 db_name.table_name，从 INSIGHT 界面发起的 DDL 操作要求必须携带库名。

3 DATA TYPES

3.1 GoldenDB 支持数据类型列表

表 3-1 GoldenDB 支持的数据类型列表

序号	GoldenDB 数据类型	说明
数值型	INT	4 字节整数，SIGNED: -2147483648 -- 2147483647 UNSIGNED: 0 - 4294967295
	SMALLINT	2 字节整数，SIGNED: -32768 -- 32767 UNSIGNED: 0 - 65535
	BIGINT	8 字节整数，SIGNED: -9223372036854775808 -- 9223372036854775807 UNSIGNED: 0 - 18446744073709551615

序号	GoldenDB 数据类型	说明
日期型	TINYINT	1 字节整数, SIGNED: -128 -- 127 UNSIGNED: 0 - 255
	MEDIUMINT	3 字节整数, SIGNED: -8388608 -- 8388607 UNSIGNED: 0 - 16777215
	INTEGER	4 字节整数, SIGNED: -2147483648 -- 2147483647 UNSIGNED: 0 - 4294967295
	FLOAT(5)	4 字节单精度浮点数
	DOUBLE	8 字节双精度浮点数
	DECIMAL(n,s)	数值型, n 为数值总长度, 范围 1-65, s 为小数点后长度, 范围 0-30, s 必须不大于 n
	NUMERIC(n,s)	同 DECIMAL
	NUMBER(n,s)	同 DECIMAL
	DEC(n,s)	同 DECIMAL
	DATE	日期, 范围: 1000-01-01 -- 9999-12-31
	TIMESTAMP	UTC 日期和时间, 范围: 1970-01-01 00:00:01.000000 -- 2038-01-19 03:14:07.999999
	DATETIME	日期和时间, 范围: 1000-01-01 00:00:01.000000 -- 9999-12-31 23:59:59.999999
	TIME	时间, 范围是'-838:59:59'到'838:59:59'
	YEAR	两位或四位格式的年。默认是四位格式。在四位格式中, 允许的值是 1901 到 2155 和 0000。在两位格式中, 允许的值是 70 到 69, 表示从 1970 年到 2069 年
字符型	VARCHAR(10)	变长字符型,最多 65535 个字符
	CHAR(10)	定长字符型,最多 255 个字符
	BLOB	最大长度为 65,535(2 ¹⁶ --1)字节的 BLOB 列
	LOB	二进制形式的极大文本数据, 0-4294967295 字节
	TINYBLOB	不超过 255 个字符的二进制字符串
	MEDIUMBLOB	二进制形式的中等长度文本数据, 0-16777215 字节
	CLOB	二进制形式的长文本数据, 0-65535 字节
	BINARY	类似 CHAR 的二进制存储, 特点是插入定长不足补 0
	VARBINARY	类似 VARCHAR 的变长二进制存储, 特点是定长不补 0

序号	GoldenDB 数据类型	说明
	LONGTEXT	极大文本数据，0-4294967295 字节
	TINYTEXT	短文本字符串，0-255 字节
	MEDIUMTEXT	中等长度文本数据，0-16777215 字节
	ENUM	枚举，最大可达 65535 个不同的枚举值
	TEXT	可变长度，0-65535 字节
	SET	多选字符串数据类型，最多可以包含 64 项元素
	VARCHAR2	支持 VARCHAR2(num)、VARCHAR2(num BYTE)、VARCHAR2(num CHAR) 三种用法，前两种用法等价；存储过程参数类型为 VARCHAR2 且未指定长度时，默认长度为 4000 字节；
	JSON	-

3.2 使用说明

不支持 BINARY、VARBINARY、BLOB、TEXT 类型字段的关联、运算等，计算节点只支持透传。不建议使用 float、double 类型进行比较操作，操作不限于 group by、distinct、on 等值条件、where 过滤。

3.3 GoldenDB 支持数据类型举例

非常规类型举例说明。

3.3.1 CLOB、BLOB 大对象插入

- 创建库表：

```
DROP DATABASE IF EXISTS poc_test;
```

```
CREATE DATABASE IF NOT EXISTS poc_test;
```

```
CREATE TABLE poc_test.t1 (a INT PRIMARY KEY ,b BLOB,c CLOB) DISTRIBUTED BY  
DUPLICATE(g1,g2);
```

- 插入数据

```
INSERT INTO poc_test.t1 VALUES
```

```
(1,LOAD_FILE('/home/zxdb/bin/binlog_apply_ro  
llback.py'),LOAD_FILE('/home/zxdb/bin/binlog_apply_rollback.py'));
```

- 导出图片

```
SELECT b FROM poc_test.t1 INTO dumpfile '/home/zxdb';
```

3.3.2 JSON

3.3.2.1 建表

- 建表语句：

```
DROP DATABASE IF EXISTS poc_test;
```

```
CREATE DATABASE IF NOT EXISTS poc_test;
```

```
CREATE TABLE poc_test.json_tab
```

```
(
```

```
id INT PRIMARY KEY,
```

```
json_info JSON COMMENT 'json 数据',
```

```
json_id INT GENERATED ALWAYS AS (json_info -> '$.id') COMMENT 'json 数据的虚拟字段',
```

```
json_name VARCHAR(5) GENERATED ALWAYS AS (json_info -> '$.name'),
```

```
INDEX json_info_id_idx (json_id)
```

```
)DISTRIBUTED BY HASH(id) (g1,g2);
```

- 建表带 JSON 数据类型说明：

1. 支持 JSON 虚拟列；
2. 支持 JSON 虚拟列做索引；
3. ALTER TABLE 支持增加、修改、删除 JSON 类型字段；
4. ALTER TABLE 支持增加、修改、删除 JSON 虚拟列。
5. JSON 类型字段不能做分发键、主键、[全局]索引；
6. JSON 虚拟列不能做分发键。

3.3.2.2 插入数据

插入数据语句：

```
INSERT INTO poc_test.json_tab(id,json_info) VALUES(1,"abc");
```

```
INSERT INTO poc_test.json_tab(id,json_info) VALUES(2,'123');
```

```
INSERT INTO poc_test.json_tab(id,json_info) VALUES(3,{'id': 1, "name": "张三"});
```

```
INSERT INTO poc_test.json_tab(id,json_info) VALUES(4,['1, 2, 3']);
```

```
INSERT INTO poc_test.json_tab(id,json_info) VALUES(5,{'id': 1, "name": "张三",
"age": 18, "sister": [{"name": "张大姐", "age": 30}, {"name": "张二姐", "age": 20}]});

INSERT INTO poc_test.json_tab(id,json_info) VALUES(6,JSON_OBJECT('id', 2, 'name',
'李四', 'age', 18, 'sister', JSON_ARRAY(JSON_OBJECT('name', '李大姐', 'age', 28),
JSON_OBJECT('name', '李二姐', 'age', 25))));
```

3.3.2.3 GoldenDB 计算节点支持的JSON 函数

3.3.2.3.1 计算节点支持的JSON 函数列表

表 3-2 计算节点支持的JSON 函数列表

JSON 函数	说明
JSON_ARRAY	创建 JSON 数组
JSON_ARRAY_APPEND	添加数据到指定 JSON 数组结尾
JSON_ARRAY_INSERT	往 JSON 数组指定位置插入数据
JSON_CONTAINS	JSON 文档是否在路径中包含特定对象
JSON_CONTAINS_PATH	JSON 文档是否在路径中包含任何数据
JSON_DEPTH	JSON 文档的最大深度
JSON_EXTRACT, ->	返回 JSON 文档指定路径的数据
JSON_INSERT	将数据插入 JSON 文档，添加不存在的值但不替换已存在的值
JSON_KEYS	返回有 JSON 对象的键名构成的数组
JSON_LENGTH	JSON 文档的元素数量
JSON_MERGE	合并 JSON 文档，保留重复的键。
JSON_MERGE_PATCH	合并 JSON 文档，替换重复键的值。
JSON_MERGE_PRESERVE	同 JSON_MERGE
JSON_OBJECT	创建 JSON 对象
JSON_QUOTE	引用 JSON 文档
JSON_REMOVE	从 JSON 文档中删除数据
JSON_REPLACE	替换 JSON 文档中的值
JSON_SET	将数据插入 JSON 文档，添加不存在的值并替换已存在的值
JSON_SEARCH	返回指定字符串在 JSON 文件中的路径
JSON_TYPE	JSON 值类型
JSON_UNQUOTE	取消引用 JSON 值

JSON 函数
JSON_VALID

说明
JSON 值是否有效

3.3.2.3.2 支持的 JSON 函数示例

- JSON_ARRAY([val[, val] ...])

```
SELECT JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME());
```

```
+-----+
| JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME()) |
```

```
+-----+
| [1, "abc", null, true, "20:29:16.000000"] |
```

- JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...)

```
SELECT JSON_ARRAY_APPEND(['a', ['b', 'c'], 'd'], '$[1]', 1);
```

```
+-----+
| JSON_ARRAY_APPEND(['a', ['b', 'c'], 'd'], '$[1]', 1) |
```

```
+-----+
| ["a", ["b", "c", 1], "d"] |
```

- JSON_ARRAY_INSERT(json_doc, path, val[, path, val] ...)

```
SELECT JSON_ARRAY_INSERT(['a', {'b': [1, 2]}, [3, 4]], '$[1]', 'x');
```

```
+-----+
| JSON_ARRAY_INSERT(['a', {'b': [1, 2]}, [3, 4]], '$[1]', 'x') |
```

```
+-----+
| ["a", "x", {"b": [1, 2]}, [3, 4]] |
```

```
+-----  
-----+
```

- JSON_CONTAINS(target, candidate[, path])
SELECT JSON_CONTAINS('{ "a": 1, "b": 2, "c": {"d": 4}}', '1', '\$.a');

```
+-----  
-----+
```

```
| JSON_CONTAINS('{ "a": 1, "b": 2, "c": {"d": 4}}', '1', '$.a') |
```

```
+-----  
-----+
```

```
| 1 |
```

```
+-----  
-----+
```

- JSON_CONTAINS_PATH(json_doc, one_or_all, path[, path] ...)
SELECT JSON_CONTAINS_PATH('{ "a": 1, "b": 2, "c": {"d": 4}}', 'one', '\$.a', '\$.e');

```
+-----  
-----+
```

```
| JSON_CONTAINS_PATH('{ "a": 1, "b": 2, "c": {"d": 4}}', 'one', '$.a', '$.e') |
```

```
+-----  
-----+
```

```
| 1 |
```

```
+-----  
-----+
```

- JSON_DEPTH(json_doc)
SELECT JSON_DEPTH('[10, {"a": 20}]');

```
+-----+
```

```
| JSON_DEPTH('[10, {"a": 20}]') |
```

```
+-----+
```

```
| 3 |
```

```
+-----++
```

- JSON_EXTRACT(json_doc, path[, path] ...)

SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '\$[1]');

+-----

-----+

| JSON_EXTRACT('[10, 20, [30, 40]]', '\$[1]') |

+-----

-----+

| 20 |

+-----

-----+

- JSON_INSERT(json_doc, path, val[, path, val] ...)

SELECT JSON_INSERT('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]');

+-----

-----+

| JSON_INSERT('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]') |

+-----

-----+

| {"a": 1, "b": [2, 3], "c": "[true, false]"} |

+-----

-----+

- JSON_KEYS(json_doc[, path])

SELECT JSON_KEYS('{ "a": 1, "b": {"c": 30}}', '\$.b');

+-----

-----+

| JSON_KEYS('{ "a": 1, "b": {"c": 30}}', '\$.b') |

+-----

-----+

| ["c"] |

+-----

-----+

- JSON_LENGTH(json_doc[, path])

SELECT JSON_LENGTH('{ "a": 1, "b": {"c": 30}}', '\$.b');

```
+-----
-----+

| JSON_LENGTH('{ "a": 1, "b": { "c": 30 } }', '$.b') |
```

```
+-----
-----+

| 1 |
```

```
+-----
-----+
```

- JSON_MERGE(json_doc, json_doc[, json_doc] ...)

```
SELECT JSON_MERGE('[1, 2]', '[true, false]');
```

```
+----
-----+

| JSON_MERGE('[1, 2]', '[true, false]') |
```

```
+----
-----+
```

```
| [1, 2, true, false] |
```

```
+----
-----+
```

- JSON_MERGE_PATCH(json_doc, json_doc[, json_doc] ...)

```
SELECT JSON_MERGE_PATCH('{ "a": 1, "b": 2 }, { "a": 3, "c": 4 }');
```

```
+-----
-----+

| JSON_MERGE_PATCH('{ "a": 1, "b": 2 }, { "a": 3, "c": 4 }') |
```

```
+-----
-----+
```

```
| { "a": 3, "b": 2, "c": 4 } |
```

```
+-----
-----+
```

- JSON_MERGE_PRESERVE(json_doc, json_doc[, json_doc] ...)

```
SELECT JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }, { "a": 3, "c": 4 }');
```

```
+-----
```

```
-----+
| JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }', '{ "a": 3, "c": 4 }') |
```

```
+-----+
-----+
```

```
| { "a": [1, 3], "b": 2, "c": 4 } |
```

```
+-----+
-----+
```

```

    •   JSON_OBJECT([key, val[, key, val] ...])
SELECT JSON_OBJECT('id', 87, 'name', 'Tom');
```

```
+---
-----+
```

```
| JSON_OBJECT('id', 87, 'name', 'Tom') |
```

```
+---
-----+
```

```
| {"id": 87, "name": "Tom"} |
```

```
+---
-----+
```

```

    •   JSON_QUOTE(string)
SELECT JSON_QUOTE('[1, 2, 3]');
```

```
+-----+
```

```
| JSON_QUOTE('[1, 2, 3]') |
```

```
+-----+
```

```
| "[1, 2, 3]" |
```

```
+-----+
```

```

    •   JSON_REMOVE(json_doc, path[, path] ...)
SELECT JSON_REMOVE(['a', ['b', "c"], "d"], '$[1]');
```

```
+-----+
-----+
```

```
| JSON_REMOVE(['a', ['b', "c"], "d"], '$[1]') |
```

```
+-----
```

-----+

| ["a", "d"] |

+-----

-----+

- JSON_REPLACE(json_doc, path, val[, path, val] ...)
SELECT JSON_REPLACE('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]');

+-----

-----+

| JSON_REPLACE('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]') |

+-----

-----+

| {"a": 10, "b": [2, 3]} |

+-----

-----+

- JSON_SET(json_doc, path, val[, path, val] ...)
SELECT JSON_SET('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]');

+-----

-----+

| JSON_SET('{ "a": 1, "b": [2, 3]}', '\$.a', 10, '\$.c', '[true, false]') |

+-----

-----+

| {"a": 10, "b": [2, 3], "c": "[true, false]"} |

+-----

-----+

- JSON_SEARCH(json_doc, one_or_all, search_str[, escape_char[, path] > ...])
SELECT JSON_SEARCH(["abc", [{"k": "10"}, "def"], {"x": "abc"}, {"y": "bcd"}], 'all', '%a%') AS path;

+-----+

| path |

+-----+

| ["\$[0]", "\$[2].x"] |

```
+-----+
•    JSON_TYPE(json_val)
SELECT JSON_TYPE(JSON_EXTRACT('{\"a\": [10, true]}', '$.a[0]'));
+-----+
-----+
```

```
| JSON_TYPE(JSON_EXTRACT('{\"a\": [10, true]}', '$.a[0]')) |
+-----+
-----+
```

```
| INTEGER |
+-----+
-----+
```

```
•    JSON_UNQUOTE(json_val)
SELECT JSON_UNQUOTE(\"abc\");
+-----+
+-----+
```

```
| JSON_UNQUOTE(\"abc\") |
+-----+
+-----+
```

```
| abc |
+-----+
+-----+
```

```
•    JSON_VALID(val)
SELECT JSON_VALID('{\"a\": 1}');
+-----+
+-----+
```

```
| JSON_VALID('{\"a\": 1}') |
+-----+
+-----+
```

```
| 1 |
+-----+
+-----+
```

3.3.2.3.3 JSON 函数数据类型说明

1. 计算节点计算 JSON_OBJECT 函数时，不支持按键名进行键-值对自动排序；
2. 计算节点计算 JSON_TYPE 函数时，目前只能识别 OBJECT、ARRAY、STRING、DOUBLE、INTEGER、BOOLEAN、NULL 这些 JSON 值类型；

- 3. 计算节点不能解析用于表示 JSON 数组子集范围的 to 参数和 last 参数（示例：'\$[last-3 > to last-1]'）。
- 4. 计算节点聚合函数目前有 COUNT、COUNT DISTINCT、SUM、SUM > DISTINCT、MAX、MIN、AVG、AVG > DISTINCT 支持 JSON 数据类型，GROUP_CONCAT 函数支持带 JSON 数据类型的复制表。> 此外，MAX 和 MIN 支持 DOUBLE、INTEGER 等 JSON 值类型，其他 JSON 值类型按字符串处理；
- 5. 计算节点 GROUP BY 和 ORDER > BY 目前支持 STRING、DOUBLE、INTEGER、BOOLEAN、NULL 等 JSON 值类型。但数据节点和计算节点均不支持 OBJECT、ARRAY 等 JSON 值类型进行 GROUP > BY 或 ORDER BY 操作，只能得到一个近似结果；

4 OPERATORS

表 4-1 操作符

运算符	说明
-	减号
-	负号
=	等于操作
% MOD	取余
<=>	等于操作
>	大于
>=	大于等于
!=	不等于
<>	不等于
*	乘
/	除
,OR	或
+	加
<=	小于等于
<	小于
AND, &&	与
BETWEEN ... AND ...	范围判断
DIV	除

运算符	说明
IS	是判断
IS NOT	非判断
IS NOT NULL	判断不为 NULL
IS NULL	判断 NULL
LIKE	LIKE 函数
NOT BETWEEN ... AND ...	范围判断
NOT LIKE	LIKE 函数
NOT,!	非
XOR	异或

5 FUNCTIONS

5.1 GoldenDB 支持函数列表

表 5-1 函数说明表

函数名	说明
ABS	绝对值
AVG	平均值
CASE WHEN	CASE 操作
CAST	类型转换
CEIL/CEILING	最小整数值
CHAR	数值转字符
CHAR_LENGTH	字符长度
COALESCE	返回第一个非空的参数值
CONCAT	返回结果为连接参数产生的字符串
CONVERT	类型转换
COUNT	行总数
COUNT(DISTINCT)	不同值得总数
CURDATE	获取日期
CURRENT_DATE	同 CURDATE
CURRENT_TIMESTAMP	同 NOW
CURTIME	当前时间
DATE	取日期部分值

函数名	说明
DATE_ADD	增加时间
DATE_FORMAT	格式日期
DATE_SUB	减时间
DATEDIFF	两日期值相减
DAY	同 DAYOFMONTH
DAYNAME	返回星期名词
DAYOFMONTH	返回一月的第几天
DAYOFWEEK	返回星期序号
DAYOFYEAR	返回一年的第几天
FLOOR	返回小于等于入参值的最大整数
HEXTORAW	十六进制字符串转换为 RAW
HOUR	返回 time 对应的小时数
IN	IN 函数
INSTR	子串的位置
LAST_DAY	月最后一天
LCASE	同 LOWER
LENGTH	字符串长度
LOWER	转为小写字符
LTRIM	去除空格
MAX	最大值
MIN	最小值
MINUTE	分钟
MONTH	月
NOT IN	IN 函数
NOW	获取当前时间
RAND	随机值
REPLACE	字符串
ROUND	返回参数 X, 其值接近于最近似的整数
RTRIM	除空格
STR_TO_DATE	字符串转为日期
STRCMP	字符串比较
SUBSTR	取子字符串

函数名	说明
SUM	求和
TIME	时间函数
TIMEDIFF	时间相减
TIMESTAMPADD	日期相加
TIMESTAMPDIFF	日期相减
TO_DAYS	转换成天
TO_SECONDS	转换成秒
TRIM	去除空格
UPPER	转为大写字符
WEEK	范围星期值
YEAR	获取年
GROUP_CONCAT	分组连接
FROM_UNIXTIME	时间格式转换
LEFT	字符串左截取
RIGHT	字符串右截取
LPAD	左补字符
RPAD	右补字符
TO_DATE	日期转换函数
SYSDATE	获取系统当前时间
INITCAP	将字符串中各单词首字母转大写，其他字母转小写
NVL	获取非空串
DECODE	类似 CASE WHEN
	连接符
ROWNUM	序列
TO_TIMESTAMP	日期转换函数
TO_CHAR	字符串转换函数
CHR(X)	返回在数据库字符集中与 X 拥有等价数值的字符
TRUNC(x[,y])	算截尾到 y 位小数的 x 值.y 缺省为 0,结果变为一个整数值.如果 y 是一个负数,那么就截尾到小数点左边对应的位上
NEXT_DAY(SYSDATE,X)	从当前日期开始得到到未来星期数的日期
INSTR (string1, string2[a,b])	得到在 string1 中包含 string2 的位置.

函数名	说明
TO_NUMBER()	将 CHAR 或 VARCHAR2 类型的 string 转换为一个 number 类型的数值
SYSTIMESTAMP	函数返回本机数据库上当前系统日期和时间(包括微秒和时区)
ADD_MONTHS(X, Y)	计算在时间 x 之上机上 Y 个月后的时间值，要是 Y 的值为负数的话就是在这个时间点之前的时间值（这个时间-Y 个月）
LENGTH(str)	在设置 Oracle 模式时为获取字符串的字符数；在设置为 MySQL 模式时为获取字符串的字节数
LENGTHB(str)	Oracle 模式和 MySQL 模式下均为获取字符串的字节数
FETCH FIRST N ROW[S] [ONLY]	获取前 n 行
ROW_NUMBER() OVER()	分析函数，会在数据表生成一个排序列
DENSE_RANK() OVER()	

5.2 函数解析模式说明

分布式数据库中涉及到的函数的解析模式分为两种，通过计算节点的配置文件(proxy.ini)中的配置项 parse_mode 进行设置，其值为 1（默认）时表示 MySQL 模式，值为 2 时表示 Oracle 模式，当涉及到的函数在 MySQL 和 Oracle 之间有二义性时需要根据需要设置函数解析模式。以下几个 Oracle 函数与 MySQL 具有二义性，需要开启 Oracle 模式，其余函数均为 MySQL 模式：

- DECODE()函数：在设置 Oracle 模式时为根据条件设置输出的结果，类似于 SELECT > ... CASE ... WHEN > ...；在设置 MySQL 模式时为密文解密，与 ENCODE()为互逆。
- LENGTH()函数：在设置 Oracle 模式时为获取字符串的字符数；在设置为 MySQL 模式时为获取字符串的字节数。
- ||运算符：在设置 Oracle 模式时表示字符串拼接，等价于 MySQL 的 CONCAT()函数；在设置 MySQL 模式时表示逻辑关系"或"。
- 日期减法：在计算两个日期相减且第一个参数或者两个都是日期相关函数时需要设置为 Oracle 模式，否则会将日期字符串转换为数字计算减法。涉及到的日期函数包括 SYSDATE()、STR_TO_DATE()、TO_DATE()、LAST_DAY()。

5.3 复杂函数说明

- CAST/CONVERT 类型转换函数支持 SIGNED > /UNSIGNED/DATE/TIME/DATETIME/DECIMAL 类型。
- NEXT_DAY 函数，第二个参数支持数字，目前不支持对字段操作，第一个参数为纯字段的报错。
- TO_CHAR 函数，支持数值型和日期型转化为字符型，其中日期型转化为字符型时第一个参数可以是时间字符串（时间字符串语法规则与 MySQL 中的 DATE_FORMAT 函数相同）。数值型 FM 格式只支持"FM09"。
- TRUNC 函数，目前支持对数字，不支持对字段操作，支持第一个参数为纯字段的报错。
- TO_DATE 函数，目前支持输入一个或者两个参数，日期字符串格式显示键附录 A.2，当前日期的输出支持按照年月日时分秒的 24 小时制输出，计算日期的减法时要在 parse_mode=2 模式下执行。
- INITCAP 函数，目前支持输入一个参数，参数类型支持常量、字段、函数。支持输入参数个数不为 1 时报错。
- ROW_NUMBER() OVER() 与 DENSE_RANK() > OVER() 窗口函数，目前支持 OVER([PARTITION BY] ORDER > BY)，可以出现的位置包括 FIELD > LIST、FROM 子查询、WHERE 子查询。不支持 OVER 中带有参数的窗口函数与 GROUP > BY、其他聚合函数、ORDER BY 操作共用。

6 PSEUDO COLUMNS

6.1 SEQUENCE

兼容 Oracle SEQUENCE 语法。

6.1.1 语法说明

- CREATE SEQUENCE
CREATE SEQUENCE schema.sequence

INCREMENT BY num

START WITH num

MAXVALUE num | NOMAXVALUE

MINVALUE num | NOMINVALUE

CYCLE | NOCYCLE

CACHE | NOCACHE

- ALTER SEQUENCE

ALTER SEQUENCE schema.sequence

INCREMENT BY num

- DROP SEQUENCE

DROP SEQUENCE schema.sequence

6.1.2 用例

- 例子 1:

```
CREATE SEQUENCE schema.seq_abc INCREMENT BY 1 START WITH 10  
MAXVALUE 9999999 NOCYCLE NOCACHE;
```

- 例子 2:

```
ALTER SEQUENCE schema.seq_abc INCREMENT BY 5;
```

- 例子 3:

```
DROP SEQUENCE schema.seq_abc;
```

6.1.3 使用说明

- SEQUENCE 默认 CACHE 为 100、最大值(int64 最大值)、最小值 1。
- SEQUENCE 负的最小值应该比理论 int64 最小值大 1。
- SEQUENCE 前缀含义非用户概念，类似 DATABASE 概念。
- SEQUENCE 目前不支持 ORDER|NOORDER 属性。
- SEQUENCE 修改步长时会导致当前 CACHE 丢失。
- SEQUENCE 的 DDL 操作需要指定库名。

6.1.4 限制说明

- 由于 SEQUENCE 的值是在 SQL 语句执行之前向 GTM 申请，语句执行失败的情形下申请的 SEQUENCE 值不能够回收，所以表中的数据会有不连续的情形。

6.1.5 最佳实践

无

6.2 伪列 ROWNUM

Oracle 用 ROWNUM 伪列来标示各记录的行号，多用于 SELECT 查询和 WHERE 条件过滤。

6.2.1 语法说明

`SELECT ROWNUM FROM tbl_name WHERE ROWNUM <N;`

6.2.2 用例

- 例子 1:

`SELECT ROWNUM, t1.* FROM t1 WHERE ROWNUM < 3; //单独字段，无别名`

`SELECT ROWNUM AS rn, t1.* FROM t1 WHERE ROWNUM < 3; //单独字段，有别名`

- 例子 2:

`SELECT (ROWNUM+1)*2 AS rn, t1.* FROM t1 WHERE ROWNUM < 3; // 表达式计算`

`SELECT COUNT(ROWNUM), MIN(ROWNUM),MAX(ROWNUM) FROM t1; // 聚合函数`

`SELECT MAX(ROWNUM)+1 AS rn FROM t1 GROUP BY type; //聚合函数+表达式计算`

- 例子 3:

`SELECT ROWNUM AS rn, t1.id FROM t1 UNION ALL SELECT ROWNUM, t1.id FROM t1 ; // union`

6.2.3 限制说明

- `SELECT` 语句中，在表中不存在索引字段且 `ORDER BY` 普通字段的情形下，分布式数据库结果集与 `Oracle` 结果集相同，都是对结果集先进行编号后进行排序，但当多行数据的排序字段值相同时结果集与 `Oracle` 的会有不同。
- `SELECT` 语句中，在表中不存在普通索引且 `ORDER BY` 主键降序排列时，分布式数据库结果集与 `Oracle` 结果集不同，分布式数据库先对结果集进行编号然后按照主键降序排列；`Oracle` 先按照主键降序排列然后再对结果集进行编号。
- `SELECT` 语句中，在表中存在联合索引且 `ORDER BY` 普通字段的情形下，分布式数据库结果集与 `Oracle` 数据库结果集不同，分布式数据库先按照 `ORDER BY` 字段进行排列，然后对结果集进行编号；`Oracle` 数据库先对结果集进行编号然后对结果集进行排序。
- `SELECT` 语句中，在表中存在索引且 `ORDER BY` 主键的情形下，分布式数据库结果集与 `Oracle` 数据库结果集不同，分布式数据库先对结果集进行编号然后进行排序；`Oracle` 数据库先对结果集按照主键进行排序然后进行编号。

6.2.4 最佳实践

无

7 DDL

7.1 CREATE DATABASE

7.1.1 语法说明

CREATE [DATABASE | SCHEMA] [IF NOT EXISTS] db_name

[create_specification] ...

create_specification:

[DEFAULT] CHARACTER SET [=] charset_name

- 字符集支持: latin1、gbk、utf8、utf8mb4 和 gb18030。
- 字符序支持按照二进制值进行, 即对应上述四种字符集的字符序分别为 latin1_bin、gbk_bin、utf8_bin、utf8mb4_bin 和 gb18030_bin。

7.1.2 用例

7.1.2.1 一般建库语句

CREATE DATABASE IF NOT EXISTS db1 CHARACTER SET utf8;

7.1.2.2 分库建库语句

写法 1: 不显式指定物理库名, 系统自动生成, 下面语句生成的物理库名为 db1_1 - db1_8

```
CREATE DATABASE IF NOT EXISTS db1 CHARACTER SET UTF8
LIST(pg1(g1,g2,g3,g4), pg2(g5,g6,g7,g8));
```

写法 2: 显式指定物理库名

```
CREATE DATABASE IF NOT EXISTS db2 CHARACTER SET UTF8
LIST(pg1(g1<my_db1>,g2<my_db2>,g3<my_db3>,g4<my_db4>),
pg2(g5<my_db5>,g6<my_db6>,g7<my_db7>,g8<my_db8>));
```

写法 3: 写法 1 的简写

```
CREATE DATABASE IF NOT EXISTS db3 CHARACTER SET UTF8 LIST(pg1(g1-g4),
pg2(g5-g8));
```

7.1.3 使用说明

对于带有分库属性的库禁止以下操作

- 建表不允许 多级分片表/全局索引。
- 表闪回。
- COPY TABLE。
- 视图。
- 存储过程。

- 自定义函数。
- 同义词。

7.1.4 最佳实践

无

7.2 DROP DATABASE

7.2.1 语法说明

DROP [DATABASE | SCHEMA] [IF EXISTS] db_name

7.2.2 用例

DROP DATABASE IF EXISTS db1;

7.2.3 使用说明

无

7.2.4 最佳实践

无

7.3 ALTER DATABASE

7.3.1 语法说明

ALTER {DATABASE | SCHEMA} [db_name]

alter_specification ...

alter_specification:

[DEFAULT] CHARACTER SET [=] charset_name

- 字符集支持：latin1、gbk、utf8、utf8mb4 和 gb18030。
- 字符序支持按照二进制值进行，即对应上述四种字符集的字符序分别为 latin1_bin、gbk_bin、utf8_bin、utf8mb4_bin 和 gb18030_bin。

7.3.2 用例

ALTER DATABASE db1 CHARACTER SET gbk;

7.3.3 使用说明

无

7.3.4 最佳实践

无

7.4 CREATE TABLE

7.4.1 语法说明

CREATE TABLE [IF NOT EXISTS] table_name

(create_definition, ...)

[table_options]

[partition_options]

[distributed_option]

[select_statement]

| LIKE table_name

create_definition:

col_name column_definition

| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)

[index_option] ...

| {INDEX|KEY} [index_name] [index_type] (index_col_name,...)

[index_option] ...

| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]

[index_name] [index_type] (index_col_name,...)

[index_option] ...

| GLOBAL PRIMARY KEY [index_type] (index_col_name,...)

[index_option] ...

| GLOBAL [UNIQUE] [INDEX|KEY]

[index_name] [index_type] (index_col_name,...)

[index_option] ...

column_definition:

data_type [NOT NULL | NULL] [DEFAULT default_value]

[AUTO_INCREMENT][UNIQUE [KEY] | [PRIMARY] KEY | GLOBAL PRIMARY KEY|

GLOBAL UNIQUE [KEY]][COMMENT 'string']

data_type:

- | TINYINT[(length)] [UNSIGNED]
- | SMALLINT[(length)] [UNSIGNED]
- | MEDIUMINT[(length)] [UNSIGNED]
- | INT[(length)] [UNSIGNED]
- | INTEGER[(length)] [UNSIGNED]
- | BIGINT[(length)] [UNSIGNED]
- | DOUBLE[(length,decimals)] [UNSIGNED]
- | FLOAT[(length,decimals)] [UNSIGNED]
- | DECIMAL[(length[,decimals])] [UNSIGNED]
- | NUMERIC[(length[,decimals])] [UNSIGNED]
- | DATE
- | TIME[(fsp)]
- | TIMESTAMP[(fsp)]
- | DATETIME[(fsp)]
- | YEAR
- | CHAR[(length)]
- | VARCHAR(length)
- | BINARY[(length)]
- | VARBINARY(length)
- | BLOB
- | TEXT [BINARY]
- | ENUM(value1,value2,value3,...)
- | SET(value1,value2,value3,...)

index_col_name:

col_name [(length)]

index_type:

USING {BTREE | HASH}

index_option:

| COMMENT 'string'

table_options:

AUTO_INCREMENT [=] value

| COMMENT [=] 'string'

| COMPRESSION = {'ZLIB' | 'LZ4' | 'NONE'}

| [DEFAULT] CHARACTER SET [=] charset_name

| ENGINE [=] engine_name

| MAX_ROWS [=] value

| ROW_FORMAT [=]

{DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}

| [DEFAULT] CHARACTER SET [=] charset_name

| CONNECTION [=] 'connect_string'

| KEY_BLOCK_SIZE [=] value

partition_options:

PARTITION BY

{ [LINEAR] HASH(expr)

| [LINEAR] KEY [ALGORITHM={1|2}] (column_list)

| RANGE{(expr) | COLUMNS(column_list)}

| LIST{(expr) | COLUMNS(column_list)} }

[PARTITIONS num]

[(partition_definition [, partition_definition] ...)]

[PARTITIONSTORAGEDB(group_list)

(partition_options)]

partition_definition:

```

PARTITION partition_name
[VALUES
{LESS THAN {(expr | value_list) | MAXVALUE}
|
IN (value_list)}]
[COMMENT [=] 'comment_text' ]
distributed_option:
DISTRIBUTED BY
HASH(column_list) ({groupno},...)
| RANGE(expr) ({groupno VALUES LESS THAN { value | MAXVALUE}},...)
| RANGE(expr)({groupno [value,value]},...)
| RANGE(expr)({groupno [value,value] [value,value]}...,...)
| LIST(expr) ({groupno VALUES IN (value_list | DEFAULT)},...)
| DUPLICATE ({groupno},...)

```

- 说明

FSP 表示日期时间精度；即秒的小数点后的位数。这个 FSP 值（如果有）必须在 0 到 6 之间。值为 0 表示没有小数部分。如果省略，默认精度为 0。

7.4.1.1 distributed_option 说明

distributed_option 为分布式表的分发选项定义。

新增 range 表创表分发范围带区间形式，目前仅支持左闭右开（g1[0,100)）形式，该语法不影响原有创表语法规则，不支持和原有分发规则混用。

- **单节点存储：**表数据只存储于一个数据节点上，且分发策略为复制类型。
设置方法：

DISTRIBUTED BY DUPLICATE(groupno);

- **多节点分布存储：**表数据是水平分布在不同数据节点上，数据分发策略包括：HASH 分发、RANGE 分发、LIST 分发等。

设置方式为在 distributed_option 中设置分发字段和分发策略，设置存储的数据节点安全组。

数据节点安全组的个数可以设置一个或多个。

- **复制表：**表数据是全量存储在各个数据节点上。

设置方式为在 `distributed_option` 中设置 `DUPLICATE`，设置要全量存储的数据节点安全组。

数据节点安全组的个数可以设置一个或多个。

7.4.1.2 数据分发策略原则

- **HASH 分发**

适用于将数据均匀的分布到预先定义的几个数据节点上，保证各数据节点的数据数量大致一致，一般情况下不需要关心分发字段的值的具体含义。

比较适合场景：该表上的操作基本都是等值操作。

- **RANGE 分发**

适用于指定一个给定的列值或列值集合应该保存在哪个数据节点上，常用于时间字段上，比如数据按照自然月或天来分布存储。

- **LIST 分发**

适用于含有一系列限定性值的列的场景，常用于机构代码或国家代码字段上。

- **复制表**

适用于小表且不常修改的表，或者 JOIN 和子查询中使用的表。主要目的是减少节点间网络数据的传输，以提高查询的性能。

- **针对多节点分布存储的表**

在采用 HASH/RANGE/LIST 分发策略时，尽量保持各个数据节点上的数据量均衡，均衡的数据分布在多节点并行计算时会有更高的性能表现。

- **主文件类的表**

优先考虑用 HASH 分发策略，分发键使用业务主键，对于联机和批量同时存在的情况优先保障联机场景。

- **其他场景优先考虑使用 RANGE/LIST 分发策略。**

7.4.1.3 分发选项说明

表 7-1 分发选项说明表

分发选项	说明
HASH	HASH 分发
RANGE	范围分发
LIST	列表分发

分发选项
DUPLICATE
GroupNo

说明
复制表
存储安全组定义

7.4.1.4 不同分区不同分片表

计算节点通过关键字 PARTITIONSTORAGEDB 可以在不同的分片上指定不同的分区规则。

```
create table ps_test.range(  
int_1 int not null,  
int_2 int,  
date_1 date default '2021-01-01',  
time_1 time,  
datetime_1 datetime,  
timestamp_1 timestamp,  
year_1 year,  
json_1 json,  
primary key(date_1, int_1, int_2)  
)  
partition by range (TO_DAYS(date_1))  
(  
partition p202101 values less than (TO_DAYS('2021-01-01')),  
partition p202102 values less than (TO_DAYS('2021-02-01')),  
partition p202103 values less than (TO_DAYS('2021-03-01')),  
partition p202104 values less than (TO_DAYS('2021-04-01')),  
partition p202105 values less than (TO_DAYS('2021-05-01')),  
partition p202106 values less than (TO_DAYS('2021-06-01')),  
partition p202107 values less than (TO_DAYS('2021-07-01')),  
partition p202108 values less than (TO_DAYS('2021-08-01')),
```

```

partition p202109 values less than (TO_DAYS('2021-09-01')),
partition p202110 values less than (TO_DAYS('2021-10-01')),
partition p202111 values less than (TO_DAYS('2021-11-01')),
partition p202112 values less than (TO_DAYS('2021-12-01')),
partition other values less than maxvalue)
PARTITIONSTORAGEDB(g1,g2)
(partition by range(int_1)
(
partition p0 values less than (100),
partition p1 values less than (200),
partition p2 values less than (300),
partition p3 values less than MAXVALUE))
PARTITIONSTORAGEDB(g3)
(partition by range(int_2)
(
partition p0 values less than (10000),
partition p1 values less than (20000),
partition p2 values less than (30000),
partition p3 values less than MAXVALUE))
distributed by hash(int_1)(g1,g2,g3,g4);

```

对于表 `ps_test.range`

通过 `PARTITIONSTORAGEDB(g1,g2)`，指定 `g1,g2` 上的分片规则为

```

PARTITION BY RANGE (`int_1`)
(PARTITION p0 VALUES LESS THAN (100) ENGINE = InnoDB,
PARTITION p1 VALUES LESS THAN (200) ENGINE = InnoDB,
PARTITION p2 VALUES LESS THAN (300) ENGINE = InnoDB,

```


PARTITION p3 VALUES LESS THAN MAXVALUE ENGINE = InnoDB)

g3 上的分发规则

PARTITION BY RANGE (`int_2`)

(PARTITION p0 VALUES LESS THAN (10000) ENGINE = InnoDB,

PARTITION p1 VALUES LESS THAN (20000) ENGINE = InnoDB,

PARTITION p2 VALUES LESS THAN (30000) ENGINE = InnoDB,

PARTITION p3 VALUES LESS THAN MAXVALUE ENGINE = InnoDB)

g4 未通过 PARTITIONSTORAGEDB 指定分发策略，使用默认分发策略

PARTITION BY RANGE (to_days(`date_1`))

(PARTITION p202101 VALUES LESS THAN (738156) ENGINE = InnoDB,

PARTITION p202102 VALUES LESS THAN (738187) ENGINE = InnoDB,

PARTITION p202103 VALUES LESS THAN (738215) ENGINE = InnoDB,

PARTITION p202104 VALUES LESS THAN (738246) ENGINE = InnoDB,

PARTITION p202105 VALUES LESS THAN (738276) ENGINE = InnoDB,

PARTITION p202106 VALUES LESS THAN (738307) ENGINE = InnoDB,

PARTITION p202107 VALUES LESS THAN (738337) ENGINE = InnoDB,

PARTITION p202108 VALUES LESS THAN (738368) ENGINE = InnoDB,

PARTITION p202109 VALUES LESS THAN (738399) ENGINE = InnoDB,

PARTITION p202110 VALUES LESS THAN (738429) ENGINE = InnoDB,

PARTITION p202111 VALUES LESS THAN (738460) ENGINE = InnoDB,

PARTITION p202112 VALUES LESS THAN (738490) ENGINE = InnoDB,

PARTITION other VALUES LESS THAN MAXVALUE ENGINE = InnoDB)

对于不同分区不同分片表，show create table 并不展示表的 partition 定义，展示提示信息，需要通过 show create table 加 storedb 展示具体 db 的分区定义。

```

SHOW CREATE TABLE ps_test.range;

Create Table: CREATE TABLE `range` (
  `int_1` int(11) NOT NULL,
  `int_2` int(11) NOT NULL DEFAULT '0',
  `date_1` date NOT NULL DEFAULT '2021-01-01',
  `time_1` time DEFAULT NULL,
  `datetime_1` datetime DEFAULT NULL,
  `timestamp_1` timestamp NULL DEFAULT NULL,
  `year_1` year(4) DEFAULT NULL,
  `json_1` json DEFAULT NULL,
  PRIMARY KEY (`date_1`,`int_1`,`int_2`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
DISTRIBUTED BY HASH(`int_1`)(g1,g2,g3,g4)

THIS TABLE HAS DIFFERENT PARTITION IN DIFFERENT GROUP, PLEASE USE
STORAGEDB

```

想要知道具体 group 的分区定义，通过语句

```
SHOW CREATE TABLE ps_test.range STORAGEDB G1;具体进行查看。
```

7.4.1.5 多级分片表（该功能为企业版特性）

```
DISTRIBUTED BY
```

```
CASE case_value
```

```
WHEN when_value THEN statement_list
```

```
[WHEN when_value THEN statement_list] ...
```

```
[ELSE statement_list]
```

```
END
```

或者

```
DISTRIBUTED BY
```

CASE

WHEN case_value > when_value THEN statement_list

[WHEN case_value < when_value THEN statement_list] ...

[ELSE statement_list]

END

- **case_value:**
 - 字段名;
 - 简单表达式(+、-、*、/);
 - TIME、DATE、HOUR、MINUTE、DAY、MONTH、YEAR 和 SUBSTR (SUBSTRING)函数;
- **when_value:**
 - value 值;
- **statement_list:**
 - CASE 分支, SUBDISTRIBUTED BY;
 - group 值;
 - 多组分发策略;
 - subdistributed by 支持 range 带区间形式（仅支持左闭右开）;

7.4.1.6 使用 DEFAULT 值的 LIST 表

对于 LIST 分发策略, 如果无法穷举分发值, 可以通过使用 DEFAULT, 来将不能穷举的值, 分发到指定的 GROUP 上。

```
CREATE TABLE tb1(id INT KEY) DISTRIBUTED BY LIST(id)(G1 VALUES  
IN(100,200,300), G2 VALUES IN(400,500), G3 VALUES IN(DEFAULT));
```

对于上述语句, 分发键不在穷举值[100,200,300,400,500]之外的值, 全部都落在 DEFAULT 的 G3 中。

对于 ALTER TABLE 修改表分发策略, 涉及到 DEFAULT 分区的场景, 支持和不支持的典型场景说明如下:

支持的场景:

- 支持新增 DEFAULT 分区

```
CREATE TABLE titles(
id int key
)
distributed by list(id)(g1 values in(100,200,300), g2 values in(400,500));

ALTER TABLE titles
distributed by list(id)(g1 values in(100,200,300), g2 values in(400,500), g3 values
in(default));
```

- 支持对非 default 分区做扩展（暂时不考虑数据问题，由业务侧自己保证）

```
CREATE TABLE titles(
id int key
)
distributed by list(id)(g1 values in(100,200,300), g2 values in(400,500), g3 values
in(default));

ALTER TABLE titles
distributed by list(id)(g1 values in(100,200,300), g2 values in(400,500,600), g3
values in(default));
```

不支持的场景：

- 不支持非扩展的修改分片界限值

```
CREATE TABLE titles(
id int key
)
distributed by list(id)(g1 values in(100,200,300), g2 values in(400,500), g3 values
in(default));

ALTER TABLE titles
distributed by list(id)(g1 values in(100,200), g2 values in(300,400,500), g3 values
in(default));
```

7.4.1.7 单个分片下支持多个区间段的 RANGE 表

| RANGE(expr) ({groupno VALUES LESS THAN { value | MAXVALUE}},...)

| RANGE(expr)({groupno [value,value]},...)

| RANGE(expr)({groupno [value,value] [value,value]...},...)

对于 RANGE 分发策略，既支持 VALUES LESS THAN 的形式，也支持指定区间范围的形式（即[value,value)，左闭右开区间），并且支持在单个分片上指定多个区间段，各个分片上区间段数量可以不相同，各区间段不要连续且递增。例如：

```
CREATE TABLE tb1(id INT KEY) DISTRIBUTED BY RANGE(id)
(
  G1[100,200)[300,400),
  G2[200,300) [400,500),
  G3[50,100)
);
```

对于指定区间范围的 RANGE 表建表语句，存在以下限制条件：

- 指定区间范围的形式与 VALUES LESS THAN 的形式不能混用
- 各区间段的范围互不重叠
- 每个区间段里，右边界值须大于左边界值
- 对于多级分片表，指定区间范围的 RANGE 形式仅支持在叶子节点使用（即 subdistributed by）
- 区间段数量不超过分片数量最大值限制，对于多级分片表则是每一层叶子节点(subdistributed by)区间段数量不超过分片数量最大值

7.4.1.8 自定义 HASH 算法

对于 HASH 分发模式，支持在分发策略后方添加参数来使用其它 HASH 算法计算分发节点，示例如下：

```
CREATE TABLE tb1
(
  id INT NOT NULL PRIMARY KEY,
  cno VARCHAR(20)
)
```

DISTRIBUTED BY HASH(cno)(g1,g2,g3,g4)

ALGORIT

HM='plugin_name/algorithm_name',PARAM='oper=lpad,length=15,data="0";

其中，`plugin_name` 为算法插件名，即动态库`.so`文件的名称（不包含`lib`和后缀，如：`libxxx.so`的名称为`xxx`），`algorithm_name`表示函数的名称。`PARAM`参数用于在计算前对分发键值做处理，目前支持`lpad`操作，示例中的含义为：如果分发键值长度不足15，则用"0"对其进行左填充，使用处理后的值计算HASH值。如无`PARAM`参数，使用原分发键值计算。

该语法目前限制条件：

- 不支持多级分片表
- ALGORITHM 中不允许有空格
- PARAM 中 oper 目前仅支持 lpad
- PARAM 中 oper、length、data 使用小写
- PARAM 中 length 长度限制 ≤ 10000

使用说明：

1、算法编写

- 需严格按照约定格式，参考 `murmur.h` 和 `murmur.cc` 格式

2、编译

- 使用 `gcc/g++ 4.8.5` 版本编译
- 编译环境(架构系统)必须与 GoldenDB 安装环境保持一致
- 动态库编译参数示例为 `g++ -shared -fPIC -fsigned-char -o libhermes.so murmur.cc`，其中 `libhermes.so` 为编译生成的动态库文件，`hermes` 为算法插件名，可自定义

3、加载动态库

- 在 GoldenDB 的 insight 界面加载动态库 `so` 文件，或手动替换，即将编译好的算法动态库`.so`文件放在 `proxy` 用户的 `lib` 目录下，并确认或修改文件用户和属组与 `bin` 目录的可执行文件 `dbproxy` 一致
- 使用前需检查环境，如果已成功加载了非法的动态库 `so` 文件，则加载新的合法的动态库 `so` 文件需要重启 CN 才能正常生效

7.4.1.9 无主键表

[GENERAL]段 配置项 support_without_primarykey = 1，即可支持创建无主键表。

使用限制：

- 无主键表创建分区时，请确保 DN 上配置项 remove_partition_key_limitation = ON

不支持的场景：

- 无主键表成功建表后，不支持新增主键
- 不支持全局索引相关功能
- 不支持 replace into 语法
- 不支持 create select 语法
- 不支持通过分发策略配置文件设定分发策略

7.4.2 用例

- HASH 表

```
CREATE TABLE titles(  
title_id INT NOT NULL,  
title VARCHAR(80) NOT NULL,  
pub_id VARCHAR(4),  
PRIMARY KEY (title_id)  
)  
DISTRIBUTED BY HASH(title_id) (g1,g2);  
  
CREATE TABLE t1(  
id INT KEY,  
f DATETIME  
)  
DISTRIBUTED BY HASH(f)(g1,g2,g3);
```

- RANGE 表

```
CREATE TABLE test.t1 (
```

```

year_col INT,
col2 CHAR(5) NOT NULL PRIMARY KEY)
DISTRIBUTED BY RANGE(year_col)(
g1 VALUES LESS THAN (1991),
g2 VALUES LESS THAN (1995),
g3 VALUES LESS THAN MAXVALUE);
CREATE TABLE t1(
id INT KEY,
f DATETIME)
DISTRIBUTED BY RANGE(MONTH(f))(g1 VALUES LESS THAN(5), g2 VALUES LESS
THAN(12));

```

```

CREATE TABLE test.t1 (
year_col INT,
col2 CHAR(5) NOT NULL PRIMARY KEY)
DISTRIBUTED BY RANGE(year_col)(
g1[0,1970) [2000,2010) [1990,1995),
g2[1970,1990),
g3[2010,2020) [2020,2030));

```

- LIST 表

```

CREATE TABLE test.t1 (
year_col INT,
col2 CHAR(5) NOT NULL PRIMARY KEY)
DISTRIBUTED BY LIST(year_col)(
g1 VALUES IN (1991,1992,1993),
g2 VALUES IN (1994,1995));

```

- DUPLICATE 表

```

CREATE TABLE IF NOT EXISTS test.dup_t1(

```



```

int_1 INT NOT NULL PRIMARY KEY,
set_1 SET('One','Two','Three','Four') NOT NULL,
set_2 SET('A','B','C','D','E','F','G'),
enum_1 ENUM('S','M','L','XL','XXL','XXXL') NOT NULL,
enum_2 ENUM('F','M') NOT NULL
) DISTRIBUTED BY DUPLICATE(g1,g2);

```

- 单节点表

```

CREATE TABLE titles(
title_id INT NOT NULL,
title VARCHAR(80) NOT NULL,
pub_id VARCHAR(4) NULL,
PRIMARY KEY (title_id)
)
DISTRIBUTED BY DUPLICATE(g1);

```

- 多级分片表

```

DELIMITER //
CREATE TABLE titles(
title_id INT NOT NULL,
cust_id INT NOT NULL,
cust_date DATETIME NOT NULL,
PRIMARY KEY (title_id)
) DISTRIBUTED BY
CASE YEAR(cust_date)
WHEN 2019 THEN g1 ;
WHEN 2018 THEN g2 ;
WHEN 2017 THEN g3 ;
ELSE g4;

```

```
END CASE; //
```

```
DELIMITER ;
```

- 多级分片表

```
DELIMITER //
```

```
CREATE TABLE titles(
```

```
title_id INT NOT NULL,
```

```
cust_id INT NOT NULL,
```

```
pub_id INT NOT NULL,
```

```
PRIMARY KEY (title_id)
```

```
) DISTRIBUTED BY
```

```
CASE pub_id
```

```
WHEN 1 THEN
```

```
CASE
```

```
WHEN cust_id<100 THEN SUBDISTRIBUTED BY HASH(title_id)(g1);
```

```
ELSE SUBDISTRIBUTED BY HASH(title_id)(g2);
```

```
END CASE;
```

```
WHEN 2 THEN SUBDISTRIBUTED BY HASH(title_id)(g3);
```

```
ELSE SUBDISTRIBUTED BY HASH(title_id)(g4);
```

```
END CASE; //
```

```
DELIMITER ;
```

- 多级分片表:

```
delimiter //
```

```
CREATE TABLE titles1(
```

```
title_id int not null,
```

```
cust_id int not null,
```

```
pub_id int not null,
```

```
PRIMARY KEY (title_id)
```

```

) distributed by
case pub_id
when 1 then
case
when cust_id<100 then subdistributed by range (title_id)(g1 [ 100,200));
else subdistributed by duplicate(g2);
end case;
else subdistributed by list(title_id)(g3 values in (10,20,30));
end case; //
delimiter ;

```

- 多级分片表:

```

delimiter //
CREATE TABLE titles2(
title_id int not null,
cust_id int not null,
pub_id int not null,
PRIMARY KEY (title_id)
) distributed by
case pub_id
when 1 then
case
when cust_id<100 then subdistributed by range (title_id)(g1 [100,200) [200,300));
else subdistributed by hash(title_id)(g2);
end case;
else subdistributed by list(title_id)(g3 values in (10,20,30));
end case; //
delimiter ;

```

- 不支持 range 区间形式和 range less than 形式混用，例如以下用例报错

```
CREATE TABLE titles3(
title_id int not null,
cust_id int not null,
pub_id int not null,
PRIMARY KEY (title_id)
) distributed by range (title_id)(
g1 [100,200),
g2 values less than(5000));
delimiter //

CREATE TABLE titles3(
title_id int not null,
cust_id int not null,
pub_id int not null,
PRIMARY KEY (title_id)
) distributed by
case pub_id
when 1 then
case
when cust_id<100 then subdistributed by range (title_id)(g1 [100,200));
else subdistributed by range (title_id)(g2 values less than(5000));
end case;
else subdistributed by list(title_id)(g3 values in (10,20,30));
end case; //
delimiter ;
```

- CREATE TABLE LIKE:
CREATE TABLE t1 LIKE t2;

7.4.3 使用说明

7.4.3.1 主键的限制

- 表必须定义主键。
- 主键不允许使用 BLOB、TEXT、FLOAT、DOUBLE 类型。
- 无主键表参见 7.4.1.9。

7.4.3.2 分发键限制

- **HASH**: 分发键支持全部类型（除 TEXT、BLOB、ENUM、SET、BINARY、VARBINARY、JSON），不支持表达式和函数。
- **RANGE > LIST**: 分发键表达式的结果支持整数、时间函数和字符串类型。具体介绍如下：
 - 整形类型：支持整形和加减乘除简单表达式。
 - 字符串型：支持用字符的码值比较，支持 SUBSTRING 函数，不支持表达式和其它函数。建议使用有实际意义的字符串进行比较。
 - 时间类型：支持时间函数使用时间函数包括 DATE/DAY/MONTH/YEAR/HOUR/MINUTE/TIME，时间函数中的字段类型支持 TIME/DATETIME/TIMESTAMP 类型。
- 支持 DIV、MOD 运算，表达式的结果取整进行分发处理，推荐在使用 DIV、MOD 运算时使用整形类型进行处理。
- 若用户不指定分发信息，优先依据分发策略配置文件中配置的分发策略进行分发；分发策略配置文件未配置、不生效或不支持场景下，依据配置项 default_distribute_method 指定的分发策略进行分发。
- 单表/多表 WHERE 多分发键过滤功能说明
 - 对于 RANGE 表多分发键不支持范围交集过滤，比如 "k1>20 AND k2 >10 > AND k1<10 AND k2<20" 不支持精确过滤。
 - 分布表达式的函数值必须是单调的，比如：x*x, x/y, x*y + > 1 等，这些表达式都是非单调表达式，目前无法正确过滤。
 - 不支持表达式过滤，比如 WHERE t.k1+t.k2 > 100。
- 多级分片表限制
 - 支持创建多级分片表 range 带区间形式，仅支持左闭右开。
 - 创建多级分片表 range 带区间形式，区间范围不能交叉。

- 创建多级分片表 **range** 带区间形式不支持与 **range less than** 形式混用。
- 创建多级分片表 **range** 带区间形式不同层不允许出现相同的分片节点，同一层可以有相同的分片节点。
- **HASH** 分发以外的分发键不能为空;当配置文件开启功能后，**HASH** 分发支持分发键设置为 **NULL**，目前此功能支持 **char**、**varchar**、**varchar2** 类型的分发键。

7.4.3.3 表限制

- 表定义不能有 **GTID** 为前缀的列。
- **CREATE...SELECT** 语句的限制。
 - **SELECT** 语句目前支持单表，该表的分发类型任意。
 - 新建的表必须显式指定一个主键字段。
 - 新建的表使用 **SELECT** 表的分发属性作为新表的分发属性，暂不支持显式指定 **CREATE TABLE... SELECT** 的分发属性。
 - 新建表中不支持指定全局索引、自增列，不支持添加新的列。
 - 新建表为多级分片表时，不支持修改新表分发键名。
 - **SELECT** 语句中的字段必须包含 **SELECT** 语句中单表的分发键，如：

```
CREATE TABLE abczyy_cs.tb1(a INT KEY, b INT,
c VARCHAR(30))DISTRIBUTED BY HASH(a)(g1,g2,g3);
CREATE TABLE abczyy_cs.tb2(d INT KEY default 0)
SELECT a,b,c FROM abczyy_cs.tb1;
```

7.4.3.4 数据类型限制

所有的 DDL 操作中避免使用中文的库名、表名字段名。

7.4.3.5 自增列相关限制

- 支持创建自增列表，不支持建表后，再通过 **ALTER TABLE** 增加自增列。
- 自增列字段支持整形字段类型。

7.4.3.6 字段属性限制

- 不支持字段列表中字段属性为 **SIGNED**。

如：

```
CREATE TABLE IF NOT EXISTS tests123.tb1(a INT KEY, b INT,  
c INT, d INT, e INT SIGNED) DISTRIBUTED BY DUPLICATE(g1,g2,g3);
```

- 字段默认值使用函数时需使用括号转义的格式。

如使用 `to_number` 函数：`pid decimal(1) (default to_number(substr(to_char(sysdate,'ss'),-1,1)))`。错误示例：`pid decimal(1) default to_number(substr(to_char(sysdate,'ss'),-1,1))`。

7.4.3.7 自增列有关配置项

对于表中的自增列，会创建一个 `sequence` 用于全局控制该自增列的特性，存在配置项可以控制自增列是否使用 `sequence` 流程。

[GENERAL]

`proxy_ignore_auto_inc = 0`

● 0 使用 `sequence` 流程

● 1 不适用 `sequence` 流程，在 `proxy` 层面不对自增值进行控制，各 `db` 上使用自己的自增特性

自增列不走 `sequence` 流程，使用 `db` 节点自增列功能影响评估如下：

1、复制表可能会数据不一致

原因：并发场景下 `insert` 顺序会不一致

2、`hash`、`range`、`list` 表不同节点自增列会有相同值，不能保证全局唯一

3、自增列字段不能做分发键

原因：作为分发键在 `insert` 中不显示指定值，会计算不出具体 `group` 节点会插入失败。

4、自增列不能做全局索引

原因：自增列的值 `proxy` 获取不到，无法插入全局索引表中。

5、自增列不能做主键

原因：在 `update` 分发键时会导致主键冲突或者当前值跳值，当前值跳值会导致当前 `db`

6、重分布可能会有主键冲突

7、导入功能可能有主键冲突

7.4.4 最佳实践

7.4.4.1 表级别定义原则

- 小表：数据量级别在 10 万以下。
- 大表：数据量级别在百万以上。

7.4.4.2 表的设计原则

- **单节点存储的表**

适用于下面几种场景：

- 非大表建议采用单节点存储。
- 需要进行 JOIN 的表要考虑放到同一个数据节点上。
- 建表时要考虑各个数据节点的数据均衡。

- **多节点分布存储的表**

适用于下面几种场景：

- 大表考虑采用多节点分布存储。
- 需要进行 JOIN 的大表要考虑采用相同的分发字段和分发策略。

例如：在交易系统中，分布字段首先考虑要满足对联机交易的性能优化支持，比如账户表可以采用用户卡号作为分发字段。

- **复制表**

适用于下面几种场景：

- 小表且更新不频繁。
- 在 JOIN 或者子查询中使用的表。

- **去范式化原则**

在分布式系统中，通常去范式化以空间换时间来提升系统性能，即在"不影响数据唯一化"原则下，去范式化后的冗余数据字段在数据库中只能唯一属于一张表，否则无法确保数据一致性。

当查询语句中出现表的 JOIN 操作时，尤其是针对分布式表间出现关联查询时，建议首先考虑采用去范式化设计来规避 JOIN 操作，达到提高查询效率的目的。

去范式化原则特别适合于需要冗余的字段很少更新的业务场景。

7.5 DROP TABLE

7.5.1 语法说明

DROP TABLE [IF EXISTS] tbl_name

7.5.2 用例

闪回开关关闭时，drop table 直接删除表。

DROP TABLE IF EXISTS db1.t1;

闪回开关开始时，drop table 并不直接删除表，生成闪回备份。

DROP TABLE IF EXISTS db1.t1;

闪回开关开始时，drop table purge 直接删除表。

DROP TABLE IF EXISTS db1.t1 purge;

闪回开关开始时，drop table no purge 并不直接删除表，生成闪回备份。

DROP TABLE IF EXISTS db1.t1 no purge;

7.5.3 使用说明

- 不支持删除多个表的语法。
- 闪回开关开启时，drop table 与 drop table no purge 相同。
- 不支持删除全局索引表辅助表（全局索引表辅助表是在建立全局索引时，proxy 根据索引键建立的表，由库名+表名+索引名根据 hash 算法计算出，以 t 开头，只有在 DB 上可以看到，例如
"t91800075761001448840000000000000152458323236710148271495"
）。

7.5.4 最佳实践

无

7.6 ALTER TABLE

7.6.1 语法说明

ALTER TABLE tbl_name

[alter_specification]

[partition_options] [distributed_option | distributed_force_option]

alter_specification:

add_col_def [, add_col_def] ...

| ADD {INDEX|KEY} [index_name]

[index_type] (index_col_name,...)

| ADD [CONSTRAINT [symbol]]

UNIQUE [INDEX|KEY] [index_name]

| ADD GLOBAL [UNIQUE] INDEX [index_name]

[index_type] (index_col_name,...)

| ADD UNIQUE [INDEX|KEY] [index_name]

[index_type] (index_col_name,...)

| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}

| MODIFY [COLUMN] col_name column_definition

| DROP [COLUMN] col_name

| DROP {INDEX|KEY} index_name

| ADD PARTITION (partition_definition)

| DROP PARTITION partition_names

| TRUNCATE PARTITION {partition_names | ALL}

| REMOVE PARTITIONING

| REORGANIZE PARTITION partition_names INTO (partition_definitions)

| COALESCE PARTITION number

| RENAME [TO|AS] new_tbl_name

| CHANGE [COLUMN] old_col_name new_col_name column_definition

[FIRST|AFTER col_name]

| COMPRESSION = {'ZLIB' | 'LZ4' | 'NONE'}

| RENAME [INDEX|KEY] old_idx_name TO new_idx_name

| RENAME COLUMN old_col_name TO new_col_name

add_col_def:

ADD [COLUMN] col_name column_definition

[FIRST | AFTER col_name]

index_col_name:

col_name [(length)]

partition_options:

partition_option [partition_option] ...

partition_option:

ADD PARTITION (partition_definition)

| DROP PARTITION partition_names

| TRUNCATE PARTITION {partition_names | ALL}

| COALESCE PARTITION number

| REORGANIZE PARTITION partition_names INTO (partition_definitions)

| EXCHANGE PARTITION partition_name WITH TABLE tbl_name

| ANALYZE PARTITION {partition_names | ALL}

| CHECK PARTITION {partition_names | ALL}

| OPTIMIZE PARTITION {partition_names | ALL}

| REBUILD PARTITION {partition_names | ALL}

| REPAIR PARTITION {partition_names | ALL}

| REMOVE PARTITIONING

注意：针对多分片场景下 PARTITION BY LIST COLUMNS 且 含有 DEFAULT 分区的情况，只支持 ADD、DROP、REORGANIZE、TRUNCATE 和 REMOVE 的分区操作。

distributed_option:

DISTRIBUTED BY

| RANGE(expr)({groupno [value,value]},...)

| RANGE(expr)({groupno [value,value] [value,value]}...,...)

| LIST(expr) ({groupno VALUES IN (value_list)},...)

distributed_force_option:

DISTRIBUTED BY distributed_option FORCE;

distributed_option:

DISTRIBUTED BY

HASH(column_list) ({groupno},...)

| RANGE(expr) ({groupno VALUES LESS THAN { value | MAXVALUE}},...)

| RANGE(expr)(groupno [value,value],...)

| LIST(expr) ({groupno VALUES IN (value_list | DEFAULT)},...)

| DUPLICATE ({groupno},...)

多级分片表（该功能仅企业版支持）：

ALTER TABLE tbl_name distributed by

CASE case_value

WHEN when_value THEN statement_list

[WHEN when_value THEN statement_list] ...

[ELSE statement_list]

END

或者：

ALTER TABLE tbl_name distributed by

CASE

WHEN case_value > when_value THEN statement_list

[WHEN case_value < when_value THEN statement_list] ...

[ELSE statement_list]

END

参数说明：

case_value:

1) 字段名；

- 2) 简单表达式(+、-、*、/);
- 3) time、date、hour、minute、day、month、year 和 substr (substring)函数;

when_value:

- 1) value 值;
statement_list:
- 1) case 分支, subdistributed by;
- 2) group 值;
- 3) 多组分发策略;
- 4) subdistributed by 支持 range 分发带区间形式(仅支持左闭右开)及 less > than 形式;
- 5) subdistributed by 支持 list 分发增加新的 group 或新的 group 值;
- 6) subdistributed by 支持 hash、duplicate 分发;

支持如下语法:

- 碎片整理语法:
 - 1、ALTER TABLE ENGINE 是在线 DDL 操作, 对 DML 操作影响不大;
 - 2、如果表存在全文索引, ALTER TABLE ENGINE 变成离线 DDL, 可能会长时间阻塞 DML 操作;
 - 3、DML 操作需要尽早提交, 不能有大事务, 否则会导致 ALTER TABLE ENGINE 在线 DDL 失败;
 - 4、配置项 innodb_sort_buffer_size、innodb_online_alter_log_max_size 与 innodb_online_alter_log_max_size 设置必须够大, MySQL 临时文件夹也必须够大, 否则会失败。

```
ALTER TABLE t2 ENGINE=innodb, ALGORITHM=INPLACE, LOCK=NONE
```
- 压缩算法语法:
 - 1、可在 CREATE TABLE 或 ALTER TABLE 时指定 COMPRESSION 属性, 支持的压缩算法包括 Zlib 和 LZ4;
 - 2、使用 ALTER TABLE 时, 只有新数据才会使用压缩算法, 如想对已有文件处理, 需使用 OPTIMIZE TABLE;
 - 3、操作系统需支持: sparse file and hole punching (稀疏文件和文件打洞), 每个表使用独立表空间;

4、配置项 `innodb_compression_level` (1-9) 用于动态调整 `zlib` 的压缩级别，默认为 6。越小压缩效果越差，但压缩速度越快。值越大，压缩效果越好且能降低压缩失败及 `page split` 的概率，但压缩速度变慢，产生的 CPU 开销也越大。

```
ALTER TABLE t1 COMPRESSION="zlib";
```

```
OPTIMIZE TABLE t1;
```

7.6.2 用例

- 例子 1:

```
ALTER TABLE db2.t2 ADD COLUMN c1 int, ADD COLUMN c2 int;
```

- 例子 2:

```
CREATE TABLE IF NOT EXISTS t2(a INT PRIMARY KEY,b  
CHAR(10))DISTRIBUTED BY RANGE(a)(g1[0,100)); ALTER TABLE t2  
DISTRIBUTED BY RANGE(a)(g1[0,100), g2[100,200));
```

- 例子 3:

```
ALTER TABLE db.t2 MODIFY a TINYINT NOT NULL;
```

- 例子 4:

```
ALTER TABLE db.t2 ADD INDEX (d);
```

- 例子 5:

```
ALTER TABLE t1 ADD COLUMN b VARCHAR(10) NOT NULL AFTER a;
```

- 例子 6:

```
ALTER TABLE t1 ALTER COLUMN c DROP DEFAULT;
```

- 例子 7:

```
ALTER TABLE t1 ALTER COLUMN a SET DEFAULT 10;
```

- 例子 8:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my  
column';
```

- 例子 9:

```
ALTER TABLE t1 MODIFY COLUMN d INT DEFAULT 666;
```

- 例子 10:

```
ALTER TABLE t1 DROP COLUMN d;
```

- 例子 11:

```
ALTER TABLE t1 DROP INDEX idx;
```

- 例子 12:

```
ALTER TABLE t1 RENAME TO t1_new;
```

- 例子 13:

```
ALTER TABLE t1 TRUNCATE PARTITION p0 UPDATE GLOBAL  
INDEXES STORAGE G1;
```

- 例子 14:

```
ALTER TABLE t1 TRUNCATE SUBPARTITION s0 UPDATE GLOBAL  
INDEXES STORAGE G1;
```

- 例子 15:

```
ALTER TABLE t1 CHANGE b a INT NOT NULL;
```

- 例子 16:

```
ALTER TABLE t1 distributed by RANGE(a)(g1 [100,200),g2 [400,500),g3  
[200,300));
```

- 例子 17:

```
ALTER TABLE t1 distributed by LIST(a)(g1 VALUES IN (10,23,50),g2 VALUES IN  
(7,21,60,77),g3 VALUES IN (80,82));
```

- 例子 18:

```
delimiter //
```

```
ALTER TABLE titles1 distributed by
```

```
case pub_id
```

```
when 1 then
```

```
case
```

```
when cust_id<100 then subdistributed by range (title_id)(g1  
[ 100,200),g4[ 200,300));
```

```
else subdistributed by duplicate(g2);
```

```
end case;
```

```
else subdistributed by list(title_id)(g3 values in (10,20,30));
```

```
end case; //
```

```
delimiter ;
```

- 例子 19:

```
delimiter //
```

```
ALTER TABLE titles2 distributed by
```

```
case pub_id
```

```

when 1 then
case
when cust_id<100 then subdistributed by range (title_id)(g1 [ 100,200));
else subdistributed by hash(title_id)(g2);
end case;
else subdistributed by list(title_id)(g3 values in (10,20,30),g4 values in (40,50));
end case; //
delimiter ;

```

- 例子 20:

```
ALTER TABLE db.t2 ADD GLOBAL INDEX g_idx(d);
```

- 例子 21:

```
ALTER TABLE db.t2 ADD GLOBAL UNIQUE INDEX g_u_idx(d);
```

- 例子 22:

```
ALTER TABLE db.tb RENAME INDEX func_idx TO new_func_idx;
```

- 例子 23:

```
ALTER TABLE db.tb RENAME COLUMN old_col_name TO new_col_name;
```

7.6.3 不同分片不同分区管理

可以通过 `alter table add/drop/coalesce/reorganize/remove partitioning storedb` 进行不同分片不同分区表管理。

- 例子 1:

```
ALTER TABLE ps_test.range_x ADD PARTITION (PARTITION p4 VALUES LESS THAN(500)) STORAGEDB G1, G2;
```

- 例子 2:

```
ALTER TABLE ps_test.range_x DROP PARTITION p0 STORAGEDB G1;
```

- 例子 3:

```
ALTER TABLE ps_test.hash_x COALESCE PARTITION 2 STORAGEDB G1;
```

- 例子 4:

```
ALTER TABLE ps_test.range_x REORGANIZE PARTITION p0 INTO
```

```
(
```

```
PARTITION s0 VALUES LESS THAN(50),
```

```
PARTITION s1 VALUES LESS THAN(100)
```


) STORAGEDB G1, G2;

- 例子 5:

ALTER TABLE ps_test.range_x REMOVE PARTITIONING STORAGEDB G1, G2;

7.6.4 使用说明

- 最后一列必须是 GTID 列，且不能变更 GTID 列的任何属性。
- 不支持增加/删除/修改自增列。
- ALTER RANGE 不支持用开闭区间分发形式的 ALTER 语句去修改原有 LESS > THAN 形式创建的表结构。
- ALTER > RANGE 各分片的区间范围不能有交叉、不能修改或删除原有分片范围，不能修改分发键，且区间段数量不超过分片数量最大值限制。
- ALTER > LIST 各分片的区间范围不能有交叉、不能修改或删除原有分片范围，不能修改分发键。
- ALTER > LIST 对原有分片增加 LIST 值，增加的值不在原有分片内的情况下，不改变原有数据分布。
- 对于 alter 多级分片表的限制说明。

1) range 分发:

支持 alter range 带区间形式（左闭右开）增加分片，但分片的区间范围不能有交叉、不能修改或删除原有分片范围、不能修改分发键、区间段数量不超过分片数量最大值限制，否则报错；

不支持修改含 less than 语法的 range 分发，包括 alter 为 range 区间形式或修改原 less than 范围，否则报错；

2) list 分发:

支持修改 list 分发策略增加分片或 list 值，但不能删除原有分片信息，且新增 list 值不能重复，并且不能修改分发键，否则报错；

3) hash/duplicate 分发:

不支持修改原表 hash 或 duplicate 分发策略；

- ALTER RANGE 修改表分发信息支持以下三种操作：(1)增加分片 > (2)已有分片上增加区间段 > (3)已有区间段取值范围扩充。其中，(1)和(2)可以同时进行(一条 alter 语句同时增加分片和增加已有分片上的区间段)，但(3)须独立操作，不能和(1)或者(2)同时进行。

- 对于同一条语句进行多个操作的限制

仅支持同一条语句进行多个 **add column** 操作，不支持同时进行其他操作

当进行多列操作时，仅有所有的 **add** 操作都不指定位置信息的场景为在线操作，不影响业务。

- 对于 **ALTER TABLE DISTRIBUTED BY > ...FORCE** 语句，强制更新表元数据，业务的正确性由业务人员自己保证。
- 对于 **alter** 创建全局[唯一]索引的限制说明。
- 支持表创建好之后新增全局索引、全局唯一索引
- 支持表创建好之后且表中有数据的情况下新增全局索引、全局唯一索引，数据会自动插入索引辅助表中
- 创建全局索引、全局唯一索引时需要给定索引名，不支持不带索引名的情况
- 索引名字已经存在，创建全局索引、全局唯一索引时报错
- 全局[唯一]索引字段不支持 **bigint unsigned, binary, blob, text, set, enum, varbinary**
- 不支持在复制表上创建全局索引、全局唯一索引
- 删除全局索引、全局唯一索引，可使用 **DROP INDEX** 或 **ALTER TABLE DROP INDEX**，与普通索引的删除方法相同
- 创建全局索引、全局唯一索引失败后，需使用【**drop remain index** 全局索引名 on 表名或库名.表名】来清理中间过程的残留（包括数据节点上的索引以及索引辅助表）
- 对于 **ALTER TABLE RENAME COLUMN** 的使用说明。
- 不支持对自增列、分区键的重命名
- 不支持界面下发语句块执行
- 对于 **ALTER TABLE RENAME [INDEX|KEY]** 的使用说明。
- 支持修改普通本地索引、本地唯一索引、普通全局索引、全局唯一索引、函数索引；
- 不支持修改临时表的索引名；
- 不支持界面下发语句块执行；

7.6.5 最佳实践

无

7.7 TRUNCATE TABLE

7.7.1 语法说明

```
TRUNCATE [TABLE] tbl_name [NO PURGE];
```

7.7.2 用例

```
TRUNCATE TABLE db1.t1 NO PURGE;
```

7.7.3 使用说明

TRUNCATE TABLE 操作需在客户端链接上执行，且必须有 CREATE 和 DROP 权限。

7.7.4 最佳实践

无

7.8 DROP INDEX

7.8.1 语法说明

```
DROP INDEX index_name ON tbl_name
```

7.8.2 用例

```
DROP INDEX idx1 ON t1;
```

7.8.3 使用说明

无

7.8.4 最佳实践

无

7.9 CREATE INDEX

7.9.1 语法说明

```
CREATE [GLOBAL] [UNIQUE] INDEX index_name
```

```
ON tbl_name (index_col_name,...)
```

```
[index_type]
```

```
| CREATE GLOBAL INDEX index_name
```

```
ON tbl_name (index_col_name,...)
```

```
| CREATE GLOBAL UNIQUE INDEX index_name
```

ON tbl_name (index_col_name,...)

index_col_name:

col_name [(length)]

index_type:

USING {BTREE | HASH}

7.9.2 用例

- 例子 1:

CREATE INDEX idx1 ON db1.t1(a);

- 例子 2:

CREATE UNIQUE INDEX idx2 ON db1.t2(b);

- 例子 3:

CREATE GLOBAL INDEX idx2 ON db1.t2(b);

- 例子 4:

CREATE GLOBAL UNIQUE INDEX idx2 ON db1.t2(b);

- 例子 5:

CREATE INDEX idx_hash_3 ON db1.t3(c) USING HASH;

7.9.3 使用说明

对于全局[唯一]索引

- 支持表创建好之后新增全局索引、全局唯一索引
- 支持表创建好之后且表中有数据的情况下新增全局索引、全局唯一索引，数据会自动插入索引辅助表中
- 索引名字已经存在，创建全局索引、全局唯一索引时报错
- 全局[唯一]索引字段不支持 `bigint unsigned`, `binary`, `blob`, `text`, `set`, `enum`, `varbinary`
- 不支持在复制表上创建全局索引、全局唯一索引
- 删除全局索引、全局唯一索引，可使用 `DROP INDEX` 或 `ALTER TABLE DROP INDEX`，与普通索引的删除方法相同
- 创建全局索引、全局唯一索引失败后，需使用【`drop remain index` 全局索引名 on 表名或库名.表名】来清理中间过程的残留（包括数据节点上的索引以及索引辅助表）

7.9.4 最佳实践

无

7.10 COPY TABLE

7.10.1 语法说明

实现从 tb1 拷贝数据到 tb2，全分片拷贝。

```
COPY TABLE FROM tb1 TO tb2 STORAGEDB all;
```

实现从 tb1 拷贝数据到 tb2，只 copy 单个分片。

```
COPY TABLE FROM tb1 TO tb2 STORAGEDB g1;
```

7.10.2 使用说明

- 非分区表限制
 - 不支持异构表（存储引擎/索引/字段类型/字段个数/行格式不一致）。
 - 不支持全文索引/外键/非独立表空间/表空间加密/页面大小不一致。
 - 不支持系统表/多表/视图/临时表。
 - 数据节点的用户需要如下权限 ALTER, DROP, RELOAD, LOCK TABLES, > SELECT, CREATE。

- 分区表限制

非分区表的使用 COPY TABLE 的限制同样适用于分区表，除此之外还内包括如下限制：

- 不支持含有子分区的表
- 不支持分区算法/名称/个数/类型/区间不同
- 配置项说明 support_copy_table
 - 动态生效，默认值是 0
 - 需要 copy table 功能时，要在配置项[GENERAL]中添加 support_copy_table=1

7.10.3 最佳实践

无

7.11 CHECK TABLE

7.11.1 语法说明

CHECK TABLE tbl_name [option];

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

7.11.2 用例

- 例子 1:

CHECK TABLE db1.tb1;

- 例子 2:

CHECK TABLE tb1;

- 例子 3:

CHECK TABLE tb1 QUICK;

- 例子 4:

CHECK TABLE tb1 QUICK CHANGED;

7.11.3 使用说明

- 单条 check table 语句只支持 check 单张表。

7.11.4 最佳实践

无

8 DML

8.1 INSERT

8.1.1 语法说明

INSERT INTO tbl_name

[PARTITION number]

(col_name,...)

{VALUES (col_values [,col_values]...),(...),... | select_statement}

[ON DUPLICATE KEY UPDATE

col_name=expr [, col_name=expr] ...]

[SAMEDB]

- 说明:

select_statement 使用参考 SELECT 说明。

8.1.2 用例

- 例子 1:

```
INSERT INTO db1.t1
```

```
(
```

```
int_1,
```

```
integer_1,
```

```
smallint_1,
```

```
tinyint_1,
```

```
mediumint_1,
```

```
bigint_1,
```

```
decimal_1,
```

```
numeric_1,
```

```
float_1,
```

```
double_1,
```

```
date_1,
```

```
timestamp_1,
```

```
datetime_1,
```

```
time_1,
```

```
varchar_1,
```

```
char_1,
```

```
char_2,
```

```
varchar_2,
```

```
text_1,
```

```
binary_1,
```

```
binary_2,
```

```
varbinary_1,
```

```

varbinary_2,
blob_1,
year_1,
tinyint_2,
enum_1,
set_1)
VALUES
(20150310,1073741824,16384,68,419430
4,4611686018427387903,12345678901234567890123456789012345.12345678
901
2345678901234567809,12345678901234567890123456789054321.1234567890
123 45678909876543201,1.123456789,1.123456789,'2014-10-13','2014-10-13
23:05:10','2014-10-13 23:05:10','23:05:10','aaaaaaaaabbbbb','bbb
','aaa','xewer','','hellohellohellohellohellohellohello',
'bbbb','aaaaa2','xxxxx','yyyyyyyy1111',2016,0,5,6);

```

- 例子 2:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
```

```
ON DUPLICATE KEY UPDATE c=c+1;
```

- 例子 3:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6)
```

```
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

- 例子 4:

```
INSERT INTO t1 (a, b)
```

```
SELECT * FROM
```

```
(SELECT c, d FROM t2
```

```
UNION
```

```
SELECT e, f FROM t3) AS dt
```

```
ON DUPLICATE KEY UPDATE b = b + c;
```

8.1.3 使用说明

- ON DUPLICATE KEY UPDATE 的使用约束:

- 表带有全局唯一索引时，不支持 insert 操作带有 ON DUPLICATE KEY UPDATE 子句；
- 带有此子句的操作，不判断数据是否存在，存在的数据不判断活性，不加锁；由于计算节点和存储节点存在延时，且延时会实时变化，所以带有此子句的操作，不建议对复制表使用；
- 在 INSERT VALUES 场景下，配置文件 proxy.ini, 配置段 GENERAL, 配置项 switch_insert_value_on_duplicate_key_update_sw, 默认为 1, 语句的写属性为 SW, 否则报错；配置为 0 时，语句写属性可为 CW；配置项修改后可动态生效；
- 在 INSERT VALUES 场景下，配置文件 proxy.ini, 配置段 GENERAL, 配置项 insert_value_on_duplicate_key_update_usage_level, 默认值为 0, 要求主键和唯一索引必须包含分发键，相同的主键和唯一索引值会分发到同一个节点上，此功能可认为在整个分布式起作用，但是增加主键和唯一索引会影响此功能使用；配置为 1 时，不要求主键和唯一索引包含分发键，只在各个分片上使用此功能；配置项修改后可动态生效；
- 在 Insert values 语句中，语句写属性为 sw, 否则报错处理：
INSERT / INTO db.tb (id1,id2,id3) VALUES(2,2,2) ON DUPLICATE KEY UPDATE id2=2;
- 在 Insert values 语句中，计算节点会判断 insert 和 update 数据是否在同一数据节点，不在同数据节点或判断不出来时会报错；目前分发键字段对应的右值只能为常量值，本身字段或对本身字段引用的 VALUES 函数如：INSERT INTO db.tb (id1,id2,id3) VALUES(2,2,2) ON DUPLICATE KEY UPDATE id2=2; INSERT INTO db.tb (id1,id2,id3) VALUES(2,2,2) ON DUPLICATE KEY UPDATE id2=id2; INSERT INTO db.tb (id1,id2,id3) VALUES(2,2,2) ON DUPLICATE KEY UPDATE id2=VALUES(id2); 注：假设 id2 为 db.tb 表的分发键
- 同一分发键在子句左值多次出现时，最后一次的右值有效。
- 在 Insert select 语句中，对带有分发键的表，ON DUPLICATE KEY UPDATE 子句的左值不能出现分发键字段；右值出现字段时，只能为 insert 表的字段，如果字段不属于 insert 表时，直接报错；如果字段同时出现在 select 表中时，不做进一步校验，默认为 insert 表的字段；此处跟直连 db 操作不同，直连 db 操作时，报歧义错误（ERROR 1052 (23000): Column '***' in field list is ambiguous）。
- 在 Insert select 语句中，唯一索引未包含分发字段存在一个显著的限制：不同数据节点间数据没有唯一性约束。即只能保证其所下压执行

group 上的功能正确性，如果和其它 group 上的数据有冲突或关联，则无法保证功能整体正确性。

- INSERT VALUES 带有同义词时，VALUES 中如果出现字段，不能用同义词修饰。
- INSERT 插入数据时，对于分发键列，需直接输入分发键值，避免使用函数取值，例如：

```
CREATE TABLE t1(id INT KEY, f INT) DISTRIBUTED BY HASH(id,f)(g1,g2,g3);
```

```
INSERT INTO t1(id,f) VALUES(1,1+1); -->失败
```

```
INSERT INTO t1(id,f) VALUES(1,2); -->成功
```

- 指定 group 下发 INSERT 插入数据时，必须指定字段名。

例如：INSERT INTO t1 (id1,id2,id3) VALUES (1,2,3),(4,5,6); 错误示例：
INSERT INTO t1 VALUES (1,2,3),(4,5,6);

8.1.4 最佳实践

INSERT SELECT

针对 INSERT INTO SELECT 语句，考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 插入表和查询表都是单节点存储表且位于同一个数据节点上。
- 插入表是单节点存储，且查询表是复制表。

对需要建二级索引的表，当大批量数据加载操作，建议采用先加载数据后创建二级索引的模式来提升性能，如果已有索引时，在加载数据前需要先删除。

8.2 DELETE

8.2.1 语法说明

```
DELETE FROM tbl_name
```

```
[PARTITION number]
```

```
[WHERE where_condition]
```

```
[ORDER BY ...]
```

```
[LIMIT row_count]
```

```
[CONSISTENCY_OPTION]
```

[STORAGEDB groupno]

[SAMEDB]

CONSISTENCY_OPTION:

[SW | CW]

where_condition:

expr

expr:

expr OR expr

| expr || expr

| expr XOR expr

| expr AND expr

| expr && expr

| NOT expr

| ! expr

| boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}

| boolean_primary

boolean_primary:

boolean_primary IS [NOT] NULL

| boolean_primary <=> predicate

| boolean_primary comparison_operator predicate

| predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:

bit_expr [NOT] IN (subquery)

| bit_expr [NOT] IN (expr [, expr] ...)

| bit_expr [NOT] BETWEEN bit_expr AND predicate

| bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]

| bit_expr

bit_expr:

bit_expr + bit_expr

| bit_expr - bit_expr

| bit_expr * bit_expr

| bit_expr / bit_expr

| bit_expr DIV bit_expr

| bit_expr MOD bit_expr

| bit_expr % bit_expr

| simple_expr

simple_expr:

literal

| identifier

| function_call

| - simple_expr

| ! simple_expr

| (subquery)

| EXISTS (subquery)

8.2.2 用例

- 例子 1:

DELETE FROM somelog WHERE user = 'jcole'

ORDER BY timestamp_column LIMIT 1;

- 例子 2:

DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2 WHERE a1.id=a2.id;

- 例子 3:

DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3

WHERE t1.id=t2.id AND t2.id=t3.id;

- 例子 4:

DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;

8.2.3 使用说明

- DELETE 单表支持 WHERE 带子查询，支持程度同 SELECT WHERE 带子查询。
- 支持 DELETE 单表 FROM 多个表。

如:

DELETE tests123.tb2 FROM tests123.tb2 INNER JOIN tests123.tb1. WHERE
tb2.id=tb1.id;

- 非指定数据节点场景，DELETE 语句只支持 WHERE 带有 SELECT 子句。

8.2.4 限制说明

- 多表语法中，只能指定删除一张表的数据，即 DELETE 后只能跟一张表，FROM 后可以跟多张表关联，但是要删除的表必须在 FROM 表中的第一个。
- 多表语法中，表未指定别名时必须先执行 USE db 语句（MySQL 限制）。

8.2.5 最佳实践

- 为了降低加锁范围，DELETE 语句需要使用索引。
- 由于存储节点数据节点的处理机制问题，当主键或索引为复合列时，WHERE 条件中请避免使用 IN 运算符，建议使用 AND > OR 方式。

举例:

DELETE FROM TABLE WHERE pk1=1 AND pk2=2 OR pk1=3 AND pk2=4;

8.3 UPDATE

8.3.1 语法说明

Single-table:

UPDATE table_reference

SET col_name1=expr [, col_name2=expr ...]

[WHERE where_condition]

[ORDER BY ...]

[LIMIT row_count]

[CONSISTENCY_OPTION]

[STORAGEDB groupno]

[SAMEDB]

multiple-table:

UPDATE table_references

SET col_name1=expr [, col_name2=expr ...]

[WHERE where_condition]

[CONSISTENCY_OPTION]

[STORAGEDB groupno]

CONSISTENCY_OPTION:

[SW | CW]

where_condition:

expr

- 说明：
WHERE 条件限制遵循 SELECT 的限制条件。

支持语法

- UPDATE 分发键=确定值。
- 支持 UPDATE tb SET 多字段功能

UPDATE tb SET (col1,col2)=(SELECT id1,id2 FROM tb2 WHERE ...)

- 左值单列多列均可，右值为带聚合函数的关联子查询语句

update t1_hash set (a_code,b_code)

= (select coalesce(sum(a_code),0),max(b_code) from t1_hash where exists (select 1 from t1_range where t1_hash.infom=t1_range.infom));

- 右值为带聚合函数关联子查询语句的计算表达式

update t1 set a_code = (a_code +(select coalesce(max(d_code),0) from t2 where t1.id=t2.id));

8.3.2 用例

- 例子 1:

UPDATE tb1 SET col1=1 WHERE id=1;

- 例子 2:

UPDATE tb1 SET(col1,col2)=(SELECT id1,id2 FROM tb2 WHERE id=1);

8.3.3 使用说明

- 多表语法中，只能更新第一张表中的列。
- JOIN 场景，遵循 SELECT 限制。
- 非指定数据节点场景，带有 SELECT 子句的 UPDATE，SELECT 子句遵循 SELECT 限制；对于 SELECT 语句有重数限制的，UPDATE 中的 SELECT 语句重数减 1。
- 不支持 UPDATE SET 主键/全局（唯一）索引=函数
- UPDATE 函数对参数执行修改操作，参数不支持主键或全局（唯一）索引
- UPDATE 的 SET 中不支持= AND 逻辑。

Eg: UPDATE tb1 SET id = 1 AND id1=1 或者 UPDATE tb1 SET id = 1 AND 1。

- 不支持 SET 分发键=函数/表达式/自增列/SEQUENCE。
- 不支持分发键=值 STORAGEDB/SAMEDB。
- 不支持全局索引表修改分发键。
- 支持 update tb set 多字段功能说明：

只支持单表更新语句，即 update db.tb set 场景。

语句不支持 order by 和 limit 子句。

只支持一个 select 语句存在 set 子句中，且此 select 子句为对单表的查询语句，不支持 union、join，和通配符*；更新表字段出现在 select 子句中的关联场景，出现聚合函数时支持单列及多列的查询场景。

目前不支持 explain 和 sequence 功能。

- update 带聚合函数的关联子查询语句只支持隔层关联，不支持跨层关联，且不支持带 in/not > in 的嵌套子查询中带聚合函数。

8.3.4 最佳实践

8.3.4.1 关联方式的多表优化

关联方式的多表操作语句如下：

```
UPDATE db.t1 SET aa=1 WHERE EXISTS(SELECT 1 FROM db.t2 WHERE t1.aa=t2.aa)
```

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

更新表和从表都是单节点存储表的且位于同一个数据节点上。

更新表是单节点存储，且从表是复制表。

8.3.4.2 非关联方式的多表优化

非关联方式的多表操作语句如下：

```
UPDATE db.t1 SET aa=1 WHERE EXISTS(SELECT 1 FROM db.t2)
```

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 更新表和子查询表都是单节点存储表，且位于同一个存储节点上。
- 子查询表是复制表。

8.3.4.3 其他优化

- 多表方式复杂度高，比较影响性能，建议尽量使用单表方式的 UPDATE。
- 为了降低加锁范围，UPDATE 语句需要使用索引。
- 由于存储节点数据节点的处理机制问题，当主键或索引为复合列时，WHERE 条件中请避免使用 in 运算符，建议使用 AND、OR 方式。

举例：

```
UPDATE table SET a=2 WHERE pk1=1 AND pk2=2 OR pk1=3 AND pk2=4;
```

8.4 REPLACE

8.4.1 语法说明

```
REPLACE [INTO] tbl_name
```

```
[(col_name,...)]
```

```
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

8.4.2 用例

```
REPLACE INTO table (id,nu) VALUES(1,4);
```


8.4.3 使用说明

无

8.4.4 最佳实践

无

8.5 MERGE INTO

8.5.1 语法说明

MERGE INTO [target-table] A

USING [source-table sql] B

ON([conditional expression] and [...])

WHEN MATCHED THEN

[UPDATE sql [WHERE...]]

WHEN NOT MATCHED THEN

[INSERT sql [WHERE ...]]

8.5.2 用例

- 例子 1:

merge into t1 using t2 on(t1.b=t2.b) when matched then insert (t1.a,t1.b,t1.c) values(t2.a,t2.b,t2.c) where t2.a>0; // 只含有 MATCHED

merge into t1 q using (select a as a1,b as b1,c c1 from t2) p on(q.b=p.b1) when matched then insert (q.a,q.b,q.c) values(p.a1,p.b1,p.c1) where p.a1>0; // USING 子查询

- 例子 2:

merge into t1 using t2 on(t1.b=t2.b) when not matched then insert (t1.a,t1.b,t1.c) values(t2.a,t2.b,t2.c) where t2.a>0; // 只含有 NOT MATCHED

merge into t1 q using (select a as a1,b as b1,c c1 from t2) p on(q.b=p.b1) when not matched then insert (q.a,q.b,q.c) values(p.a1,p.b1,p.c1) where p.a1>0; // USING 子查询

- 例子 3:

merge into t1 using t2 on(t1.b=t2.b) when matched then update set t1.c = t2.c where t2.a>0 when not matched then insert (t1.a,t1.b,t1.c) values(t2.a,t2.b,t2.c) where t2.b>0; // 同时含有 MATCHED 和 NOT MATCHED

merge into t1 q using (select a as a1,b as b1,c c1 from t2) p on(q.b=p.b1) when matched then insert (q.a,q.b,q.c) values(p.a1,p.b1,p.c1) where p.a1>0 when not matched then insert (q.a,q.b,q.c) values(p.a1,p.b1,p.c1) where p.a1>0; // 同时含有 MATCHED 和 NOT MATCHED

8.5.3 限制说明

- merge into 不支持视图。
- 不支持 update + delete 操作。
- 仅含有 MATCHED 的情况，限制同 UPDATE。
- 仅含有 NOT MATCHED 的情况，限制同 INSERT。
- 同时含有 MATCHED 和 NOT MATCHED 的情况，不支持操作全局索引表和修改分发键。

8.5.4 最佳实践

无

9 DQL

9.1 SELECT

9.1.1 语法说明

SELECT

expr , ...

[FROM table_references]

[WHERE where_condition]

[PARTITION number]

[GROUP BY col_name]

[HAVING where_condition]

[ORDER BY col_name [ASC | DESC], ...]

[LIMIT [offset,] row_count]

[FOR UPDATE [OF ...] | LOCK IN SHARE MODE]]

[CONSISTENCY_OPTION]

[STORAGEDB groupno]

[READ_STRATEGY]

[SAMEDB]

READ_STRATEGY:

[READMASTER | READSLAVE | READBALANCE]

table_references:

table_reference [, table_reference] ...

table_reference:

table_factor

| join_table

table_factor:

tbl_name [[AS] alias]

| table_subquery [AS] alias

join_table:

table_reference [INNER] JOIN table_factor [join_condition]

| table_reference {LEFT|RIGHT|CROSS} JOIN table_reference join_condition

join_condition:

ON conditional_expr

where_condition:

expr

同时 GoldenDB 兼容支持 Oracle 在 WHERE 子句中使用"+"的左右连接用法。

左连接示例如下：

```
SELECT * FROM testjoina a, testjoinb b WHERE a.id=b.id1(+);
```

9.1.2 用例

- 例子 1:

```
SELECT CONCAT(last_name,', ',first_name) AS full_name FROM mytable ORDER  
BY full_name;
```

- 例子 2:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2 WHERE t1.name = t2.name;
```

- 例子 3:

```
SELECT college, region, seed FROM tournament ORDER BY region, seed;
```

- 例子 4:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- 例子 5:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

- 例子 6:

```
SELECT user, MAX(salary) FROM users GROUP BY user HAVING MAX(salary) > 10;
```

- 例子 7:

```
SELECT * FROM tbl LIMIT 5,10;
```

- 例子 8:

```
SELECT 12 AS a, a FROM t GROUP BY a STORAGEDB g1;
```

- 例子 9:

```
SELECT college, region, seed FROM tournament ORDER BY region, seed FOR UPDATE;
```

9.1.3 使用说明

9.1.3.1 SELECT LIST

- 不支持关系表达式（AND/OR/&&/||之类）。
- 不支持 SOME/ANY/ALL 操作。
- SELECT LIST 中不支持自定义函数与其他函数或常量的嵌套组合。
- 查询字段最大数为 2048 个，列名称或别名最大长度为 256 字节。
- 涉及多表 sql 语句中，select list 中可以含有二义性字段，该字段自动拼接第一个匹配上的表的名字，不会报错。如下（a 是 t1、t2 同名字段）。

```
select a from t1 join t2 on t1.b1 = t2.b2;
```

9.1.3.2 FUNCTION

- COUNT 后面支持 DISTINCT 单个表达式列表的语法。

如：

```
SELECT COUNT(DISTINCT a) FROM tests123.tb1;
```

- 聚合函数中含有表达式、表达式中含有聚合函数时精度和数据节点不一致，聚合函数中有 FLOAT、DOUBLE，只能保证有效精度范围之内的结果和数据节点一致。
- 汇聚函数中不能嵌套汇聚函数。
- SUM、AVG 需使用数值类型(整型/浮点/DECIMAL)。
- GROUP_CONCAT 目前带 DISTINCT 去重存在问题（示例：SELECT > GROUP_CONCAT(DISTINCT id) FROM t1），但带有 DISTINCT 和 ORDER > BY 的场景是支持的（示例：SELECT GROUP_CONCAT(DISTINCT id ORDER BY > id) FROM t1）。

GROUP_CONCAT 列最大字段长度为 1k，GROUP_CONCAT 含有列个数最多支持 16 个。

9.1.3.3 WHERE

- SELECT 表达式中支持= SOME/= ANY/<> ALL 非直接下压情况，= SOME/= ANY/<> ALL 限制同 IN/NOT > IN，SOME/ANY/ALL 范围比较不支持。
- 不支持 WHERE SOME/ANY/ALL 子查询执行 JOIN 操作。

9.1.3.4 data type

BLOB、TEXT 系列类型只能单独出现在最顶层 SELECT LIST 中。

9.1.3.5 多表

- 支持 SELECT ... FROM t1,t2,t3... WHERE > ...，限制场景同 JOIN 外，还增加 WHERE 条件中不能有 OR。
- UNION/UNION ALL 语句的合并暂时支持最大只有一张非复制表场景。
- 多表查询中，如果多表有多个相同的字段名，字段被添加的 db、tb 会是他遇到的第一个库、表；因此这种情况，建议手动添加上 db、tb。
- 只支持 FROM、SELECT LIST 和 WHERE 子查询，且 FROM 子查询最多嵌套 10 层。
- SELECT LIST 中含有子查询时存在以下约束：
 - 在 CR 场景下目前只支持 single row 类型的子查询。
 - 子查询为非 single row 类型时，不支持主子查询不能合。
 - 不支持含有 Oracle 函数。

- 不支持需要在计算节点层做 WHERE 和 HAVING 过滤的 SELECT LIST 子查询。
- 子查询支持 EXISTS/NOT EXISTS/IN/NOT IN/比较运算符。
- 支持等值条件的关联查询，即直接的等值操作，如 db.t1.a=db.t2.a，其它场景不支持。
- IN/NOT IN WHERE 子查询不支持有 LIMIT 且不能合并场景。
- FLOAT，DOUBLE 类型字段出现 WHERE > 子查询条件中且不能合并时，只能是 EXISTS 子查询。

9.1.3.6 select ... reads slave

- 不论是否开启读写分离，带 reads slave 的 select 语句，满足读写分离条件的情况下，优先选择备 db 下发

9.1.3.7 其它

- ORDER BY，GROUP BY，HAVING 支持 SELECT LIST 中的字段别名，JOIN > ON 不支持 SELECT LIST 中的字段别名。

如：

```
SELECT t1.id AS ID,t2.id AS TB FROM t1 JOIN t2 on ID=TB;
```

- GROUP BY 后不支持聚合函数/常量。
- 不支持聚合函数别名与字段名相同且该字段或同名字段位于 GROUP > BY 中的场景。

如：

```
SELECT SUM(t1.b) AS b FROM t1,t4 GROUP BY t4.b;
```

- ORDER BY 后面的每个字段结果的长度不能超过 32；
- ORDER BY > 需要计算节点重新排序(数据节点排好序后计算节点在此基础上再排序不包括在内)的最大列数不能超过 30 列(31 列开始不支持)。
- ORDER BY/GROUP BY 中字段必须出现在 FROM 子查询的 SELECT LIST 中。
- having 后面不支持 BLOB 类型。
- having 后的表字段如果不是聚合函数参数，则需要在查询字段中出现。
- 系统变量只支持变量本身查询，不支持四则运算与函数，不支持 UPDATE、DELETE、INSERT，变量不支持写操作，例如 SELECT > id INTO @a FROM t1。

- 支持：

SET@a=0; SET@b=1

SELECT @a,@b FROM t1;

- 不支持：

SELECT @a+1 FROM t1;

SELECT @a+@b FROM t1;

SELECT abs(@a) FROM t1;

- 不支持 SELECT...FROM 子查询 FOR UPDATE OF

- 视图

- 视图不支持 SELECT *;
- 视图不支持 SELECT NULL;
- 视图不支持子查询;
- 分布式视图(support_distribute_view = 1)不支持 DML 操作;

- DISTINCT

DISTINCT 对字符串尾部空格不删除;

- 计算节点对行记录大小 1M 的使用说明

- 当查询结果需要在计算节点层做二次处理的时候，比如跨分片 JOIN、ORDER > BY 等场景下，为了防止结果集以及单行记录过大，计算节点计算耗时过长导致查询语句长时间挂住，计算节点对结果集行记录进行了 1M 的大小限制：如果单行记录长度超过 1M，则计算节点报错。
- 当 SELECT 语句仅涉及到单 group，比如复制表或者 WHERE > 分发键=具体值（能计算出具体分片），计算节点不需要对结果集进行二次处理，没有行记录大小 1M 的限制；
- 1M 是固定限制，计算节点无相关配置；

- start with ... connect by ...

- from 的表是只能是单表；
- 不支持参数：nocycle, connect_by_iscycle, connect_by_isleaf, level;

- connect by 后面条件，多分片只支持 列=列，不支持 > 列=变量、表达式、函数；单分片无此限制；
- 语法顺序只支持：start with connect by，顺序调换不支持；
- 不支持 start with 嵌入在关联子查询中；
- 不支持多级分片；
- 多分片不支持视图；
- 结果顺序跟 oracle 不一样，结果集一样；

- SELECT PROXY_VERSION(); 查看 CN 版本号

9.1.4 最佳实践

9.1.4.1 子查询的优化

- FROM 子查询

语句如下：

```
SELECT sb1,sb2,sb3
```

```
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM db.t1) AS sb
```

```
WHERE sb1 > 1;
```

语句可以优化转换，去掉子查询，语句转换为：

```
SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM db.t1 WHERE sb1 > 1;
```

- 关联子查询

语句如下：

```
SELECT col1,col2,col3 FROM db.t1 WHERE col1 IN
```

```
(SELECT col1 FROM db. t2 WHERE db.t2.col2 = db.t1.col2);
```

```
SELECT col1,col2,col3 FROM db.t1 WHERE EXISTS
```

```
(SELECT col1 FROM db.t2 WHERE db.t2.col2 = db.t1.col2)
```

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 主表和子查询表都是单节点存储表，且位于同一个 DB 节点上。
- 主表是单节点存储，且子查询表是复制表。
- 非关联子查询

语句如下：

`SELECT col1 FROM db.t1 WHERE EXISTS (SELECT col1 FROM db.t2);`

`SELECT s1 FROM db.t1 WHERE s1 IN (SELECT s1 FROM db.t2);`

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 主表和子查询表都是单节点存储表，且位于同一个 DB 节点上。
- 子查询表是复制表。
- 子查询含有 DISTINCT

语句如下：

`SELECT col1,col2,col3 FROM db.t1 WHERE col2 IN (SELECT DISTINCT col2 FROM db.t2);`

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 主表和子查询表都是单节点存储表的，且位于同一个 DB 节点上。
- 子查询表是复制表。
- 子查询含有分组聚合

语句如下：

`SELECT col2 FROM db.t1 WHERE col1 IN (SELECT SUM(col1) FROM db.t2);`

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 主表和子查询表都是单节点存储表的，且位于同一个 DB 节点上。
- 子查询表是复制表。
- 带聚合函数的关联子查询在 SELECT WHERE 中

语句如下：

`select * from t1_hash where a_code > (select count(c_code) from t1_range where t1_hash.infom=t1_range.infom);`

具体支持情况如下：

- 1) 只支持 sum、max、min、count，暂不支持其他聚合函数。
- 2) 关联条件支持使用 and 拼接多个条件，但只支持等值关联方式。
- 3) select where 带聚合函数的有关联条件的嵌套子查询支持关联多级分片类型的表，支持不同隔离级别 sw/cw ur/cr，支持相同/不同分发类型的关联表。
- 4) 带聚合函数的有关联条件的嵌套子查询支持多层嵌套，但关联条件仅出现在相邻两层，不支持隔层表关联等值条件。

5) 不支持带 in/not in 的嵌套子查询中带聚合函数。

- 带聚合函数的关联子查询在计算表达式中
语句如下：

```
select a_code - (select min(d_code) from t1 where t1.id=t2.id) from t2;
```

具体支持情况如下：

- 1) 只支持 sum、max、min、count，暂不支持其他聚合函数。
- 2) 关联条件支持使用 and 拼接多个条件，但只支持等值关联方式。
- 3) 表达式支持+、-、*、/、mod、div。

9.1.4.2 JOIN 的优化

- 使用说明

JOIN 的规避方法：

一般情况下 JOIN 复杂度和执行效率上都比简单查询的代价要高，建议在表的模型设计时，采用去范式化手段把 JOIN 转换为简单查询。

- 多表关联：关联查询的表个数不能超过 61 个。
- LEFT JOIN/RIGHT JOIN 能合并的场景，第一张外表不能是复制表；
- 不支持 natural JOIN 语法。

如：

```
SELECT tests123.tb1.id FROM tests123.tb1 natural RIGHT JOIN tests123.tb2;
```

- 不支持 on 条件里存在子查询。

如：

```
SELECT id FROM t1 JOIN t2 on t1.id=(SELECT 1 FROM t2);
```

```
SELECT * FROM t1 LEFT JOIN t2 on t1.id=t2.id AND t1.id1=(SELECT id FROM t3  
LIMIT 1);
```

- 多表 JOIN 的用法：当前关联表名必须在其后的 on 条件中出现。
- 大小表的判断标准：

小表和大表的区分不是根据表的原始数据来判断的，而是根据条件过滤后的数据来判断。

- 小表和小表之间两表关联

考虑到语句的执行效率，在表设计时建议优先考虑下面几种模型：

- 两个表都是单节点存储表的，且位于同一个 DB 节点上。

- 存在一个表是复制表。
- 小表和大表的关联
考虑到语句的执行效率，在表设计时建议优先考虑：大表采用多节点分布存储，小表采用复制表。

9.1.4.3 分组聚合的优化

考虑到语句的执行效率，在表设计时建议优先考虑采用单节点存储方式，或复制表方式。

两个限制：

- COUNT(DISTINCT)函数会导致网络间大量的网络传输，性能较差。
- 在非脏读隔离级别下，查询中带聚合函数时，如果数据存在读和写并发的情况，可能会出现聚合函数计算不准确的问题，即出现脏读，业务需要慎重考虑此场景的影响。

9.1.4.4 分页的优化

- 传统分页

```
SELECT db.table.col2 FROM db.table LIMIT 10000,10;
```

- 说明：
偏移量越大则越慢。

- LIMIT 方式推荐分页

- 方式一：

```
SELECT col1 FROM db.table WHERE id>=23423 LIMIT 10; # (每页 10 条)
```

```
SELECT col1 FROM db.table WHERE id>=23434 LIMIT 10;
```

- 说明：
- 只适合包含单增类型字段的场景。
- 如果存在 ORDER BY 时，排序字段中需要包含此单增类型字段。

- 方式二：

```
SELECT id FROM table WHERE LIMIT 10000,10;
```

```
SELECT col FROM table WHERE id IN ('123',456...);
```

- 说明：
- id 是指主键或者唯一索引。
- 如果存在 ORDER BY 时，排序字段中需要包含此字段。

- LIMIT 分页与缓存结合

先一次性查询多页记录数据缓存到内存，业务程序实现分页功能。

9.2 UNION

9.2.1 语法说明

SELECT ...

UNION [ALL] SELECT ...

[UNION [ALL] SELECT ...]

9.2.2 用例

- 例子 1:

(SELECT a FROM t1 WHERE a=10 AND b=1 ORDER BY a LIMIT 10)

UNION

(SELECT a FROM t2 WHERE a=11 AND b=2 ORDER BY a LIMIT 10);

- 例子 2:

SELECT column_name(s) FROM table_name1

UNION ALL

SELECT column_name(s) FROM table_name2

9.2.3 使用说明

- UNION 语句对于跨类型的连续多个 UNION 操作不保证结果准确。
- UNION 语句不支持 BLOB 和 TEXT 及相关的大字段类型。
- 对于应用在 UNION 语句前的应用 ORDER BY 的 SELECT 语句，应在添加括号后使用，如用例 1。

9.2.4 最佳实践

无

10 VIEW

10.1 语法说明

10.1.1 CREATA VIEW

CREATE

[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH CHECK OPTION]

10.1.2 ALTER VIEW

ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH CHECK OPTION]

10.1.3 DROP VIEW

DROP VIEW [IF EXISTS]
view_name [, view_name] ...
[RESTRICT | CASCADE]

10.2 用例

- 例子 1:

CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;

- 例子 2

ALTER VIEW view_students_info AS SELECT id,name,age FROM tb_students_info;

- 例子 3:

DROP VIEW view_students_info;

10.3 使用说明

计算节点支持不参与对结果集进行二次处理(简单合并可以)的视图。

如：

```
CREATE VIEW view_name AS SELECT c FROM tb1 ORDER BY tb1.a;
```

10.4 最佳实践

无

11 TRIGGER

11.1 语法说明

CREATE TRIGGER 语法

CREATE

TRIGGER trigger_name

trigger_time trigger_event

ON tbl_name FOR EACH ROW

trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

DROP TRIGGER 语法

DROP TRIGGER trigger_name;

11.2 用例

```
CREATE TRIGGER testref BEFORE INSERT ON test1
```

```
FOR EACH ROW BEGIN
```

```
DELETE FROM test3 WHERE a3 = NEW.a1;
```

```
UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
```

```
END;
```

11.3 使用说明

- 不支持跨存储节点的触发器，即基础语句和激活的语句应能保证在同一存储节点执行。
- 不支持利用 DEFINER 关键字自定义特定用户有权限执行语句。

如：

```
CREATE DEFINER=CURRENT_USER
```

```
TRIGGER test_trigger1 BEFORE DELETE ON t1
```

```
FOR EACH ROW BEGIN
```

```
INSERT INTO t2 (id, name, info) VALUES (OLD.id, OLD.name, OLD.info);
```

```
END;
```

- 若 trigger_body 中有 INSERT 语句，必须显示指明所插入的列名。

如：

```
CREATE TRIGGER test_trigger2 AFTER UPDATE ON t3
```

```
FOR EACH ROW BEGIN
```

```
INSERT INTO t4 (id, name) VALUES (NEW.id, NEW.name);
```

```
END;
```

11.4 最佳实践

无

12 TRANSACTION AND LOCKING STATEMENTS

12.1 TRANSACTION

12.1.1 语法说明

```
START TRANSACTION [CONSISTENCY_OPTION][SAMEDB]
```

```
COMMIT
```

```
ROLLBACK
```

说明：

- START TRANSACTION 语句可以开始一项新的事务。

- COMMIT 可以提交当前事务，是变更成为永久变更。
- ROLLBACK 可以回滚当前事务，取消其变更。
- 如果一致性级别设置在事务和 SQL 语句中互相冲突时，以 SQL 语句中一致性级别为准。

12.1.2 使用说明

不支持 SET TRANSACTION 语法。

如：

```
SET TRANSACTION READ ONLY;
```

12.1.3 最佳实践

数据一致性级别对性能影响

数据一致性级别要求越高时，对事务的性能影响越大。

12.2 SAVEPOINT

12.2.1 语法说明

12.2.1.1 SAVEPOINT

```
SAVEPOINT identifier
```

12.2.1.2 ROLLBACK TO SAVEPOINT

```
ROLLBACK TO [SAVEPOINT] identifier
```

12.2.1.3 RELEASE SAVEPOINT

```
RELEASE SAVEPOINT identifier
```

12.2.2 用例

```
DECLARE
```

```
v_number number;
```

```
BEGIN
```

```
v_number := 1;
```

```
INSERT INTO t11 VALUES(T11_SEQ.nextval,v_number);
```

```
savepoint A;
```

```
INSERT INTO t11 VALUES(T11_SEQ.nextval,v_number+1);
```



```
savepoint B;  
INSERT INTO t11 VALUES(T11_SEQ.nextval,v_number+2);  
savepoint C;  
ROLLBACK TO B;  
END;
```

12.2.3 使用说明

无

12.2.4 最佳实践

无

13 PREPARE

13.1 语法说明

GoldenDB 支持 Prepared Statements 语法，提升 SQL 处理效率，同时可实现数据与逻辑分离，同时结合 fetchSize 功能，可以满足应用的批处理需求。

语法说明如下：

```
PREPARE stmt_name FROM preparable_stmt  
EXECUTE stmt_name [USING @var_name [, @var_name] ...]  
{DEALLOCATE | DROP} PREPARE stmt_name
```

13.2 用例

```
PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';  
SET @a= 3;  
SET @b = 4;  
EXECUTE stmt1 USING @a, @b;  
DEALLOCATE PREPARE stmt1;
```

13.3 使用说明

- 目前支持 SELECT 语句的 Prepared 语句。

- 应用可以结合 `fetchSize` 取固定值功能，将结果集缓存在 GoldenDB 数据库，客户端根据设置的固定值依次从服务端获取结果集。
- `prepare` 不支持的格式：GROUP BY ?/ ORDER BY ?/ LIMIT ?/ WHERE ?= 。

13.4 最佳实践

无

14 SYNONYM

14.1 语法说明

14.1.1 CREATE SYNONYM

CREATE [PUBLIC] SYNONYM [schema.] synonym_name

FOR [schema.] object

14.1.1 DROP SYNONYM

DROP [PUBLIC] SYNONYM [schema .] synonym [FORCE]

14.2 用例

- 例子 1:

CREATE [PUBLIC] SYNONYM syn_abc FOR db1.tb1;

- 例子 2:

DROP [PUBLIC] SYNONYM syn_abc;

14.3 使用说明

- 暂不支持跨库同义词（解决方法：在输入 SQL 查询语句时不用带上同义词的库名）。
- 多个跨库同义词不要同名。

14.4 最佳实践

无

15 数据库管理语句

15.1 语法说明

15.1.1 账户管理语法

```
• CREATE USER
CREATE USER [IF NOT EXISTS]
user auth_option
[password_option] ...
auth_option: {
IDENTIFIED BY 'auth_string' }
password_option:
{ PASSWORD EXPIRE [DEFAULT | NEVER | INTERVAL N DAY]
| PASSWORD HISTORY {DEFAULT | N}
| PASSWORD REUSE INTERVAL {DEFAULT | N DAY} }
```

注：建议通过 insight 页面创建用户并进行管理

表 15-1 user 名

用户名	是否支持
'user_name' @'10.10.10.10'	是
'user_name' @'10.10.10.10/8'	是
'user_name' @'10.10.10.10/255.255.255.0'	不支持，与 / 功能重复
'user_name' @'10.10.10.%'	是
'user_name' @'8888:FF:FF:FF:FF:FF:FF:1111'	' 不支持
'user_name' @'8888::1111'	不支持，ipv6 不支持
'user_name' @'8888:0:0:0:0:0:0:%'	不支持，ipv6 不支持
'user_name' @'%'	是
'user_name' @ 'localhost'	不支持
'user_name' @ '127.0.0.1'	是
'user_name' @ '::1'	不支持，ipv6 不支持
'user_name' @ '%.mysql.com'	不支持
多个 user	否，暂时只支持一个 user

表 15-2 password_option

选项	备注
PASSWORD EXPIRE [DEFAULT NEVER INTERVAL N DAY]	密码到期选项
PASSWORD HISTORY {DEFAULT N}	密码重用选项，更改次数
PASSWORD REUSE INTERVAL {DEFAULT N DAY}	} 密码重用选项，时间
<ul style="list-style-type: none">ALTER USER ALTER USER [IF EXISTS]	
user [auth_option]	
[password_option] ...	
auth_option: {	
IDENTIFIED BY 'auth_string' }	
password_option: {	
{ PASSWORD EXPIRE [DEFAULT NEVER INTERVAL N DAY]	
PASSWORD HISTORY {DEFAULT N}	
PASSWORD REUSE INTERVAL {DEFAULT N DAY} }	
<ul style="list-style-type: none">DROP USER DROP USER user	
<ul style="list-style-type: none">RENAME USER RENAME USER old_user TO new_user	
注：只支持修改 ip，不支持修改用户名，不支持多个用户	
<ul style="list-style-type: none">GRANT GRANT	
priv_type	
[, priv_type] ...	
ON [object_type] priv_level	
TO user	
[WITH GRANT OPTION]	
}	

```

object_type: {
TABLE
| FUNCTION
| PROCEDURE
}
priv_level: {
*
| *.*
| db_name.*
| db_name.tbl_name
| tbl_name
| db_name.routine_name
}

    • REVOKE
REVOKE
priv_type
[ priv_type] ...
ON [object_type] priv_level
FROM user

REVOKE ALL [PRIVILEGES], GRANT OPTION
FROM user

```

15.1.2 账户管理用例

```

CREATE USER 'jeffrey'@'%' IDENTIFIED BY 'MYPASS@test01';

GRANT ALL ON db1.* TO 'jeffrey'@'%';

REVOKE INSERT ON *.* FROM 'jeffrey'@'%';

DROP USER 'jeffrey'@'%';

```

15.1.3 SHOW 语法

- SHOW DATABASES

SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']

- SHOW CREATE DATABASE

SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name

- SHOW TABLES

SHOW TABLES [{FROM | IN} db_name]

- SHOW CREATE TABLE

SHOW CREATE TABLE tbl_name

- 说明:

输出表结构信息，同时携带计算节点扩展的表分布信息。

- SHOW CREATE PROCEDURE

SHOW CREATE PROCEDURE proc_name

- SHOW CREATE FUNCTION

SHOW CREATE FUNCTION func_name

- SHOW CREATE VIEW

SHOW CREATE VIEW view_name

- SHOW DISTRIBUTION FROM table_name

SHOW DISTRIBUTION FROM showdb.zte

- SHOW VARIABLES

SHOW VARIABLES

SHOW VARIABLES LIKE '%max_connect%';

SHOW SESSION VARIABLES LIKE '%max_connect%';

SHOW GLOBAL VARIABLES LIKE '%max_connect%';

- SHOW WARNINGS

SHOW WARNINGS;

SHOW COUNT(*) WARNINGS;

- SHOW STATUS

SHOW STATUS

SHOW STATUS LIKE '%show%';

SHOW SESSION STATUS LIKE '%show%';

SHOW GLOBAL STATUS LIKE '%show%';

SHOW PLUGINS

SHOW FUNCTION STATUS LIKE '%func%'

SHOW PROCEDURE STATUS LIKE '%proc%'

SHOW TABLE STATUS IN showdb

SHOW TABLE STATUS FROM showdb;

SHOW TABLE STATUS IN showdb LIKE '%zte%';

- 元数据类

1、

SHOW CHARACTER SET

SHOW CHARACTER SET LIKE '%char%';

2、

SHOW COLLATION

SHOW COLLATION LIKE '%utf8%';

3、

SHOW CREATE PROCEDURE

4、

SHOW CREATE FUNCTION

5、

SHOW CREATE TRIGGER

6、

SHOW DISTRIBUTION FROM showdb.zte

SHOW DISTRIBUTION FROM showdb.zte WHERE id=xx

7、

SHOW ENGINES

8、

SHOW KEYS FROM showdb.zte

SHOW KEYS FROM FROM zte FROM showdb;

9、

SHOW INDEX FROM showdb.zte;

SHOW INDEX IN zte FROM showdb;

10、

SHOW COLUMNS FROM zte FROM showdb;

SHOW COLUMNS FROM showdb.zte;

11、

SHOW FIELDS FROM ztet1 FROM showdb;

SHOW FIELDS FROM showdb.zte;

- SHOW GROUPS

show groups g1,g2;

show master groups g1,g2;

show groups ALL;

show master groups ALL;

15.1.4 SHOW 用例

mysql> SHOW DATABASES;

+-----+

| DATABASE |

+-----+

| testdb |

| testdb_sj2 |

| zhl |

| zz |

+-----+

4 rows in set (0.00 sec)


```
mysql> SHOW CREATE DATABASE testdb;
```

```
+---+-----+-----+
+-----+
```

```
| DATABASE | CREATE DATABASE |
```

```
+---+-----+-----+
+-----+
```

```
| testdb | CREATE DATABASE `testdb` /*!40100 DEFAULT CHARACTER SET utf8 */
|
```

```
+---+-----+-----+
+-----+
```

```
1 row in set (0.01 sec)
```

```
mysql> SHOW TABLES;
```

```
+-----+
```

```
| Tables_in_test1 |
```

```
+-----+
```

```
| blobtest |
```

```
| de |
```

```
+-----+
```

```
2 rows in set (0.00 sec)
```

```
mysql> SHOW CREATE TABLE zhl.t1;
```

```
***** 1. row *****
```

```
Table: t1 识别
```

```
CREATE Table: CREATE TABLE `zhl`.`t1` (
```

```
`col1` INT(11) DEFAULT NULL,
```

```
`col2` char(5) DEFAULT NULL,
```

```
`col3` DATE DEFAULT NULL,
```

```
`wp` INT(11) DEFAULT NULL
```

```
)
```

```
/*!50100 PARTITION BY HASH ( YEAR(col3)) PARTITIONS 4 */
```

```
DISTRIBUTED BY HASH(col1) (g1,g2);
```

```
1 row in set (0.02 sec)
```

```
SHOW CREATE PROCEDURE simpleproc\G;
```

```
***** 1. row *****
```

```
Procedure: simpleproc
```

```
sql_mode: STRICT_TRANS_TABLES
```

```
Create Procedure: CREATE DEFINER=`root`@`zte-179` PROCEDURE  
`simpleproc`( IN a INT,OUT param1 INT)
```

```
BEGIN
```

```
INSERT INTO t VALUES(@a);
```

```
SELECT COUNT(*) INTO param1 FROM t;
```

```
END
```

```
DISTRIBUTED BY DUPLICATE (g1,g2)
```

```
character_set_client: utf8
```

```
collation_connection: utf8_bin
```

```
Database Collation: utf8_bin
```

```
SHOW CREATE function hello\G;
```

```
***** 1. row *****
```

```
Function: hello
```

```
sql_mode: STRICT_TRANS_TABLES
```

```
Create Function: CREATE DEFINER=`root`@`localhost` FUNCTION `hello`(s  
CHAR(20)) RETURNS CHAR(50) CHARSET utf8 COLLATE utf8_bin DETERMINISTIC
```

```
RETURN CONCAT('Hello, ',s,'!')
```

```
character_set_client: latin1
```

```
collation_connection: latin1_swedish_ci
```

```
Database Collation: utf8_bin
```

```
MySQL [(none)]> show groups g1,g2;
```

```
+-----  
---+-----+-----+-----+  
| GroupID | DBIP | DBPORT | IsMaster |  
+-----  
---+-----+-----+-----+  
| g1 | 10.229.31.54 | 5501 | 1 |  
| g2 | 10.229.31.54 | 5502 | 1 |
```

```
+-----  
---+-----+-----+-----+  
MySQL [(none)]> show master groups g1,g2;
```

```
+-----  
---+-----+-----+-----+  
| GroupID | DBIP | DBPORT | IsMaster |  
+-----  
---+-----+-----+-----+  
| g1 | 10.229.31.54 | 5501 | 1 |  
| g2 | 10.229.31.54 | 5502 | 1 |
```

```
+-----  
---+-----+-----+-----+  
MySQL [(none)]> show groups all;
```

```
+-----  
---+-----+-----+-----+  
| GroupID | DBIP | DBPORT | IsMaster |  
+-----  
---+-----+-----+-----+  
| g1 | 10.229.31.54 | 5501 | 1 |  
| g2 | 10.229.31.54 | 5502 | 1 |  
| g3 | 10.229.31.54 | 5503 | 1 |  
| g4 | 10.229.31.54 | 5504 | 1 |
```

```
+-----+
---+-----+-----+-----+
```

15.1.5 SET 命令

SET variable_assignment [, variable_assignment] ...

variable_assignment:

user_var_name = exp

15.1.6 SET 用例

SET @a=1;

select @a;

15.1.7 DESC 命令

{ DESCRIBE | DESC } tbl_name

15.1.8 DESC 用例

mysql> desc zhl.t1;

```
+-----+
-----+-----+-----+-----+

| Field | Type | Null | Key | Default | Extra |
+-----+
-----+-----+-----+-----+

| col1 | INT(11) | YES | | NULL | |
| col2 | CHAR(5) | YES | | NULL | |
| col3 | DATE | YES | | NULL | |
| wp | INT(11) | YES | | NULL | |
+-----+
-----+-----+-----+-----+
```

15.1.9 碎片整理

ALTER TABLE t2 ENGINE=innodb, ALGORITHM=INPLACE, LOCK=NONE

当使用磁盘碎片整理功能时，可能会导致主备大量延时，影响故障切换。

15.2 使用说明

proxy 不支持 set passwd

15.3 最佳实践

无

16 实用语法

16.1 EXPLAIN

16.1.1 语法说明

EXPLAIN [DB|PROXY|explain_format] explainable_stmt [storagedb]

explainable_stmt: {

SELECT statement

| DELETE statement

| INSERT statement

| UPDATE statement

}

explain_format: {

FORMAT = JSON

}

16.1.2 用例

- 例子 1:

EXPLAIN SELECT * FROM t1 WHERE id=1 or id=11 ORDER BY id;

- 例子 2:

EXPLAIN PROXY SELECT * FROM t1 WHERE id=1 or id=11 ORDER BY id;

- 例子 3:

EXPLAIN DB SELECT * FROM t1 WHERE id=1 or id=11 ORDER BY id STORAGEDB g2;

- 例子 4:

EXPLAIN FORMAT = JSON SELECT * FROM t1 WHERE id=1 or id=11 ORDER BY id;

16.1.3 使用说明

- 支持仅输出计算节点的分布式执行计划。此时需使用 **PROXY** 选项，即 **EXPLAIN > PROXY explainable_stmt**。
- 支持将分布式执行计划和 **DB** 执行计划结合输出。
- 支持仅输出 **DB** 的执行计划。

此时需使用 **DB** 选项，并指明要查看的存储节点。

示例：

```
EXPLAIN DB SELECT * FROM t1 STORAGEDB g1;
```

- **explain** 目前不支持的场景：
update 操作，set 带有 select 子句时，不支持 **explain** 功能：
explain update db.t1 set t1.id2 = (select t2.id2 from t2);
ERROR 11269 (HY000): ERR: the statement does not support the explain operation for one right value with subselect!
insert、update 和 delete 操作，语句中带有 sequence 时，不支持 **explain** 功能：
explain insert into seq.sing1 (a,b,c) values (seq.seq_t.NEXTVAL,'1',1);
ERROR 11251 (HY000): ERR: the statement with the sequence does not support the explain operation!

16.1.4 最佳实践

无

16.2 USE

16.2.1 语法说明

USE db_name

- 说明：
将 db_name 变为后续执行 SQL 语句的默认库。

16.2.2 用例

```
USE db1;
```

```
SELECT COUNT(*) FROM mytable;
```

16.2.3 使用说明

无

16.2.4 最佳实践

无

16.3 外部流水号设置

16.3.1 语法说明

```
SET @transaction_serial_number="xxx";
```

- 说明：

transaction_serial_number 为 GoldenDB 特有，请勿把它当作普通变量使用。

16.3.2 用例

```
SET autocommit=0;
```

```
SET @transaction_serial_number="xxx";
```

```
UPDATE....
```

```
INSERT...
```

```
COMMIT;
```

16.3.3 使用说明

无

16.3.4 最佳实践

无

16.4 IGNORE_SPACE

16.4.1 语法说明

计算节点 parse mode 设置为 Oracle 模式，数据节点的 SQL Mode 添加 IGNORE_SPACE 模式，将允许函数名和字符'('间带空格。

该模式下，内置函数名被视为保留字，在使用与内置函数同名的标识符时需要用反引号'`'。

16.4.2 用例

例如聚合函数 COUNT(), 如果使用 COUNT 作为表名，建表时将报如下错误：

```
mysql> CREATE TABLE COUNT (i INT) DISTRIBUTED BY DUPLICATE(g1,g2);;
```

ERROR 1064 (42000): You have an error in your SQL syntax

表名必须使用反引号`：

```
mysql> CREATE TABLE `COUNT` (i INT ,PRIMARY KEY(i)) DISTRIBUTED BY  
DUPLICATE(g1,g2);;
```

Query OK, 0 rows affected (0.00 sec)

不需要此功能时，不建议配置 IGNORE_SPACE 模式。

16.4.3 最佳实践

无

16.5 PURGE

16.5.1 语法说明

16.5.1.1 PURGE TABLE

```
PURGE TABLE table_name [FOR RECYCLEBIN_VERSION version];
```

16.5.1.2 PURGE RECYCLEBIN

```
PURGE RECYCLEBIN;
```

16.5.2 用例

PURGE TABLE 不带版本号，删除该表所有的闪回副本。

```
PURGE TABLE db.tb1;
```

PURGE TABLE 带版本号，删除该表对应版本的闪回副本。

```
PURGE TABLE db.tb1 FOR RECYCLEBIN_VERSION "version_info";
```

PURGE RECYCLEBIN 清除所有表的所有闪回副本。

```
PURGE RECYCLEBIN;
```

版本号查询。

```
select VERSION_ID from information_schema.recyclebin;
```

16.5.3 限制说明

无

16.5.4 最佳实践

无

16.6 FLASHBACK

16.6.1 语法说明

16.6.1.1 FLASHBACK TABLE

```
FLASHBACK TABLE table_name [FOR RECYCLEBIN_VERSION version] TO BEFORE  
DROP;
```

16.6.2 用例

FLASHBACK TABLE 不带版本号，闪回到最近的一个闪回副本。

```
FLASHBACK TABLE db.tb1 TO BEFORE DROP;
```

FLASHBACK TABLE 带版本号，闪回到制定版本的闪回副本。

```
FLASHBACK TABLE db.tb1 FOR RECYCLEBIN_VERSION "version_info" TO BEFORE  
DROP;
```

16.6.3 限制说明

无

16.6.4 最佳实践

无

16.7 INITCAP

16.7.1 语法说明

INITCAP 函数只支持输入一个参数。当输入参数个数不为 1 时，就会报错。支持参数为常量、字段、函数等场景；支持增、删、改、查等场景，在 where 条件、字段值、order by、group by 等处使用 INITCAP。

当 sql_mode 设置为 EMPTY_STRING_IS_NULL 时，兼容 oracle 将空串转换为 NULL 值，其他类似的 oracle 兼容函数也类似。

16.7.2 用例

- 例子 1:

```
SELECT INITCAP (STRING) FROM TAB1 WHERE ID = 8;
```

- 例子 2:

```
SELECT INITCAP(A.STRING),INITCAP(B.STRING) FROM TAB1 A LEFT JOIN TAB2
```

B ON A.ID = B.ID;

- 例子 3:

```
SELECT STRING,INITCAP(STRING) AS RESULT FROM TAB1 GROUP BY STRING
HAVING COUNT(*) > 2;
```

16.7.3 限制说明

输入参数只能为 1 个

16.7.4 最佳实践

无

16.8 GDBLOCK

16.8.1 语法说明

GDBLOCK TABLE database_name table_name [all/read/write] ;

GDBUNLOCK TABLE database_name table_name ;

16.8.2 用例

GDBLOCK TABLE 加读锁。

```
GDBLOCK TABLE db.tb1 read;
```

GDBLOCK TABLE 加写锁。

```
GDBLOCK TABLE db.tb1 write;
```

GDBUNLOCK TABLE 解锁。

```
GDBUNLOCK TABLE db.tb1;
```

16.8.3 限制说明

无

16.8.4 最佳实践

无

17 USER-DEFINED FUNCTION

17.1 语法说明

17.1.1 CREATE FUNCTION

```
+-----+ | CREATE ||| [DEFINER
= { user | CURRENT_USER } ] ||| FUNCTION sp_name ([func_parameter[...]]) |||
  RETURNS type ||| [characteristic ...] routine_body ||| func_parameter: |||
param_name type ||| type: ||| Any valid MySQL data type ||| characteristic: |||
| COMMENT 'string' ||| | LANGUAGE SQL ||| | [NOT] DETERMINISTIC ||| |
```

```
GDBUNLOCK TABLE db.tb1;
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } |||| SQL
SECURITY { DEFINER | INVOKER } ||| routine_body: ||| Valid SQL routine
statement | +-----+
```

17.1.2 DROP FUNCTION

DROP { FUNCTION } [IF EXISTS] sp_name

17.2 用例

```
CREATE FUNCTION hello (s CHAR(20))
```

```
RETURNS CHAR(50) DETERMINISTIC
```

```
RETURN CONCAT('Hello, ',s,'!');
```

```
SELECT hello('world');
```

17.3 使用说明

- 自定义函数仅对入参进行处理，函数体内只能使用复制表的数据，数据库不做校验。
- 自定义函数出现在 FROM 子查询中时：

主查询和子查询中不能含有 SELECT *，不支持 CR 条件，也不支持子查询表是复制表。

如：

```
SELECT * FROM (SELECT func() FROM db1.t1) tt1;
```

- 自定义函数不支持在计算节点层做表达式计算和汇聚计算。
 - 不能合并下压时，不支持 WHERE 子查询结果集大于 20 条的场景。
 - 不支持多节点时含有汇聚函数。
 - 不支持含有 Oracle 函数。
 - 不支持不能合并的 JOIN。

17.4 最佳实践

无

18 AP 节点(MPP)的使用

MPP 是分布式 SQL 查询引擎，适用于交互式分析查询，允许你查询存储于 GoldenDB 中的 GB 到 PB 量级的数据，并可实现联邦查询。MPP 利用多个 Worker 线程可以轻松实现并行查询，并可以通过集群进行横向扩展，增加计算节点增强查询效率。

18.1 hint 注释控制是否走 MPP

`/*!parallel-processing=on*/` 或 `/*M!parallel-processing=on*/` (注意：M 一定要大写，否则不生效)

`/*!parallel-processing=off*/` 或 `/*M!parallel-processing=off*/`

`/*!parallel-processing=safe*/` 或 `/*M!parallel-processing=safe*/`

推荐使用 `/*M!` 这种注释，即使语句下 DB 也会当成注释处理，不会出现解析错误的问题

可添加在语句任意位置，例句：

```
explain /*!parallel-processing=on*/select * from t1 where b in (select b
from t2);
```

```
explain /*!parallel-processing=off*/select * from t1 where b in (select b
from t2);
```

开关：

#默认并行计算模式，0：关闭，1：打开，2：安全（函数判断）

#uint: NA, range: 0-OFF,1-ON,2-SAFE, default: 0, dynamic: yes;

default_parallel_processing_mode = 0

注：该配置项在 proxy.ini 配置文件[GENERAL]下，不是配置在[LINK]，否则会不生效。

18.2 代价估算自适应配置

CN 为了自适应 MPP, 部分流程会对 sql 语句进行代价估算，设定一个阈值，超过此阈值则由 MPP 处理，反之由 CN 层自行处理。

CN 层代价估算计算数阈值，超过则由 MPP 处理

unit64: B, range: 0-18446744073709551615, default: 1000010000100, dynamic: yes; sendtompp_cost_rows = 10000000000

18.3 MPP 的限制

1. 目前只支持 UR 级别（读未提交）的事务隔离级别。CR 隔离级别暂时不支持。

UR 由配置生效，不能以 hint 形式加在语句中。

2. MPP 只能查询静态的数据。当前事务中的 DML 操作的数据，对于 MPP 都是不可见的。
3. MPP 目前只支持 DQL。其中 INSERT、UPDATE 派生的 DQL 也支持。

建议：事务中暂不要使用 MPP。

19 存储过程

19.1 单机存储过程

19.1.1 语法说明

19.1.1.1 CREATE PROCEDURE

CREATE

[DEFINER = { user | CURRENT_USER }]

PROCEDURE sp_name ([proc_parameter[,...]])

[characteristic ...] routine_body

[distributed_option]

proc_parameter:

[IN | OUT] param_name type

type:

Any valid MySQL data type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

routine_body:

Valid SQL routine statement

distributed_option:

DISTRIBUTED BY

HASH(column_list) ({groupno},...)

| RANGE(expr) ({groupno VALUES LESS THAN { value | MAXVALUE}},...)

| LIST(expr) ({groupno VALUES IN (value_list)},...)

| DUPLICATE ({groupno},...)

19.1.1.2 DROP PROCEDURE

DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

19.1.1.3 CALL PROCEDURE

CALL sp_name([parameter[,...]])

CALL sp_name[]

19.1.2 用例

DELIMITER //

CREATE PROCEDURE simpleproc (IN a INT, OUT param1 INT)

BEGIN

INSERT INTO t VALUES(@a);

SELECT COUNT(*) INTO param1 FROM t;

END

DISTRIBUTED BY DUPLICATE (g1)

END//

DELIMITER ;

CALL simpleproc(10,@b);

19.1.3 使用说明

- 计算节点直接将存储过程定义下发至数据节点执行，因此存储过程定义内容（除分发信息）必须完全兼容数据节点语法。
- 不支持分布式事务。
- 不支持 INOUT 类型参数。

- 支持单点下压的存储过程。根据入参判断出存储过程可直接下压 DB。
- 支持下发全部数据节点的存储过程，根据分发策略规则，将存储过程下发至所有数据节点进行执行，并对执行结果汇总后返回给客户端。

此类存储过程只能存在 IN 类型的参数。

- 创建存储过程必须指定分发策略。
- 如果带有 OUT 参数，必须是单节点表。
- 涉及 PROCEDURE 的语句中，若包含 Oracle 函数，是否设置 parse_mode =2 模式。

19.1.4 最佳实践

无

19.2 分布式存储过程

19.2.1 语法说明

19.2.1.1 存储过程创建

CREATE

[DEFINER = { user | CURRENT_USER }]

PROCEDURE sp_name ([proc_parameter[...]])

[characteristic ...] routine_body

proc_parameter:

[IN | OUT | INOUT] param_name type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

routine_body:

Valid SQL routine statement

19.2.1.2 存储过程执行

CALL sp_name([parameter[,...]])

CALL sp_name[]

19.2.1.3 存储过程修改

只支持同名存储过程整体替换，不支持仅修改指定字段。

CREATE OR REPLACE

[DEFINER = { user | CURRENT_USER }]

PROCEDURE sp_name ([proc_parameter[,...]])

[characteristic ...] routine_body

proc_parameter:

[IN | OUT | INOUT] param_name type

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

routine_body:

Valid SQL routine statement

CREATE OR REPLACE PROCEDURE proc_name [characteristic ...]

characteristic:

COMMENT 'string'

| LANGUAGE SQL

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

19.2.1.4 存储过程删除

DROP PROCEDURE [IF EXISTS] sp_name

19.2.1.5 存储过程查看

查看单个存储过程：

SHOW CREATE PROCEDURE sp_name

查看所有存储过程状态信息：

SHOW PROCEDURE STATUS

19.2.1.6 存储过程中支持的 SQL 列表

START TRANSACTION

COMMIT

SAVEPOINT identifier

ROLLBACK

ROLLBACK TO identifier

RELEASE SAVEPOINT identifier

SELECT

INSERT

INSERT ... SELECT

REPLACE

REPLACE ... SELECT

DELETE

UPDATE

COPY TABLE

TRUNCATE TABLE

SET

CREATE DB

DROP DB

CREATE TABLE

需要按 GoldenDB 的语法指定分发选项

CREATE TABLE ... LIKE 语法在 GoldenDB 中不支持。

CREATE TABLE 时 FULLTEXT 不支持，因为在 GoldenDB 中不支持。

DROP TABLE

RENAME TABLE

ALTER TABLE

CREATE INDEX

DROP INDEX

CREATE SEQUENCE

DROP SEQUENCE

ALTER SEQUENCE

CREATE SYNONYM

DROP SYNONYM

CREATE VIEW

DROP VIEW

CREATE PROCEDURE

DROP PROCEDURE

CALL PROCEDURE

SHOW VARIABLES

SHOW COLLATION

SHOW CREATE TABLE

SHOW CREATE TABLE 后加 WHERE 不支持。

SHOW DATABASES

SHOW TABLES

SHOW CREATE DATABASE

SHOW SEQUENCES

SHOW CREATE SEQUENCE

SHOW TRIGGERS

SHOW CREATE TRIGGER

SHOW CREATE PROCEDURE

SHOW CREATE FUNCTION

SHOW FIELDS: 例如 SHOW FIELDS FROM tbl_1

SHOW CHARSETS

SHOW WARNS

SHOW STATUS

SHOW WARNINGS

SHOW ENGINES

SHOW TABLE STATUS

SHOW PROCEDURE STATUS

SHOW FUNCTION STATUS

SHOW PLUGINS

SHOW KEYS

SHOW EVENTS

SHOW PRIVILEGES

SHOW GRANTS

SIGNAL

RESIGNAL

GET DIAGNOSTICS

存储过程中的语句:

DECLARE 语句

CURSOR 相关语句:

OPEN CURSOR,

FETCH INTO

CLOSE CURSOR

19.2.1.7 存储过程中明确不支持的 SQL 列表

- 不支持 ALTER PROCEDURE, CREATE /DROP/ALTER TRIGGER。
- 不支持 USE/SHOW ERRORS。
- 不支持 PREPARE、DEALLOCATE PREPARE、EXECUTE。

19.2.2 使用说明

- 不支持自定义函数。
- 存储过程中不支持@变量, 和@@变量。
- 分布式存储过程的出入参支持基本的数据类型(各种 INT/INT > UNSIGNED/CHAR/VARCHAR/FLOAT/DOUBLE/DECIMAL/DATE/TIME/DATETIME 等)。
- 支持的时间类型: 支持 yyyy-mm-dd > hh:mm:ss 标准格式, 其它格式是未定义的。
- 建议尽可能地减少嵌套调用, 或者适当调大 SPENGINE 的配置项 max_sp_engine_stack_size, 并使其生效。
- 不支持隐式传参。
- 存储过程中, 不支持 IN/ALL/ANY/SOME subquery 定义的 SET 语句。

20 XA 事务语法

20.1 语法说明

20.1.1 XA START/BEGIN

XA {START|BEGIN} *xid*

20.1.2 XA END

XA END *xid*

20.1.3 XA PREPARE

XA PREPARE *xid*

20.1.4 XA COMMIT

XA COMMIT *xid* [ONE PHASE]

20.1.5 XA ROLLBACK

XA ROLLBACK *xid*

20.1.6 XA RECOVER

XA RECOVER [CONVERT XID]

20.2 用例

- 例子 1:

XA START "aa";

insert into testdb.t1 values (1);

XA END "aa";

XA PREPARE "aa";

XA COMMIT "aa";

- 例子 2

XA START "aa";

insert into testdb.t1 values (1);

XA END "aa";

XA COMMIT "aa" ONE PHASE;

- 例子 3:

XA RECOVER;

20.3 使用说明

- 不能在普通事务中开启 XA 事务
- 不能在 XA 事务中开启普通事务
- 部分语句类型不能在 XA 事务中执行，如 DDL 语句
- 不支持 xid 内容为空，如 `xa start ""`;
- GDB 语法中，data+formatID 相同认为是同一个 xid。例如 xid "aa","bb",7 与 "aab","b",7 认为是同一个 xid。

21 保留字

表 19-1 保留字

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CALL	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	CONDITION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GET	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOURL_MICROSECOND
HOURL_MINUTE	HOURL_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INOUT
INSENSITIVE	INSERT	INT
INT1	INT2	INT3
INT4	INT8	INTEGER
INTERVAL	INTO	IO_AFTER_GTIDS
IO_BEFORE_GTIDS	IS	ITERATE
JOIN	KEY	KEYS

ACCESSIBLE	ADD	ALL
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINEAR	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_BIND
MASTER_SSL_VERIFY_SERVER_CERT	MATCH	MAXVALUE
MEDIUMBLOB	MEDIUMINT	MEDIUMTEXT
MIDDLEINT	MINUTE_MICROSECOND	MINUTE_SECOND
MOD	MODIFIES	NATURAL
NOT	NO_WRITE_TO_BINLOG	NULL
NUMERIC	ON	OPTIMIZE
OPTION	OPTIONALLY	OR
ORDER	OUT	OUTER
OUTFILE	PARTITION	PRECISION
PRIMARY	PROCEDURE	PURGE
RANGE	READ	READS
READ_WRITE	REAL	REFERENCES
REGEXP	RELEASE	RENAME
REPEAT	REPLACE	REQUIRE
RESIGNAL	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SIGNAL
SMALLINT	SPATIAL	SPECIFIC
SQL	SQLEXCEPTION	SQLSTATE
SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL	STARTING
STRAIGHT_JOIN	TABLE	TERMINATED
THEN	TINYBLOB	TINYINT

ACCESSIBLE	ADD	ALL
TINYTEXT	TO	TRAILING
TRIGGER	TRUE	UNDO
UNION	UNIQUE	UNLOCK
UNSIGNED	UPDATE	USAGE
USE	USING	UTC_DATE
UTC_TIME	UTC_TIMESTAMP	VALUES
VARBINARY	VARCHAR	VARCHARACTER
VARYING	WHEN	WHERE
WHILE	WITH	WRITE
XOR	YEAR_MONTH	ZEROFILL
PARTITION	PROXY	DISTRIBUTED
MULTI_STEP_QUERY	UR	SEMI
CR	SW	CW
CHGDISKEY	STORAGEDB	SYSDATE
DECODE	ROWNUM	SEQUENCES
INCREMENT	NOMAXVALUE	MINVALUE
NOMINVALUE	CYCLE	NOCYCLE
NOCACHE	NOORDER	READMASTER
READSLAVE	READBALANCE	CURRVAL
NEXTVAL	GROUPS	NULLCOLS
SYNONYM	PUBLIC	SAMEDB
VARCHAR2	CLOB	PLS_INTEGER
SUBDISTRIBUTED	PARSE_VCOL_EXPR	PAGE_CHECKSUM
RETURNING	CURRENT_ROLE	DISTRIBUTION
OVER	KEEPALIVE	NOGTID
WRITE_SAMEDB	REDISTRIBUTING	CONNECT
RULE	SYSTEM	EXCEPT
EMPTY	OF	RANK
USER		
WINDOW		

22 函数限制

22.1 随机值函数和时间函数

22.1.1 复制表

对于复制表，`insert` 语句和 `update` 语句会下发到表的所有分片上执行，在各分片插入相同的数据或进行相同的更新。但某些场景下，存在实际插入或更新到各分片上的数据可能不一致的风险，主要是随机值函数和时间函数的使用所导致的。下面列出存在分片间数据不一致风险的场景，在实际使用时需评估风险、谨慎考虑。

- 复制表进行 `insert` 或者 `update` 操作，使用数据库的下述函数生成数据
- 生成随机值函数 `rand`
- 时间函数（获取当前时刻）
- `now`
- `sysdate`
- `curtime`
- `current_time`
- `current_timestamp`
- `localtime`
- `localtimestamp`
- `utc_time`
- `utc_timestamp`
- `systimestamp`
- 生成时间戳函数
- `unix_timestamp`
- 建表 DDL 中时间类型字段的自动更新特性
- `TIMESTAMP(n) DEFAULT CURRENT_TIMESTAMP(n) ON UPDATE CURRENT_TIMESTAMP(n)`（不带精度同样也存在风险）
- 字段类型
- `TIMESTAMP`
- `DATETIME`
- `DEFAULT` 值
- `CURRENT_TIMESTAMP`

- LOCALTIME
- LOCALTIMESTAMP
- UPDATE 值
- CURRENT_TIMESTAMP
- LOCALTIME
- LOCALTIMESTAMP
- 虚拟列表达式中包含自动更新特性的时间函数类型的列
- GENERATED ALWAYS AS 表达式中含上述自动更新特性的时间函数类型的列

22.1.2 分发表

对于分发表，`update` 语句使用数据库的下述时间函数生成 `set` 的数据时（即 `update` 语句的 `set` 表达式中带下述时间函数），存在同一批更新的数据最终实际值不一致的风险，尤其是在所要更新数据的数量较大的场景下。

- 时间函数（获取当前时刻）
- `now`
- `sysdate`
- `curtime`
- `current_time`
- `current_timestamp`
- `localtime`
- `localtimestamp`
- `utc_time`
- `utc_timestamp`
- `systimestamp`
- 生成时间戳函数
- `unix_timestamp`