LightDB 与 MySQL（5.7-8.0）主要特性比较

目录

简介

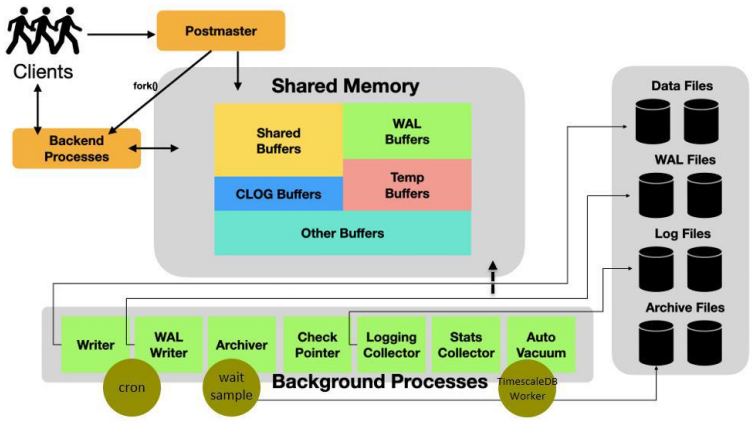
Light-DB 是恒生自主设计、研发的开源分布式关系型数据库，是一款同时支持在线事务处理与在线分析处理 (Hybrid Transactional and Analytical Processing, HTAP）的融合型分布式数据库产品，具备水平扩容或者缩容、金融级高可用、实时 HTAP、云原生的分布式数据库、兼容 MySQL 8.0 协议和 MySQL 生态等重要特性。Light-DB 致力于为用户提供一站式 OLTP (Online Transactional Processing)、OLAP (Online Analytical Processing)、HTAP 解决方案。适用于高可用、强一致要求较高、数据规模较大等应用场景。

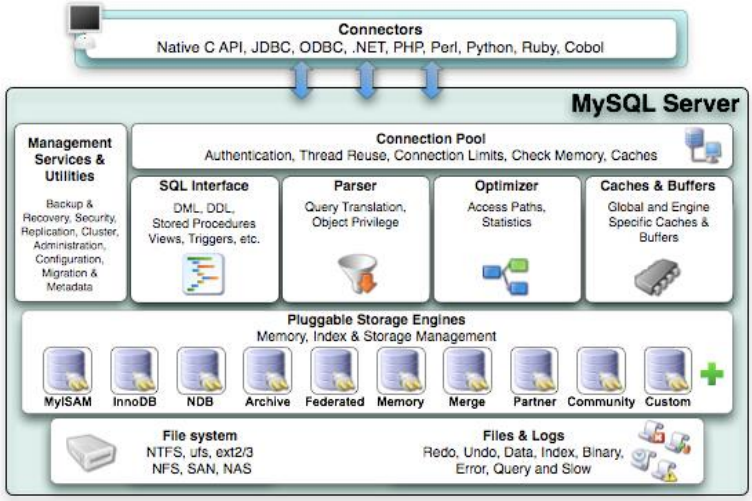
MySQL 目前的主要版本是 5.7 和 8.0，且 5.7 和 8.0 之间没有明显区别，故下面对 LightDB 与 MySQL5.7, MySQL8.0 进行比较。

一、LightDB 与 MySQL 关键特性兼容性与替代实现

MySQL5.7 与 MySQL8.0 间没有明显区别的，下面直接以 MySQL 替代。

二、架构差异指南





差异对比表

特性对比	LightDB	MySQL	Oracle
集群	cluster	instance	instance
日志	Wal/archive	redo/binlog	Redo/archive
清理日志	lightdb_archive_retention_size/lightdb_archive_dir 同时配置	expire_logs_days	rman delete archive
用户	user	user	user 就是 schema
schema	schema	schema 就是 database	schema 就是 user
database	database 隔离	database 就是 schema	database 就是 instance
支持事务的引擎	heap 表，同 Oracle	InnoDB	默认 heap
进程/线程	多进程	多线程	多进程
逻辑备份恢复	lt_dump/lt_restore	mysqldump	expdp/impdp
物理备份	lt_probackup	xtrabackup	rman
集群	LightDB 集群版	MGR、MHA 等	RAC
主从	基于 wal	基于 binlog	基于 redo
多表复杂 join 性能	较好	较差	较好
共享内存	shared_buffers	innodb_buffer_pool_size	sga

多进程与多线程

LightDB 为多进程架构，MySQL 为多线程架构。多线程架构优势： 线程的创建删除，切换成本相比进程低。多进程的优势：更健壮，稳定，一个客户端连接的崩溃不会导致整个

数据库的崩溃，而多线程极易导致。从开发代码角度，多进程易于开发，资源天然隔离。

存储引擎支持

都支持多存储引擎。MySQL 支持的存储引擎包括：InnoDB, MyISAM, Memory, CSV, Archive, Blackhole, Merge, Federated, Example。可通过在建表时指定 engine：Create table test(key1 int) engine=innodb。

数据库和 schema 概念的差别

LightDB 目前只支持一个存储引擎（heap）,不支持指定 engine。LightDB 有类似的 lightdb_syntax_compatible_type 参数可以切换到不同的数据库引擎，比如切换到 Oracle 和 MySQL。

MySQL 相同的用户可以访问不用的数据库，通过权限控制，比如 db1.table1，db2.table3

LightDB 相同的用户也可以访问不用的 schema 的表对象，比如 schema1.table1，schema2.table3

Oracle 相同的用户可以访问不用的用户的表对象，通过权限控制，比如 user1.table1，user2.table3

在 MySQL 中，create schema 和 create database 是等价的。

```
mysql> create database test_database;
Query OK, 1 row affected (0.00 sec)

mysql> create schema test_schema;
Query OK, 1 row affected (0.00 sec)

mysql> show database;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'database' at line 1
mysql> show schemas;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
| test_database      |
| test_schema        |
+--------------------+
7 rows in set (0.01 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
```

```
| mysql              |
| performance_schema |
| sys                |
| test               |
| test_database      |
| test_schema        |
+--------------------+
7 rows in set (0.00 sec)
```

而 LightDB 中，一个 database 可以包含多个 schema。

```
lightdb@postgres=# create database test_database;
NOTICE:  Canopy partially supports CREATE DATABASE for distributed databases
DETAIL:  Canopy does not propagate CREATE DATABASE command to workers
HINT:  You can manually create a database and its extensions on workers.
CREATE DATABASE
lightdb@postgres=# create schema test_schema;
CREATE SCHEMA
lightdb@postgres=# \l test_*
                                          List of databases
      Name       |  Owner  | Encoding |     Collate     |      Ctype      | Access privileges
-----------------+---------+----------+-----------------+-----------------+-------------------
 test_database | lightdb | UTF8     | zh_CN.UTF-8 | zh_CN.UTF-8 |
(1 row)

lightdb@postgres=# \dn test_*
     List of schemas
     Name      |  Owner
---------------+---------
 test_schema | lightdb
(1 row)
```

实际开发中，建议将 mysql 中的 create database 替换成 create schema，因为 LightDB 跨库关联查询很麻烦。

同时建议用户创建自己的数据库，不建议将用户 schema 创建到 postgres 库中。

闪回

物理回退：通过物理文件恢复，都支持。

在当前库回退：官方都不支持。

物理备份

MySQL 支持在线和离线完全备份以及崩溃和事务恢复，需要第三方软件才能支持热备份。LightDB 支持在线和离线完全备份以及崩溃、时间点和事务恢复，可以支持热备份。LightDB EM 提供了自动备份功能、集中备份管理功能以及远程备份。

慢日志

MySQL 通过如下参数控制，记录在单独文件。

- slow_query_log：慢查询开启状态；
- slow_query_log_file：慢查询日志存放的位置（一般设置为 MySQL 的数据存放目录）；
- long_query_time：查询超过多少秒才记录。

LightDB 通过 log_min_duration_statement 参数控制，如果指定值时没有单位，则以毫秒为单位。将这个参数设置为零将打印所有语句的执行时间。设置为 -1（默认值）将停止记录语句持续时间。只有超级用户可以改变这个设置。

事务隔离

MySQL 在 RR 下会有幻读，可以并发更新。LightDB 在 RR 下不会有幻读，并发更新会报错，报数据已经被更新。

数据同步

MySQL 数据同步为流式复制包括异步复制，半同步复制，GTID 复制，MGR 复制（paxos）。LightDB 数据同步支持基于文件的复制，流式复制包括 remote_apply，on，remote_write，local。

按同步的数据格式分，都支持逻辑复制和物理复制。LightDB 的逻辑复制只支持表的复制，可以实现表级别的订阅和发布，不支持 DDL 复制；而 MySQL 的逻辑复制只能基于实例进行复制，支持 DDL。

高可用

MySQL 的高可用性解决方案目前大致分为 5 种，按照高可用的级别（99.9999%为最高级）排序依次为，主从复制、具有自动故障转移功能的主从复制、利用共享存储、OS 或虚拟化软件实现主备架构、MySQL Group Replication 群组复制，以及 MySQL NDB Cluster。

- MySQL Replication：允许数据从一台实例上复制到一台或多台其它的实例上。
- MySQL Group Replication：群组复制提供更好的冗余性、自动恢复以及写入扩展。
- MySQL InnoDB Cluster：基于群组复制，提供了易于管理的 API、应用故障转移和路由、易于配置，提供比群组复制更高级别的可用性。
- MySQL NDB Cluster：容易与 MySQL InnoDB Cluster 混淆，是另外一款产品，提供更高级别的可用性和冗余性。适用于分布式计算环境，使用内存型的 NDB 存储引擎。

LightDB 原生集成了高可用模块，支持单中心和双中心部署，能够自动预防脑裂。支持一主多副，副本可用于读写分离。在高可用级别上支持最大保护模式、最大可用模式、最大性能模式以及最大一致性模式（LightDB 21.3 特性），比 Oracle 多一级。LightDB 高可用安装默认不提供负载均衡特性，如果用户希望保证强一致的读写分离，需要使用最大一致性模式。LightDB 22c 将提供原生负载均衡和多主。

三、与 MySQL 兼容性概述

本节主要介绍 LightDB 数据库的 MySQL 模式与原生 MySQL 数据库的兼容性对比信息。

LightDB 数据库的 MySQL 模式兼容 MySQL 5.7/8.0 的绝大部分功能和语法。由于产品架构不同，或者客户需求不大，有些功能并没有被支持。本节主要从以下几方面介绍 LightDB 数据库的 MySQL 模式与原生 MySQL 数据库的不同：

- 数据类型
- SQL 语法
- 系统视图
- 字符集和字符序
- 函数与表达式
- 分区支持
- 备份恢复
- 存储引擎
- 优化器
- 暂不支持的功能

特性	MySQL	LightDB
数据类型	23	25
内建函数	109	128
SQL 语法	10	10
系统视图	79	138
字符集和字符序 Collation	10	10
分区支持	4	3
备份恢复	4	4
存储引擎	8	8
优化器	5	5

批注 [张君华1]: 需要补充

数据类型

LightDB 数据库支持的数据类型有：

- 数值类型
    - 整数类型：BOOL/BOOLEAN、SMALLINT、INT/INTEGER 和 SIGNED/BIGINT。
    - 定点类型：DECIMAL 和 NUMERIC。

    - 浮点类型：FLOAT/DOUBLE PRECISION 和 REAL。
    - 序列类型：SMALLSERIAL、SERIAL 和 BIGSERIAL。
    - Bit-Value 类型：BIT。

- 日期时间类型：DATETIME/TIMESTAMP、DATE、TIME 和 INTERVAL。
- 字符类型：CHARACTER/CHAR 和 CHARACTER VARYING/VARCHAR、。
- 大对象类型：BLOB/BYTEA 和 CLOB。
- 文本类型：LONGTEXT/TEXT。
- 枚举类型：ENUM。
- 几何类型：POINT、LINE、LSEG、BOX、PATH、POLYGON 和 CIRCLE。
- JSON 数据类型。

批注 [张君华2]: 除了链接外的格式要处理下

LightDB 数据类型支持情况是等于或大于 MySQL 数据库。

SQL 语法

SELECT

- 支持大部分查询功能，包括支持单、多表查询；支持子查询；支持内联接、半联接以及外联接；支持分组、聚合；常见的概率、线性回归等数据挖掘函数等。
- 支持对多个 SELECT 查询的结果进行 UNION、UNION ALL、MINUS、EXCEPT 或 INTERSECT 等集合操作。
- 支持通过如下方式查看执行计划：

```
EXPLAIN [(option[, …])] statement
option:
    ANALYZE
  | VERBOSE
  | COSTS
  | SETTIINGS
  | BUFFERS
  | WAL
  | TIMING
  | SUMMARY
  | FORMAT {TEXT | XML | JSON | YAML}
```

- 支持 SELECT ... FOR SHARE ... 语法（OceanBase 不支持）。

INSERT

- 支持单行和多行插入，以及指定分区插入。

- 支持 `INSERT INTO ... SELECT ...` 语句。
- 支持 insert into ... on duplicate key update 主键存在则执行指定的更新操作，不存在则直接新增记录。

UPDATE

- 支持单列和多列更新。
- 支持使用子查询。
- 支持集合更新。

DELETE

- 支持单表删除。

TRUNCATE

- 支持完全清空指定表。

REPLACE INTO

- 支持 replace into，主键存在则先删除再更新，不存在则直接新增记录。

@变量

- 不支持用于分析函数的复杂@变量用法

系统视图

LightDB 数据库实现了 information_schema 内部数据库中的视图，由于架构不同，LightDB 数据库不保证所有视图中所有的列含义与 MySQL 相同。

更多系统视图的说明信息请参考《LightDB-X 参考手册》文档中 信息视图 章节。

字符集和字符序

LightDB 数据库兼容 MySQL 数据库的部分字符集和字符序，具体支持情况如下：

- 字符集：UF8、GBK、GB18030、SQL_ASCII、EUC、ISO。
- 字符序：LightDB 使用 LC_COLLATE 和 LC_CTYPE 两个参数来定义字符排序和字符分类的规则，具有更灵活的命名规则和更丰富的排序选项。相对于 MySQL 在字符排序和字符分类方面提供了更多的灵活性和可配置性，可以更精确地满足不同语言和地区的排序需求

函数

与 MySQL 数据库对比，LightDB 数据库的 MySQL 模式不支持如下函数：

- 字符串函数：LOAD_FILE()、MATCH() 和 SOUNDEX()。
- 时间和日期函数：LAST_DAY()、CURDATE()/CURRENT_DATE()、CURTIME()/CURRENT_TIME()、CURRENT_TIMESTAMP()、ADDTIME()、DAYNAME()、DAYOFMONTH()和 DAYOFYEAR()。
- XML 函数：ExtractValue() 和 UpdateXML()。
- 加密和压缩函数：RANDOM_BYTES()、SHA1(), SHA()、SHA2()、UNCOMPRESSED_LENGTH() 和 VALIDATE_PASSWORD_STRENGTH()。
- 锁定函数：GET_LOCK()、IS_FREE_LOCK()、IS_USED_LOCK()、RELEASE_ALL_LOCKS() 和 RELEASE_LOCK()。
- 其他函数：MASTER_POS_WAIT() 和 NAME_CONST()。

分区支持

LightDB 数据库与 MySQL 数据库对分区的支持差异如下：

- LightDB 数据库支持一级分区，模板化和非模板化二级分区；MySQL 数据库不支持非模板化二级分区。
- LightDB 数据库的二级分区支持 Hash、Range、Range Columns、List 和 List Columns 分区、组合分区（组合分区 RANGE|LIST|HASH）；MySQL 数据库的二级分区仅支持 Hash 分区和 Key 分区。

备份恢复

LightDB 数据库兼容了部分 MySQL 数据库的备份恢复特性，主要支持情况如下：

- 支持全量备份和增量备份。
- 支持集群级别的备份恢复（OceanBase 不支持）。
- 支持热备份和冷备份（OceanBase 不支持冷备份）。
- 支持租户内部部分数据库和表级的备份恢复（OceanBase 不支持）。
- 支持备份数据的有效性验证（OceanBase 不支持）。

存储引擎

MySQL 与 LightDB 均支持多存储引擎。支持的存储引擎包括：InnoDB, MyISAM, Memory, CSV, Archive, Blackhole, Merge, Federated, Example。可通过在建表时指定 engine：Create table test(key1 int) engine=innodb。

优化器

LightDB 数据库在优化器方面与 MySQL 数据库的区别，主要表现在以下几个方面：

- 查看执行计划的命令：
  - 执行计划输出包含更详细的列信息，如节点类型、操作符、谓词信息等。
  - 支持使用 EXPLAIN 命令来查看执行计划，并且可以使用 ANALYZE 选项来获取真实的查询性能统计信息。
  - 提供 EXPLAIN ANALYZE 命令，可以同时输出执行计划和实际执行时间。
- 查看统计信息：
  - 通过系统表 pg_statistic 存储并维护表的统计信息，包括列值的直方图、唯一值的数量等。
  - 提供 ANALYZE 语句用于更新和收集表的统计信息。
- 查询改写优化：
  - 支持查询改写优化策略，包括联接重排、子查询优化、谓词下压、常量折叠等。
  - 支持外联接优化、块嵌套循环和批量 Key 访问联接等优化技术。
- Optimizer Hint 机制：
  - 不支持。
- 并行执行能力：
  - 支持并行查询，可以通过设置参数来控制并行查询的并发度。
  - 支持并行复制和并行写入等功能，可以提高系统的处理能力。
- 计划缓存和预编译：
  - 使用计划缓存来存储已编译的查询计划，可以避免重复编译相同的查询语句。
  - 支持预编译语句，可以提高执行效率和重用性。

  MySQL 数据库不支持计划缓存和预编译。

暂不支持的功能

- 不支持存储过程、函数、触发器
- 不支持复杂的@变量
- 不支持 convert 函数进行类型转换。
- 不支持 PAD_CHAR_TO_FULL_LENGTH;
- 不支持关于日期为零值的处理；
- 不支持 MySQL 优化器提示和优化器开关；
- 不支持 sql_mode 选项；
- MySQL datetime 支持指定精度，如： key1 datetime(6) ; LightDB 的 datetime 不支持精度，可通过 timestamp(p) without time zone 替代。
- 对于优化器，查看执行计划的命令不支持使用 SHOW WARNINGS 显示额外的信息。

四、开发差异实例

创建用户赋权操作

LightDB

```
    -- Users creation
CREATE USER fund60trans1 WITH PASSWORD 'fund60trans1';

-- Database creation
CREATE DATABASE fund60;

-- Revoke Privileges from PUBLIC
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON DATABASE fund60 FROM PUBLIC;

-- login on as osg_dev
ltsql -d fund60
create schema fund60trans1;

-- Read/write role
ltsql -d fund60
CREATE ROLE readwrite;
GRANT CONNECT ON DATABASE fund60 TO readwrite;
GRANT USAGE, CREATE ON SCHEMA fund60trans1 TO readwrite;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA fund60trans1 TO readwrite;
ALTER DEFAULT PRIVILEGES IN SCHEMA fund60trans1 GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO readwrite;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA fund60trans1 TO readwrite;
```

```
ALTER DEFAULT PRIVILEGES IN SCHEMA fund60trans1 GRANT USAGE ON SEQUENCES TO
readwrite;

-- Alter schema Owner
ALTER SCHEMA fund60trans1 OWNER TO fund60trans1;

-- Grant privileges to users
GRANT readwrite TO fund60trans1;
```

MySQL:

```
create database fund60trans1;
create user 'fund60trans1'@'ip' identified by 'fund60trans1';
grant all on fund60trans1.* to 'fund60trans1'@'%' with grant option;
```

update 多表关联更新

MySQL 的 update 语句中 update 后填充 set 子句中引用的表和 where 子句引用的表，且一个 update 语句可以更新多张表。以下示例中，testa 和 testb 的列都会被更新。

```
mysql> select * from testa;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
+------+------+
2 rows in set (0.00 sec)

mysql> select * from testb;
+------+------+
| a    | b    |
+------+------+
|    2 |    2 |
|    3 |    3 |
+------+------+
2 rows in set (0.00 sec)

mysql> update testa,testb set testa.b =3 ,testb.b = 1 where testa.a=testb.b
    -> ;
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from testa;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    3 |
+------+------+
2 rows in set (0.00 sec)

mysql> select * from testb;
+------+------+
```

```
| a      | b      |
+------+------+
|      2 |      1 |
|      3 |      3 |
+------+------+
2 rows in set (0.00 sec)
```

LightDB 中，一个 update 语句只能更新一张表，update 后面填待更新的表名，from 后面填 where 条件中使用的表名。更新两张表需要两个 sql 完成。同样的功能 LightDB 中 sql 如下：

```
lightdb@postgres=# select * from testa;
 a | b
---+---
 1 | 1
 2 | 2
(2 rows)

lightdb@postgres=# select * from testb;
 a | b
---+---
 2 | 2
 3 | 3
(2 rows)

lightdb@postgres=# update testa set testa.b =3    from testb where testa.a=testb.b;
UPDATE 1
lightdb@postgres=# update testb set testb.b =1    from testa where testa.a=testb.b;
UPDATE 1
lightdb@postgres=# select * from testa;
 a | b
---+---
 1 | 1
 2 | 3
(2 rows)

lightdb@postgres=# select * from testb;
 a | b
---+---
 3 | 3
 2 | 1
(2 rows)
```

delete 多表关联删除

MySQL 支持关联查询，删除哪张表由 delete 关键字后面的表指定，且 delete 后面可以跟多张表。MySQL 可以一个 sql 删除多张表的记录。如下 a，b 表的记录都被删除。

```
mysql> select * from testa;
+------+------+
| a      | b      |
+------+------+
|      1 |      1 |
|      2 |      3 |
```

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

mysql> select * from testb;
+------+------+
| a    | b    |
+------+------+
|    2 |    1 |
|    3 |    3 |
+------+------+
2 rows in set (0.00 sec)

mysql> delete testa,testb from testa,testb where testa.a = testb.b;
Query OK, 2 rows affected (0.00 sec)

mysql> select * from testa;
+------+------+
| a    | b    |
+------+------+
|    2 |    3 |
+------+------+
1 row in set (0.00 sec)

mysql> select * from testb;
+------+------+
| a    | b    |
+------+------+
|    3 |    3 |
+------+------+
1 row in set (0.00 sec)
```

LightDB 中，删除的表名直接放在 delete 关键字之后，using 子句中指明 where 子句引用的表。同时 LightDB 不支持一个 delete 语法删除两张表的内容，需要替换成匿名块或存储过程。

```
lightdb@postgres=# select * from testa;
 a | b
---+---
 1 | 1
 2 | 3
(2 rows)

lightdb@postgres=# select * from testb;
 a | b
---+---
 3 | 3
 2 | 1
(2 rows)

lightdb@postgres=# delete   from testa   using testb where testa.a = testb.b;
DELETE 1
lightdb@postgres=# select * from testa;
 a | b
---+---
 2 | 3
```

```
(1 row)

lightdb@postgres=# delete testb where a = 3 ;
DELETE 1

lightdb@postgres=# select * from testb;
 a | b
---+---
 2 | 1
(1 rows)
```

类型转换

MySQL 支持 cast 标准类型转换方式，也支持 convert 函数进行转换。

```
mysql> select cast(1 as signed);
+------------------+
| cast(1 as signed) |
+------------------+
|                 1 |
+------------------+
1 row in set (0.00 sec)

mysql> select convert(1 , signed);
+---------------------+
| convert(1 , signed) |
+---------------------+
|                   1 |
+---------------------+
1 row in set (0.00 sec)
```

LightDB 只支持 cast 类型转换，为兼容 MySQL 的 cast 表达式，可以通过 create domain 来支持 signed。

```
lightdb@postgres=# create domain public.signed as bigint;
CREATE DOMAIN
lightdb@postgres=# select cast('1' as signed);
 signed
--------
      1
(1 row)
```

replace into

MySQL 中有 replace into 语法，判断主键或唯一约束如果存在就先删除再更新，不存在则直接新增记录。

```
mysql> create table test_replace (a int,b int,primary key(a));
Query OK, 0 rows affected (0.01 sec)

mysql> desc test_replace;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
```

```
+-------+---------+------+-----+---------+-------+
| a     | int(11) | NO   | PRI | NULL    |       |
| b     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> replace into test_replace values(3,2);
Query OK, 1 row affected (0.00 sec)

mysql> select * from test_replace;
+---+------+
| a | b    |
+---+------+
| 3 |    2 |
+---+------+
1 row in set (0.00 sec)

mysql> replace into test_replace values(3,2);
Query OK, 1 row affected (0.00 sec)

mysql> select * from test_replace;
+---+------+
| a | b    |
+---+------+
| 3 |    2 |
+---+------+
1 row in set (0.00 sec)
```

LightDB 支持 replace into 语句，判断主键如果存在就先删除再更新，不存在则直接新增记录；

LightDB 中唯一约束尚不支持 replace into 语句，可以通过 insert into 的 on conflict 子语实现。Insert into 语句中，通过 conflict 指定冲突的列为主键，do update 后接更新的列。

```
lightdb@postgres=# create table test_replace (a int,b int);
CREATE TABLE
lightdb@ postgres =# alter table test_replace add constraint uk_a unique(a);
ALTER TABLE
lightdb@postgres=# insert into test_replace values(3,2) on conflict (a) do update set b=2;
INSERT 0 1
lightdb@postgres=# select * from test_replace;
 a | b
---+---
 3 | 2
(1 row)

lightdb@postgres=# insert into test_replace values(3,1) on conflict (a) do update set b=1;
INSERT 0 1
lightdb@postgres=# select * from test_replace;
 a | b
---+---
 3 | 1
(1 row)
```

insert into … on duplicate key update

MySQL 中有 insert into … on duplicate key update 语法，判断主键或唯一约束如果存在就不插入而执行指定的更新操作，不存在则直接新增记录。

```
mysql> create table test_replace (a int,b int,primary key(a));
Query OK, 0 rows affected (0.01 sec)

mysql> desc test_replace;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | NO   | PRI | NULL    |       |
| b     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> insert into test_replace values(3,2);
Query OK, 1 row affected (0.00 sec)

mysql> select * from test_replace;
+---+------+
| a | b    |
+---+------+
| 3 |    2 |
+---+------+
1 row in set (0.00 sec)

mysql> insert into test_replace values(3,20) as new on duplicate key update b=new.b;
Query OK, 1 row affected (0.00 sec)

mysql> select * from test_replace;
+---+------+
| a | b    |
+---+------+
| 3 |   20 |
+---+------+
1 row in set (0.00 sec)
```

LightDB 支持 insert into … on duplicate key update 语句，判断主键如果存在就不插入而执行指定的更新操作，不存在则直接新增记录；

LightDB 中唯一约束尚不支持 insert into … on duplicate key update 语句，可以通过 insert into 的 on conflict 子语实现。Insert into 语句中，通过 conflict 指定冲突的列为主键，do update 后接更新的列。

```
lightdb@postgres=# create table test_replace (a int,b int);
CREATE TABLE
lightdb@ postgres =# alter table test_replace add constraint uk_a unique(a);
ALTER TABLE
lightdb@postgres=# insert into test_replace values(3,2) ;
INSERT 0 1
lightdb@postgres=# select * from test_replace;
 a | b
```

```
---+---
 3 | 2
(1 row)

lightdb@postgres=# insert into test_replace values(3,100) on conflict (a) do update set
b=excluded.b;
INSERT 0 1
lightdb@postgres=# select * from test_replace;
 a | b
---+-----
 3 | 100
(1 row)
```

表结构迁移

MySQL 的 DDL 和 LightDB 存在一定差异，不能直接在 LightDB 中执行。为此 LightDB 提供了数据迁移工具，可以使用 ltloader 将 Mysql 的表直接迁移到 LightDB，通过 ltdump 可以获取完整的建表语句。ltloader 用法示例可查阅 MySQL 迁移 LightDB 用例。lt_dump 用法可阅读官方手册。

日期格式

MySQL 通过 str_to_date 将字符串转换成时间，date_format 将日期格式化成字符串。

```
mysql> select str_to_date('2017-01-06 10:20:30','%Y-%m-%d %H:%i:%s') as result;
+---------------------+
| result              |
+---------------------+
| 2017-01-06 10:20:30 |
+---------------------+
1 row in set (0.01 sec)

mysql> select date_format( cast('2017-01-06 10:20:30' as datetime),'%Y-%m-%d %H:%i:%s') as
result;
+---------------------+
| result              |
+---------------------+
| 2017-01-06 10:20:30 |
+---------------------+
1 row in set (0.00 sec)
```

LgithDB 同样支持通过 str_to_date 将字符串转换成时间，date_format 将日期格式化成字符串，但不支持使用变量绑定的 Prepare Statements；

也可以通过 to_timestamp，to_date 将字符串转换成时间，to_char 转换成字符串。除函数不同外，日期格式也不同。示例如下：

```
lightdb@postgres=# select to_timestamp('2014-12-14 22:00:00','YYYY-MM-DD HH24:MI:SS');
        to_timestamp
------------------------
 2014-12-14 22:00:00+08
(1 row)
```

```
lightdb@postgres=# select to_date('2014-12-14 22:00:00','YYYY-MM-DD HH24:MI:SS');
   to_date
------------
 2014-12-14
(1 row)

lightdb@postgres=#       select       to_char(to_timestamp('2014-12-14       22:00:00','YYYY-MM-DD
HH24:MI:SS'),'YYYY-MM-DD HH24:MI:SS');
            to_char
---------------------
 2014-12-14 22:00:00
(1 row)
```

日期格式可阅读官方手册。

单引号(‘’)、双引号(“”)、反单引号（`）

MySQL 可以使用单引号 或双引号，表示值(如 where name=”jack”)。LightDB 只能使用单引号表示值(如 where name=’jack’)，LightDB 的双引号用来表示系统标识符，如表名或字段名(如 where “name”=’jack’)。

MySQL 可以使用反单引号表示系统标识符，比如表名，字段名，LightDB 也支持该种用法。

rowid 伪列

MySQL 支持_rowid: _rowid 并不是一个真实存在的列，其本质是一个非空唯一列的别名。当表中存在一个数字类型的单列主键时，_rowid 其实就是指的是这个主键列；当表中不存在主键但存在一个数字类型的非空唯一列时，_rowid 其实就是指的是对应非空唯一列。在其他情况下不能使用。

```
create table t_a(key1 int auto_increment primary key,key2 int);
MySQL> select _rowid from t_a;       ----_rowid  即是 key1
+--------+
| _rowid |
+--------+
|          1 |
|          2 |
|          5 |
|          6 |
+--------+
4 rows in set (0.01 sec)
```

LightDB 支持 ctid 来唯一标识一行，类似于 oracle 的 rowid，与 MySQL 的_rowid 不同。

```
postgres=# select ctid from t_a; --
 ctid
-------
 (0,1)
```

```
 (0,2)
 (0,3)
(3 rows)
```

ctid 表示的是行的位置，而不是字段值，格式：(Data block, Row)，具体如下: - Data block: 记录所在的数据块编号 - Row: 记录的行编号

null、空格、空字符串及尾部空格

MySQL	LightDB
x = null 永远不成立	LightDB x = null 成立 (transform_null_equals=true)
x is null	x is null
'' = null 永远不成立	LightDB '' = null 成立 (transform_null_equals=true)
' ' = ''不成立	' ' = ''不成立
' ' = ' '成立	' ' = ' '成立
'a ' = 'a'不成立（不剔除后缀空格）	'a ' = 'a'不成立（不剔除后缀空格）

Char 字段空格

MySQL char(n) 会去掉尾部空格，插入'a ',只能通过'a' 匹配；LightDB 会在尾部填充空格，插入'a '可以通过'a' 或'a '匹配。

字符集与排序规则

MySQL 支持表级别字符集 create table xxx () default charset=utf8。

LightDB 建议 utf8 字符集。只支持库级别字符集 createdb -E EUC_CN -T template0 --lc-collate=zh_CN --lc-ctype=zh_CN dbname

显示宽度

MySQL 支持对数值类型指定显示宽度如 int(4)表示显示宽度为 4，一般与 zerofill 结合使用。但 MySQL8.0 打算弃用此功能，目前 8.0.26 版本会提示 warning：

```
MySQL> create table t_d(key1 int(4) zerofill);
Query OK, 0 rows affected, 2 warnings (0.18 sec)

MySQL> show warnings;
+---------+------+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level   | Code | Message
```

```
                                                                                  |
+---------+------+-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
| Warning | 1681 | The ZEROFILL attribute is deprecated and will be removed in a future release.
Use the LPAD function to zero-pad numbers, or store the formatted numbers in a CHAR column. |
| Warning | 1681 | Integer display width is deprecated and will be removed in a future release.
                                                                                  |
+---------+------+-------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------+
2 rows in set (0.00 sec)

MySQL> insert into t_d values(1);
Query OK, 1 row affected (0.02 sec)

MySQL> insert into t_d values(123456);
Query OK, 1 row affected (0.03 sec)

MySQL> select * from t_d;
+--------+
| key1   |
+--------+
|   0001 |
| 123456 |
+--------+
2 rows in set (0.00 sec)

MySQL>
```

LightDB 不支持对字段指定此属性来实现 zerofill 功能，但可以通过使用 lpad 函数来实现此功能，这也是 MySQL8.0 推荐的实现方式。

join

在 MySQL 中，join 默认为 cross join，后面可以不接 on 条件；在 LightDB 中，join 默认为 inner join，后面需要接 on 条件，否则会报语法错误；建议 SQL 均按标准语法进行书写，join 均写全名称，而不是缩写（例如 MySQL 中 cross join 即写成 cross join，不要缩写为 join）。

datetime 数据类型

MySQL datetime 支持指定精度，如： key1 datetime(6) ; LightDB 的 datetime 不支持精度，但可以通过 timestamp(p) without time zone 来替代。

find_in_set、group_concat 等函数

MySQL 提供了 FIND_IN_SET(str,strlist)函数，用来查询 strlist 中包含 str 的结果，其中 str 是要查询的字符串，strlist 是一个字符串列表，列表中各字符串以”,”分隔，如’a,b,c,d’。该函数的返回值规则如下：

1. 如果字符串 str 存在于由 N 个字符串组成的字符串列表 strlist 中，则返回值的范围在 1 到 N 之间，具体取决于在列表中匹配到的位置；

2. 如果 str 不在 strlist 或 strlist 为空字符串，则返回值为 0；

3. 如果任意一个参数为 NULL，则返回值为 NULL；

4. 如果第一个参数含有逗号‘,’，该函数将无法正常运行，会始终返回 0。

LightDB 也支持 find_in_set，用法与 MySQL 相同，除了 LightDB 的 find_in_set 不能用在 where 后，其他都与 MySQL 相同，如下：

```
postgres=# SELECT * from t1 where FIND_IN_SET(1.2,'1,1.2');
ERROR:  argument of WHERE must be type boolean, not type integer
LINE 1: SELECT * from t1 where FIND_IN_SET(1.2,'1,1.2');
                                           ^
postgres=#

MySQL> SELECT * from t1 where FIND_IN_SET(1.2,'1,1.2');
+------+------+------+
| id   | t    | name |
+------+------+------+
|    1 |  200 | jack |
|    2 |  300 | tom  |
|    3 |  400 | john |
+------+------+------+
3 rows in set (0.00 sec)

MySQL>
```

MySQL 提供了 group_concat 行转列功能函数，可以将 select xxx group by 分组查询的各行非 NULL 值拼接成一个字符串返回，其中的各行 value 默认以逗号’,’隔开，如下例查询中，student_name 为’张三’的 test_score 共有 90、100、85 三行，则 group_concat 会将其拼接为一行字符串’90,100,85’返回。但如果要拼接的各行全都是 NULL 值，则该函数返回值也为 NULL。LightDB 同样支持 group_concat，语法语义相同。

```
SELECT student_name,
       GROUP_CONCAT(test_score)
      FROM student
  GROUP BY student_name;
```

```
GROUP_CONCAT([DISTINCT] col_name
          [ORDER BY { unsigned_integer | col_name}
              [ASC | DESC]]
          [SEPARATOR str_val])
```

max_connections

MySQL 支持的最大连接数为 max_connections+1, 多出来的 1 是保留给具有 CONNECTION_ADMIN 权限的用户使用（mysql5.7 为 supuser 权限）。

LightDB 支持的最大连接数为 max_connections，其中 superuser_reserved_connections 个连接是保留给超级用户的，即实际只能使用(max_connections - superuser_reserved_connections)个连接。

MySQL 支持在线修改 max_connections； LightDB 不支持在线修改 max_connections, 需要重启才能生效。

管理端口

数据库在超过设置最大连接数时，会报 too many connections，把新的连接拒之门外，mariadb 和 percona 支持 extra_port, 可以建立额外的连接连接数据库。 MySQL8.0 支持配置管理端口，不受 max_connections 参数限制。具体可参考 https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html。

LightDB 不支持此功能。

定时任务

定时任务，可以定时执行数据库的命令，MySQL 通过 event 实现，LightDB 通过 lt_cron 插件实现,使用方式如下：

```
CREATE EVENT IF NOT EXISTS e_test
ON SCHEDULE EVERY 1 SECOND
ON COMPLETION PRESERVE
DO CALL e_test();                    --------MySQL,隔一秒执行一次 e_test

SELECT cron.schedule('* * * * * *', 'call e_test()');     ------- LightDB,隔一秒执行一次 e_test
```

Mysql 的 event 当前一个任务没执行完，后一个任务到了时间也会开始执行（另一个线程），如果 event 有问题，如执行慢，可能导致创建大量线程。LightDB 的 cron 在当前任务没有执行完，下一个任务时间到了，不会执行，如果超时了还会被取消。

线程池

MySQL 社区版并不包含线程池特性，如果应用有大量的数据库短连接（例如成百上千连接不停的经常断开、重连），线程池对于保持 MySQL 数据库稳定是有价值的，这也是早期不少用户不选择 MySQL 社区版的主要原因。如果都是长连接或者连接并不是很多，则线程池的价值并没有其所述的那么大。

LightDB 是多进程架构，不需要线程池，对于大量短连接，可以通过使用进程池。目前只支持第三方的进程池如 PGPOOL-II 和 PgBouncer，这里不展开，具体参考相关链接。

列默认值支持表达式和函数

MySQL 8.0 支持将表达式和函数作为列的默认值，这个特性还是很有价值的，尤其是对于日期相关的类型来说，有时候因为某些原因需要使用 date 之外的类型，如果不支持表达式或函数，就比较麻烦了。如下所示：

```
CREATE TABLE tx (a varchar(32) DEFAULT (DATE_FORMAT(now(),"%Y-%m-%d %T")), b int
DEFAULT 1);
insert into tx (b) values (1);
select * from tx;
```

```
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2022-01-06 13:31:28 |    1 |
+---------------------+------+
1 row in set (0.00 sec)
```

MySQL8.0 在使用表达式和函数作为默认值是需要用括号括起来，如上，但对于 timestamp 和 datetime 指端，使用 current_timestamp 函数时可以不使用括号。支持在默认值中使用其他字段，其他字段不能为 AUTO_INCREMENT 字段，且其他字段如果为生成字段或使用了表达式作为默认值，其他字段需要定义在当前字段之前。具体可参考 https://dev.MySQL.com/doc/refman/8.0/en/data-type-defaults.html。

LightDB 在使用时可以使用也可以不使用括号括起来，LightDB 不支持在 DEFAULT 表达式中使用其他字段。

```
postgres=# CREATE TABLE tx (a varchar(32) DEFAULT to_char(current_timestamp,'YYYY-MM-DD'), b int DEFAULT 1);
postgres=# select * from tx;
        a          | b
-------------+---
 2022-01-06 | 1
(1 row)
postgres=# CREATE TABLE tx (a varchar(32) DEFAULT to_char(b,'YYYY-MM-DD'), b int DEFAULT 1);
ERROR:  cannot use column reference in DEFAULT expression
LINE 1: CREATE TABLE tx (a varchar(32) DEFAULT to_char(b,'YYYY-MM-DD...
                                                       ^
postgres=#
```

CHECK 约束

在 MySQL 8.0 之前，check 约束仅仅是语法上不报错，并未真正生效，从 8.0 开始，check 约束真正生效。语法如下：

[CONSTRAINT [*symbol*]] CHECK (*expr*) [[NOT] ENFORCED]

支持是否强制选项，指定非强制，不生效，当我们需要迁移一些低版本的历史数据时，它们可能会违反新的检查约束；此时可以先将该约束禁用，等数据迁移并处理完成之后，再次启用强制选项，LightDB 不支持强制选项，其他用法一致。

```
mysql> create table t_c (a int check(a>0) ,b int check (b> 0), constraint abc check (a>b));
into t_c VALUES(1,-1);
Query OK, 0 rows affected (0.12 sec)

mysql> insert into t_c VALUES(1,-1);
ERROR 3819 (HY000): Check constraint 't_c_chk_2' is violated.
mysql>
mysql> create table t_c (a int check(a>0) ,b int check (b> 0) not ENFORCED, constraint abc check (a>b));
Query OK, 0 rows affected (0.08 sec)

mysql> insert into t_c VALUES(1,-1);
Query OK, 1 row affected (0.02 sec)
```

```
mysql>



postgres=# create table t_c (a int check(a>0) ,b int check (b> 0), constraint abc check (a>b));
CREATE TABLE
postgres=# insert into t_c VALUES(1,-1);
ERROR:    new row for relation "t_c" violates check constraint "t_c_b_check"
DETAIL:   Failing row contains (1, -1).
postgres=# create table t_c (a int check(a>0) ,b int check (b> 0) not ENFORCED, constraint abc
check (a>b));
ERROR:    syntax error at or near "ENFORCED"
LINE 1: ...ble t_c (a int check(a>0) ,b int check (b> 0) not ENFORCED, ...
```

自增列

MySQL 与 LightDB 都支持自增列，但有区别，MySQL 通过 auto_increment 关键字指定列为自增列。插入的时候可以通过 NULL，来表示使用自动生成的值 ，如下在插入用户自定义的值 5 后,下一个自动生成的值为 6（5+1）。

```
create table t_a(key1 int auto_increment primary key, key2 int);
MySQL> insert into t_a values(NULL,1);
Query OK, 1 row affected (0.01 sec)

MySQL> insert into t_a values(NULL,1);
Query OK, 1 row affected (0.02 sec)

MySQL> insert into t_a values(5,1);
Query OK, 1 row affected (0.01 sec)

MySQL> insert into t_a values(NULL,1);
Query OK, 1 row affected (0.02 sec)

MySQL> select * from t_a;
+------+------+
| key1 | key2 |
+------+------+
|    1 |    1 |
|    2 |    1 |
|    5 |    1 |
|    6 |    1 |
+------+------+
4 rows in set (0.00 sec)
```

LightDB 可以通过 auto_increment 和 serial，identity。Auto_increment 即是 GENERATED ALWAYS AS IDENTITY，不支持通过 NULL 表示自动生成 id， 使用自定义的自增 id 时需要使用 OVERRIDING SYSTEM VALUE，且插入后，下次自增还是从上次自动生成的值开始自增：

```
postgres=# create table t_a(key1 int auto_increment primary key,key2 int);
```

```
CREATE TABLE
postgres=# insert into t_a(key2) values(1);
INSERT 0 1
postgres=# insert into t_a(key2) values(1);
INSERT 0 1
postgres=# insert into t_a     values(5,1);
ERROR:    cannot insert into column "key1"
DETAIL:    Column "key1" is an identity column defined as GENERATED ALWAYS.
HINT:    Use OVERRIDING SYSTEM VALUE to override.
postgres=# insert into t_a OVERRIDING SYSTEM VALUE    values(5,1);
INSERT 0 1
postgres=# insert into t_a(key2) values(1);
INSERT 0 1
postgres=# select * from t_a;
 key1 | key2
------+------
      1 |      1
      2 |      1
      5 |      1
      3 |      1
(4 rows)

postgres=# insert into t_a(key2) values(1);
ERROR:    duplicate key value violates unique constraint "t_a_pkey"
DETAIL:    Key (key1)=(5) already exists.
postgres=# insert into t_a(key2) values(1);
INSERT 0 1
```

Generated by default as identity 使用自定义的自增id时不需使用OVERRIDING SYSTEM VALUE，可以使用 OVERRIDING USER VALUE 来强制使用系统生成的 id:

```
postgres=# create table t_a(key1 int GENERATED by default AS IDENTITY primary key,key2 int);
CREATE TABLE
postgres=# insert into t_a values(5,1);
INSERT 0 1
postgres=# insert into t_a OVERRIDING USER VALUE values(6,1);
INSERT 0 1
postgres=# select * from t_a;
 key1 | key2
------+------
      5 |      1
1 |      1
```

使用 serail 插入自定义值不需使用 OVERRIDING SYSTEM VALUE。可以使用 OVERRIDING USER VALUE 来强制使用系统生成的 id:

```
create table t_a(key1 serial primary key,key2 int);
postgres=# insert into t_a values(10,1);
INSERT 0 1
postgres=# select * from t_a;
 key1 | key2
------+------
    10 |      1
(1 row)
```

```
postgres=# insert into t_a(key2) values(1);
INSERT 0 1
postgres=# select * from t_a;
 key1 | key2
------+------
    10 |    1
     1 |    1
(2 rows)

postgres=# insert into t_a(key2) values(1);
INSERT 0 1
postgres=# select * from t_a;
 key1 | key2
------+------
    10 |    1
     1 |    1
     2 |    1
(3 rows)

postgres=#
```

修改参数

MySQL 支持在线修改会话级参数，全局参数（重启后失效），支持持久化更改参数。LightDB 支持在线修改会话级，事务级参数，数据库级（新连接起效，重启后不失效，记录在 pg_db_role_setting 中），用户级参数（新连接起效，重启后不失效，记录在 pg_db_role_setting 中），支持持久化更改参数，相比 MySQL 不支持全局参数（重启后失效）这种方式，只能间接实现。

持久化更改参数

MySQL8.0 ，LightDB 都支持持久化更改参数，MySQL8.0 通过修改 MySQLd-auto.cnf 文件，需要 SYSTEM_VARIABLES_ADMIN or SUPER 权限.命令如下：

1. 同时修改 global 运行时参数，和配置文件：

```
SET PERSIST var_name=value|default
SET @@PERSIST.var_name=value|default
```

2. 不修改 global 运行时参数，只修改配置文件，在下次重启时生效：

```
SET PERSIST_ONLY var_name=value|default
SET @@PERSIST_ONLY.var_name =value|default
```

3. 移除持久化修改：

```
Reset PERSIST; 移除全部
Reset PERSIST var_name;
```

具体可参考 https://dev.MySQL.com/doc/refman/8.0/en/persisted-system-variables.html#persisted-system-variables-overview。

LightDB 通修改 postgresql.auto.conf 文件，只有 superusers 才能修改，执行修改命令后要对当前实例生效需要执行 pg_reload_conf 函数，或者执行 lt_ctl reload 命令。修改命令如下：

```
ALTER SYSTEM SET configuration_parameter { TO | = } { value | 'value' | DEFAULT }
ALTER SYSTEM RESET configuration_parameter
ALTER SYSTEM RESET ALL
```

DDL

LightDB ddl 支持事务的概念,可以放在事务中使用，MySQL 不支持，在执行 ddl 时会提交之前的事务。

函数&类型

LightDB 支持兼容的 mysql 函数和类型详细清单可查阅详 LightDB 官网：http://www.light-pg.com/docs/lightdb/current/myfce.html

全文检索

MySQL 的全文检索是通过全文索引来实现，通过对字段建立全文索引，可以对字段进行全文检索，只作用于 CHAR, VARCHAR, and TEXT columns。只能作用于 innodb 和 myisam 表。通过 MATCH (col1,col2,...) AGAINST (expr [search_modifier]) 检索

```
search_modifier:
  {
        IN NATURAL LANGUAGE MODE
      | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
      | IN BOOLEAN MODE
      | WITH QUERY EXPANSION
  }
Natural language:普通匹配
Boolean mode: 通过在单词前使用+ 或- 来表示存在或不存在
WITH QUERY EXPANSION：智能匹配，比如'database'  会匹配 MySQL，oracle 等。
```

```
MySQL> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' WITH QUERY EXPANSION);
+----+----------------------+-----------------------------------------+
| id | title                | body                                    |
+----+----------------------+-----------------------------------------+
|  5 | MySQL vs. YourSQL    | In the following database comparison ... |
|  1 | MySQL Tutorial       | DBMS stands for DataBase ...            |
|  3 | Optimizing MySQL     | In this tutorial we show ...            |
|  6 | MySQL Security       | When configured properly, MySQL ...     |
|  2 | How To Use MySQL Well | After you went through a ...           |
|  4 | 1001 MySQL Tricks    | 1. Never run MySQLd as root. 2. ...     |
+----+----------------------+-----------------------------------------+
6 rows in set (0.00 sec)
```

LightDB 可以在没有索引的情况下进行全文搜索，基于匹配操作符@@，它在一个 tsvector（文档）匹配一个 tsquery（查询）时返回 true，通过使用 gin 索引来加速全文检索：

```
SELECT title FROM pgweb WHERE to_tsvector(body) @@ to_tsquery('friend');
CREATE INDEX pgweb_idx ON pgweb USING GIN(to_tsvector('english', body));
```

具体可参考：

https://www.postgresql.org/docs/13/textsearch.html；

https://www.cnblogs.com/zhjh256/p/15357837.html。

JSON 类型

LightDB 支持 JSON 和 JSONB 两种类型，其中 JSON 按照文本存储， JSONB 按照二进制存储（和 mongdb 的 BSON 存储格式类似），两者均支持 JSON 对象和 JSON 对象（如果会修改，不建议使用） 数组。JSONB 对象支持全文检索，支持根据属性和属性值进行检索， 特别适合于站内结构化搜索。 属性支持 B 树索引和哈希索引。

```
SELECT artist_data->>'artist_id' AS artist_id ,
artist_data->>'artist' AS artist ,
jsonb_array_elements(artist_data#>'{albums}')->>'album_title' AS album_title ,
jsonb_array_elements(jsonb_array_elements(artist_data#>'{albums}')#>'{album_tracks}')->>'track_
name'
AS song_titles ,
jsonb_array_elements(jsonb_array_elements(artist_data#>'{albums}')#>'{album_tracks}')->>'track_
id' AS
song_id
FROM v_json_artist_data
WHERE artist_data->>'artist' = 'Metallica' ORDER BY album_title , song_id ;
```

	artist_id	artist	album_title	song_titles	song_id
1	50	Metallica	...And Justice For All	Sad But True	1802
2	50	Metallica	...And Justice For All	The Unforgiven	1804
3	50	Metallica	...And Justice For All	Don't Tread On Me	1806
4	50	Metallica	...And Justice For All	Nothing Else Matters	1808
5	50	Metallica	...And Justice For All	The God That Failed	1810
6	50	Metallica	...And Justice For All	The Struggle Within	1812
7	50	Metallica	...And Justice For All	Helpless	1813
8	50	Metallica	...And Justice For All	The Wait	1815
9	50	Metallica	...And Justice For All	Last Caress/Green Hell	1817
10	50	Metallica	...And Justice For All	Blitzkrieg	1819
11	50	Metallica	...And Justice For All	The Prince	1821
12	50	Metallica	...And Justice For All	So What	1823
13	50	Metallica	...And Justice For All	Overkill	1825
14	50	Metallica	...And Justice For All	Stone Dead Forever	1827
15	50	Metallica	...And Justice For All	Hit The Lights	1829
16	50	Metallica	...And Justice For All	Motorbreath	1831
17	50	Metallica	...And Justice For All	(Anesthesia) Pulling Teeth	1833

200 row(s) fetched - 47ms

```
CREATE INDEX idx_1 ON jsonb.actor USING GIN (jsondata);
```

通过使用 GIN 索引，通常性能可以提升数十倍。

从 5.7 版本开始，MySQL 支持由 RFC 7159 定义的原生 JSON 数据类型，可以高效地访问 JSON 文档中的数据，内部以二进制格式保存。MySQL 8.0 相比 MySQL5.7 支持更多的 json 函数，LightDB 相比 MySQL 支持更多函数，可参考：https://dev.MySQL.com/doc/refman/8.0/en/json-function-reference.html。

MySQL 不支持在 json 字段上直接建索引，只能通过生成列来建索引，如下为示例：

```
MySQL> CREATE TABLE `players` ( `id` INT UNSIGNED NOT NULL, `player_and_games` JSON NOT NULL, `names_virtual` VARCHAR(20) GENERATED ALWAYS AS (`player_and_games` ->> '$.name') NOT NULL, PRIMARY KEY (`id`),index(`player_and_games`) );
ERROR 3152 (42000): JSON column 'player_and_games' supports indexing only via generated columns on a specified JSON path.
MySQL> CREATE TABLE `players` ( `id` INT UNSIGNED NOT NULL, `player_and_games` JSON NOT NULL, `names_virtual` VARCHAR(20) GENERATED ALWAYS AS (`player_and_games` ->> '$.name') NOT NULL, PRIMARY KEY (`id`) );
Query OK, 0 rows affected (0.09 sec)

MySQL> INSERT INTO `players` (`id`, `player_and_games`) VALUES (1, '{
    "id": 1,
    "n      '>       "id": 1,
    '>       "name": "Sally",
    '>       "games_played":{
 "Batt      '>            "Battlefield": {
    '>                "weapon": "sniper rifle",
    '>                "rank": "Sergeant V",
    '>                "level": 20
    '>            },

    '>            "Crazy Tennis": {
        "      '>                "won": 4,
    '>                "lost": 1
    '>            },
    '>            "Puzzler": {
    '>                "time": 7
    '>            }
  }
        '>          }
    '>     }'
    -> );
Query OK, 1 row affected (0.03 sec)

MySQL> EXPLAIN SELECT * FROM `players` WHERE `names_virtual` = "Sally"\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: players
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
```

```
              rows: 1
          filtered: 100.00
             Extra: Using where
1 row in set, 1 warning (0.00 sec)

MySQL> CREATE INDEX `names_idx` ON `players`(`names_virtual`);
Query OK, 0 rows affected (0.14 sec)
Records: 0   Duplicates: 0   Warnings: 0

MySQL> EXPLAIN SELECT * FROM `players` WHERE `names_virtual` = "Sally"\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: players
   partitions: NULL
         type: ref
possible_keys: names_idx
          key: names_idx
      key_len: 82
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

MySQL>
```

JSON 函数

LightDB 目前实现了部分 MySQL JSON 函数（由于 LightDB 中的 JSONB 类型效率更高，故如下函数均是以 JSONB 为基础）：

名称	功能
json_pretty()	格式化 JSONB 文档
json_array()	创建 JSONB 数组
json_object()	创建 JSONB 对象
json_extract()	依据 JSONPATH 查询 JSONB 对应路径的值
json_contains()	判断 JSONB 某个路径的值是否和指定的 JSONB 值相同
json_contains_path()	判断 JSONB 是否包含一个或多个指定的路径

和 MySQL 中 JSON 函数的差异：

- ✓ LightDB 的上述函数中，JSONPATH 均不支持解析 ** 通配符；
- ✓ json_extract 只支持单个 JSONPATH 参数；

不可见列

MySQL 从 8.0.23 开始支持不可见列，其行为是在默认的 select *中不会包含不可见列，只在显示引用时可见，在 insert table values()中插入时，也不需要为其赋值。该特性使得在编写了通用的查询后，升级能够无缝进行，也可以用不可见列解决没有主键的表。

```
create table table1 (
id int auto_increment primary key,
name varchar(20),
age int invisible);   通过 invisible  关键字
```

LightDB 不支持直接使用不可见列，但其的自动更新时间戳、内置主键列是采用不可见列实现的。

生成列

生成的列是一个特殊的列，它总是从其他列计算而来。因此说，它对于列就像视图对于表一样。生成列有两种:存储列和虚拟列。 存储生成列在写入(插入或更新)时计算，并且像普通列一样占用存储空间。虚拟生成列不占用存储空间并且在读取时进行计算。如此看来，虚拟生成列类似于视图，存储生成列类似于物化视图(除了它总是自动更新之外)。LightDB 目前只实现了存储生成列。必须指定关键字 STORED。

```
CREATE TABLE people (
    ...,
    height_cm numeric,
    height_in numeric GENERATED ALWAYS AS (height_cm / 2.54) STORED
);
```

MySQL 支持存储和虚拟生成列。使用 VIRTUAL | STORED 关键字，不指定默认 VIRTUAL ，虚拟列不占有空间，查询时生成。

不可见索引

MySQL 8.0 支持不可见索引，不可见索引对优化器不可见，不能作用于主键，便于测试索引对性能的影响，毕竟对于大表，删除和重建索引是非常昂贵的操作。LightDB 不支持此关键字。MySQL 8.0 通过如下方式使用

index i_idx (i) invisible 同过对索引指定 invisible 关键字。

分析函数

使用分析函数（窗口函数），开发人员可以通过更清晰、简洁的 SQL 代码执行复杂分析。原来需要几十行甚至上百行代码完成的逻辑现在可以使用一条 SQL 语句表示复杂任务，编写和维护速度更快、效率更高。数据库中分析支持的处理优化可大幅提高查询性能。以前需要自联接或复杂过程处理的操作现在可以用原生 SQL 执行。以分组排序为例，如果有如下列表：

```
MySQL> select * from rank_over;
+------+-------+---------------------+
| id   | subid | curd                |
+------+-------+---------------------+
|    1 |     1 | 2018-09-24 00:47:12 |
|    2 |     1 | 2018-09-24 00:47:38 |
|    3 |     1 | 2018-09-24 00:47:42 |
|    4 |     2 | 2018-09-24 00:47:50 |
|    5 |     2 | 2018-09-24 00:47:54 |
```

```
|      6 |      3 | 2018-09-24 00:48:00 |
|      7 |      4 | 2018-09-24 00:48:06 |
|      8 |      3 | 2018-09-24 01:12:10 |
|      9 |      2 | 2018-09-24 01:12:11 |
+--------+--------+----------------------+
```

现在要取出每个 subid 下 curd 最大的 1 条。使用分析函数只需要很简单的 SQL：

```
select t.id,t.subid,t.curd
from(SELECT id,subid,curd,RANK() OVER(PARTITION BY subid ORDER BY curd DESC) RK
    FROM rank_over) t
where t.RK<2
```

如果没有分析函数，则要复杂得多，如下：

```
select t1.*  from
(select (@rowNum1:=@rowNum1+1) as rowNo,id,subid,curd from rank_over a,(Select
(@rowNum1 :=0)) b order by a.subid,a.curd desc) t1 left join
(select (@rowNum2:=@rowNum2+1) as rowNo,id,subid,curd from rank_over c,(Select
(@rowNum2 :=1)) d order by c.subid,c.curd desc) t2 on t1.rowNo=t2.rowNO
where t1.subid<>t2.subid or t2.subid is null
```

它们的结果都是：

```
+--------+--------+--------+----------------------+
| rowNo | id   | subid | curd                 |
+--------+--------+--------+----------------------+
|      1 |      3 |      1 | 2018-09-24 00:47:42 |
|      4 |      9 |      2 | 2018-09-24 01:12:11 |
|      7 |      8 |      3 | 2018-09-24 01:12:10 |
|      9 |      7 |      4 | 2018-09-24 00:48:06 |
+--------+--------+--------+----------------------+
4 rows in set (0.00 sec)
```

分析函数不仅用于提高开发效率，而且数据库优化器通常会对分析函数的执行进行优化，典型的即是避免了对基表的二次扫描。

如上分析函数需要使用如上的 OVER 子句来使用，当聚集函数使用 OVER 子句就可以作为分析函数使用，不然只是普通的聚集，为整个集合返回一行（窗口函数可以为每组返回多个值）。

LightDB 和 MySQL 支持的分析函数都相同，可参考链接，MySQL 窗口函数有下述限制：

1. 窗口函数不能出现在 UPDATE 和 DELETE 中；
2. 窗口函数不支持 DISTINCT；
3. 窗口函数不支持 NESTED。

MySQL 与 LightDB 聚集函数对比如下：

Mysql	LightDB
AVG()	AVG()
BIT_AND()	BIT_AND()
BIT_OR()	BIT_OR()
BIT_XOR()	不支持
COUNT()	COUNT()
COUNT(DISTINCT)	COUNT(DISTINCT)
GROUP_CONCAT()	GROUP_CONCAT()
JSON_ARRAYAGG()	json_agg
JSON_OBJECTAGG()	json_object_agg

MAX()	MAX()
MIN()	MIN()
STD()	stddev_pop
STDDEV()	stddev_pop
STDDEV_POP()	stddev_pop
STDDEV_SAMP()	STDDEV_SAMP() ； stddev
SUM()	SUM()
VAR_POP()	VAR_POP()
VAR_SAMP()	VAR_SAMP()；variance（）
VARIANCE()	VAR_POP()
STDDEV_SAMP()	stddev
VAR_SAMP()	variance
	array_agg
	bool_and
	bool_or
	every
	string_agg
	xmlagg
	corr
	covar_pop
	covar_samp
	regr_avgx
	regr_avgy
	regr_count
	regr_intercept
	regr_r2
	regr_slope
	regr_sxx
	regr_sxy
	regr_syy
	mode () WITHIN GROUP (ORDER BY anyelement)
	percentile_cont (fraction double precision) WITHIN GROUP (ORDER BY)
	percentile_cont (fractions double precision[]) WITHIN GROUP (ORDER BY)
	percentile_disc (fraction double precision) WITHIN GROUP (ORDER BY anyelement)

	percentile_disc (fractions double precision[]) WITHIN GROUP (ORDER BY anyelement)
	rank (args) WITHIN GROUP (ORDER BY sorted_args)
	dense_rank (args) WITHIN GROUP (ORDER BY sorted_args)
	percent_rank (args) WITHIN GROUP (ORDER BY sorted_args)
	cume_dist (args) WITHIN GROUP (ORDER BY sorted_args)
	GROUPING (group_by_expression(s))

在 MySQL 中，分析函数的执行计划并不体现在常规的 explain 输出中，要查看关于分析函数执行计划相关的信息，需要使用 EXPLAIN FORMAT=JSON 模式，然后查看其中 windowing 的部分。LightDB 不需要，直接使用 EXPLAIN 即可。

CTE 与递归 CTE

CTE 主要有两个用处：一、复用子查询；二、性能优化 。LightDB , MySQL8.0 都支持 CTE，语法相同，语义相同。MySQL5.7 不支持，。

集合操作符

LightDB 支持 UNION， EXCEPT 和 INTERSECT；MySQL 只支持 UNION

临时表

都支持临时表。LightDB 临时表分两种，一种是会话级临时表，一种是事务级临时表。会话级生命周期为整个会话，事务级在事务结束时操作临时表。MySQL 只有会话级临时表。

LightDB 临时表在事务块结束时的行为可以用 ON COMMIT 控制，三个选项是：

1. PRESERVE ROWS 在事务结束时不采取特殊的动作。这是默认行为。
2. DELETE ROWS 在每一个事务块结束时临时表中的所有行都将被删除。本质上，在每次提交时会完成一次自动的 TRUNCATE。
3. DROP 在当前事务块结束时将删掉临时表。

分区

都知道，在单表数据量巨大时有效采用分区能够极大的提高 SQL 语句的性能（但是也需要注意的是，因为 MySQL 本质上实现了 oracle 分区本地索引的概念，所以对非唯一索引搜索性能相比非分区而言会降低，https://zhuanlan.zhihu.com/p/28703566），同时降低维护复杂性。MySQL 从 5.1 开始就引入分区功能，在随后版本中功能增加并不多，但是在 5.7.17 开始采用了存储引擎自带的分区处理而不是作为插件来支持。

LightDB 内置提供 Range，List，Hash 分区及其组合，其他方式的分区可以通过继承或 union all 来实现。

下表总结了 MySQL 和 LightDB 分别支持的分区类型：

MySQL	Lightdb	描述
HASH	支持	
KEY（hash 的衍生版本，表达式由 MySQL 服务器自己决定）	不支持	
组合分区 RANGE|LIST/HASH|KEY	支持	
LIST	LIST	支持表达式作为分区键（相当于虚拟列）
RANGE	RANGE	支持表达式作为分区键（相当于虚拟列）
COLUMNS（MySQL 中合并称为 COLUMNS）	RANGE COLUMNS	多列范围分区，相当于 RANGE/RANGE 组合分区
	LIST COLUMNS	相当于 LIST/LIST 组合分区

正则表达式

下面列出了 MySQL 和 LightDB 分别支持的正则函数和操作符：

MySQL	LightDB
NOT REGEXP	!~，!~*
REGEXP	~,~*
REGEXP_INSTR()（8.0）	
REGEXP_LIKE()（8.0）	~,~*
REGEXP_REPLACE()（8.0）	regexp_replace()

REGEXP_SUBSTR()（8.0）	Substring()
RLIKE 即 REGEXP	~，~*
	regexp_match
	regexp_matches
	regexp_split_to_table
	regexp_split_to_array

LightDB 还支持 similar to 方式：

```
'abc' SIMILAR TO 'abc' true
'abc' SIMILAR TO 'a' false
'abc' SIMILAR TO '%(b|d)%' true
'abc' SIMILAR TO '(b|c)%' false
'-abc-' SIMILAR TO '%\mabc\M%' true
'xabcy' SIMILAR TO '%\mabc\M%' false
```

注：虽然我们在应用中广泛的使用正则表达式，但是一般不推荐在数据库中使用正则表达式匹配和其它相关操作。

TEXT 类型

MySQL 的各种 text 字段有不同的限制，要手动区分 tinytext, text, mediumtext 和 longtext,在严格模式下不能有默认值。LightDB text 能支持各种大小，可以有默认值，21.3 支持 longtext 类型，便于迁移。

DML returning

支持 dml 返回 Resultset，通常 INSERT/UPDATE/DELETE 这样的 DML 语句后，都会跟随 SELECT 查询当前记录内容，以进行接下来的业务处理，支持 dml retuuring, 可以减少一次交互。MySQL 不支持此用法。LightDB 支持此用法，用法如下：

```
UPDATE products SET price = price * 1.10
  WHERE price <= 99.99
  RETURNING name, price AS new_price;
```

具体可参考 https://www.hs.net/lightdb/docs/html/dml-returning.html。

外部数据源支持

LightDB 通过 FDW 支持 70 种外部数据源 (包括 MySQL, Oracle, CSV, hadoop …) 当成自己数据库中的表来查询。MySQL 不支持。LightDB 用法如下：

```
(postgres@[local]:5000) [postgres] > create extension file_fdw;
CREATE EXTENSION
Time: 752.715 ms
(postgres@[local]:5000) [postgres] > create server srv_file_fdw foreign data wrapper file_fdw;
CREATE SERVER
Time: 23.317 ms
(postgres@[local]:5000) [postgres] > create foreign table t_csv ( a int, b varchar(50) )
server srv_file_fdw
```

```
options ( filename '/var/tmp/data.csv', format 'csv' );
CREATE FOREIGN TABLE
Time: 74.843 ms
(postgres@[local]:5000) [postgres] > select count(*) from t_csv;
count
-------
1000
(1 row)
Time: 0.707 ms
(postgres@[local]:5000) [postgres] > create table t_csv2 as select * from t_csv;
SELECT 1000
Time: 147.859 ms
(postgres@[local]:5000) [postgres] > insert into t_csv values (-1,'a');
ERROR: cannot insert into foreign table "t_csv"
Time: 18.113 ms
(postgres@[local]:5000) [postgres] > explain select count(*) from t_csv;
QUERY PLAN
------------------------------------------------------------------------
Aggregate (cost=171.15..171.16 rows=1 width=0)
-> Foreign Scan on t_csv (cost=0.00..167.10 rows=1621 width=0)
Foreign File: /var/tmp/data.csv
Foreign File Size: 38893
(4 rows)
Time: 0.521 ms
```

管理平台

MySQL 社区版没有；Lightdb 有 EM 管理平台。

空闲会话超时管理

空闲会话超时主要是为了防止连接过多。MySQL 通过 wait_timeout 参数控制空闲会话超时，默认 8 小时。LightDB 支持空闲事务超时 idle_in_transaction_session_timeout，目前不支持空闲会话超时。

SQL 语句超时

SQL 语句超时用于控制 SQL 的执行时间，防止 SQL 执行时间过长，长时间加着锁，影响其实事务导致阻塞。超过超时时间，SQL 会被取消 MySQL 通过 max_execution_time 控制；LightDB 通过 statement_timeout 控制。

即时加字段 INSTANT ADD COLUMN

INSTANT 算法的优势在于，仅在数据字典中进行元数据更改。 table 更改期间无需获取元数据锁定，也不会 touch 表中的数。MySQL5.7 不支持，LightDB, MySQL8.0 支持，MySQL8.0 用法如下，默认即为 INSTANT：

```
ALTER TABLE products ADD COLUMN description text,ALGORITHM=INSTANT;
```

LightDB 默认支持，不需也不能指定 ALGORITHM。

表连接方式

MySQL 5.7 只支持循环嵌套（nestloop）。MySQL 8.0 支持循环嵌套和 hashjoin。Lightdb 支持循环嵌套，hashjoin 和 mergejoin。

优化器提示/跟踪

准确的说，MySQL 从 5.7 开始真正算引入了优化器提示，其用法和 oracle 的优化器提示完全相同，MySQL 8.0 对此进行了进一步加强，引入了更多的优化器提示。如下所示：

MySQL5.7	MySQL 8.0
BKA, NO_BKA	BKA, NO_BKA
BNL, NO_BNL	BNL, NO_BNL
MAX_EXECUTION_TIME	MAX_EXECUTION_TIME
MRR, NO_MRR	MRR, NO_MRR
NO_ICP	NO_ICP
NO_RANGE_OPTIMIZATION	NO_RANGE_OPTIMIZATION
QB_NAME	QB_NAME
SEMIJOIN, NO_SEMIJOIN	SEMIJOIN, NO_SEMIJOIN
SUBQUERY	SUBQUERY
	DERIVED_CONDITION_PUSHDOWN
	GROUP_INDEX, NO_GROUP_INDEX
	HASH_JOIN, NO_HASH_JOIN
	INDEX, NO_INDEX
	INDEX_MERGE, NO_INDEX_MERGE
	JOIN_FIXED_ORDER
	JOIN_INDEX, NO_JOIN_INDEX
	JOIN_ORDER
	JOIN_PREFIX
	JOIN_SUFFIX
	ORDER_INDEX, NO_ORDER_INDEX
	RESOURCE_GROUP
	SKIP_SCAN, NO_SKIP_SCAN
	SET_VAR

各优化器提示的含义参见：

https://dev.MySQL.com/doc/refman/5.7/en/optimizer-hints.html。

https://dev.MySQL.com/doc/refman/8.0/en/optimizer-hints.html。

对于 MySQL，用户可以用 EXPLAIN 检查优化器提示是如何影响执行计划的，如果要查看某个优化器提示是否被使用了，可以在执行 EXPLAIN 后执行 SHOW WARNINGS，

EXPLAIN EXTENDED 可以看到哪个提示被用了，如下：

```
mysql> explain
select /*+ JOIN_SUFFIX(a) BKA
 */* from a,b where a.id=b.id;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------
--------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
                          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------
--------------------------+
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |      100 | NULL
                          |
|  1 | SIMPLE      | a     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |      100 | Using where; Using join b
uffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------

mysql> show warnings;
+---------+------+-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------+
| Level   | Code | Message
                                                                                                               |
+---------+------+-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------+
| Warning | 1064 | Optimizer hint syntax error near '*/' at line 2
                                                                                                               |
| Note    | 1003 | /* select#1 */ select /*+ JOIN_SUFFIX(@`select#1` `a`) */ `test`.`a`.`id` AS `id`,`test`.`a`.`b` AS `b`,`test`.`b
`.`id` AS `id`,`test`.`b`.`b` AS `b` from `test`.`a` join `test`.`b` where (`test`.`a`.`id` = `test`.`b`.`id`) |
+---------+------+-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------+
2 rows in set

mysql> explain
select /*+ JOIN_SUFFIX(a) BKA(a
) */* from a,b where a.id=b.id;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------
--------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra
                          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------
--------------------------+
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |      100 | NULL
                          |
|  1 | SIMPLE      | a     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |      100 | Using where; Using join b
uffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------------------
--------------------------+
2 rows in set

mysql> show warnings;
+-------+------+-------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message
                                                                                                                                 |
+-------+------+-------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select /*+ JOIN_SUFFIX(@`select#1` `a`) BKA(`a`@`select#1`) */ `test`.`a`.`id` AS `id`,`test`.`a`.`b
` AS `b`,`test`.`b`.`id` AS `id`,`test`.`b`.`b` AS `b` from `test`.`a` join `test`.`b` where (`test`.`a`.`id` = `test`.`b`.`id`) |
+-------+------+-------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+
1 row in set
```

LightDB 也支持优化器提示，已覆盖常用的 hint，如各种 scan method，join method，join order 等，具体可参考 https://www.hs.net/lightdb/docs/html/pghint_plan.html。

缓存预热

MySQL 通过 innodb_buffer_pool_load_now 等参数预热，具体可参考 https://dev.MySQL.com/doc/refman/8.0/en/innodb-preload-buffer-pool.html。

Lightdb 通过使用 pg_prewarm 插件预热，具体可参考 https://www.hs.net/lightdb/docs/html/pgprewarm.html。

查看正在运行 SQL 的执行计划

MySQL 可以通过 EXPLAIN FOR CONNECTION <thread_id>查看，但是 MySQL 没有提供用户级别的。如下所示：

EXPLAIN [options] FOR CONNECTION connection_id;

Lightdb 可以通过 pg_show_plans 插件查看运行中 SQL 的执行计划，记录在 pg_show_plans 中，提供用户级查看。具体可参考 https://zhuanlan.zhihu.com/p/342527749。

文本数据导入导出

MySQL 通过 select into outfile，load data infile into。Lightdb 通过 copy to,copy from。

性能视图

MySQL 从 5.7 开始在 performance_schema 的基础上新增了 sys 性能库，它是基于 performance_schema 的视图，还包括一些存储过程，在此之前， 通常需要自己编写自定义 sql 查询 performance_schema，其友好性相对来说不是很好。在 MySQL 8.0 中，该库仍然存在并且进行了加强，具体可参考 https://dev.MySQL.com/doc/refman/8.0/en/sys-schema.html。

LightDB 通过 pg_stat_statements 插件，pg_stat_statements 插件提供了丰富的性能视图：如：等待事件，系统统计信息等。

执行计划与 analyze

Explain Analyze 会做出查询计划，并且会实际执行，以测量出查询计划中各个关键点的实际指标，例如耗时、条数，最后详细的打印出来。MySQL5.7 不支持。MySQL8.0 支持。LightDB 支持 Explain Analyze。语法相同。

进度报告

进度报告用来展示某些命令的执行进度，MySQL 只能在 processlist 中简单展示当前 sql 执行 STATE 。LightDB 目前可以展示 ANALYZE, CLUSTER, CREATE INDEX, VACUUM, and BASE_BACKUP 的进度报告，通过查询对应的视图来查看，如 pg_stat_progress_analyze。具体可查看 https://www.hs.net/lightdb/docs/html/progress-reporting.html。

索引类型

MySQL 取决于存储引擎，可以有 btree 索引，全文索引，表达式索引(需要建虚拟列)，hash 索引。 LightDB 多种索引类型（btree , hash , gin , gist , sp-gist , brin , bloom , rum , zombodb , bitmap，部分索引，表达式索引）。

只读表

默认情况下，数据库为了保证 MVCC，会检查所有数据行的事务可见性，当为大数据量查询时，对性能影响较大。可以通过设置为只读表，不去进行可读性检查。MySQL 不支持。LightDB 支持，通过 alter table xxx set read only。

Automatic Workload Repository&Active Session History

Automatic Workload Repository 通过对比两次快照的统计信息，了解数据库负载。Active Session History 记录活动的会话的活动信息。MySQL 不支持上述统计，LightDB 支持。

基准性能测试工具

MySQL: MySQLslap sysbench tpcc-MySQL benchmarksql 等。
LightDB: ltbench sysbench benchmarksql 等。

http://www.sqlines.com/mysql-to-postgresql