

MySQL 优化

主要内容

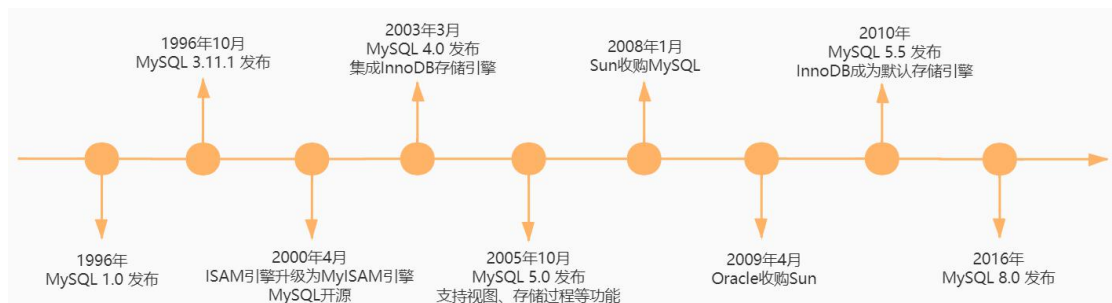
MySQL 架构分析
SQL 语句优化
MySQL 中的索引

学习目标

知识点	要求
MySQL 架构分析	掌握
SQL 语句优化	掌握
MySQL 中的索引	掌握

一、MySQL 架构分析

1 MySQL 的发展史



2 MySQL 分支

2.1 Percona Server-XtraDB

Percona Server 是 MySQL 重要的分支之一，它基于 InnoDB 存储引擎的基础上，提升了性能和易管理性，最后形成了增强版的 XtraDB 引擎，可以用来更好地发挥服务器硬件上的性能。所以 Percona Server 也可以称为增强的 MySQL 与开源的插件（plugin）的结合。

2.2 MariaDB

Mariadb 是由 MySQL 创始人 Monty 创建的，是一款高度兼容的 MySQL 产品，主要由开源社区维护，采用 GPL 授权许可。Oracle 把 MySQL 收购之后，为避免 MySQL 在开源粒度上的下降，MariaDB 由此而生。

2.3 InnoDB

InnoDB 是网易杭州研究院维护的 MySQL 分支。

3 MySQL 的连接

3.1 MySQL 连接介绍

3.1.1 通信类型

3.1.1.1 同步通信

同步通信依赖于被调用方，受限于被调用方的性能。也就是说，应用操作数据库，线程会阻塞，等待数据库的返回。

3.1.1.2 异步通信

异步可以避免应用阻塞等待，但是不能节省 SQL 执行的时间。如果异步存在并发，每一个 SQL 的执行都要单独建立一个连接，避免数据混乱。但是这样会给服务端带来巨大的压力（一个连接就会创建一个线程，线程间切换会占用大量 CPU 资源）。另外异步通信还带来了编码的复杂度，所以一般不建议使用。如果要异步，必须使用连接池，排队从连接池获取连接而不是创建新连接。

3.1.2 连接方式

3.1.2.1 短连接

短连接操作步骤：

连接-->数据传输-->关闭连接。

3.1.2.2 长连接

长连接操作步骤

连接-->数据传输-->保持连接-->数据传输-->保持连接-->.....-->关闭连接

3.1.3 协议

3.1.3.1 TCP/IP

TCP/IP 套接字连接方式是 MySQL 在任何平台都提供的一种连接方式，也是网络中使用最多的一种方式。

3.1.3.2 Unix Socket

UNIX Socket 是连接 MySQL 需要一个物理文件，文件的存放位置在配置文件中有定义，所以只能在 MySQL 客户端和数据库实例在同一台服务器上的情况下使用。它所有协议中最高效的一个。

3.2 在服务端中查看连接信息

3.2.1 查看服务端中的连接数

show global status like 'Thread%'

Variable_name	Value
Threads_cached	1
Threads_connected	2
Threads_created	3
Threads_running	1

Thread_cached：线程缓存中的线程数

Thread_connected：当前打开的连接数

Thread_created：为处理连接而创建的线程数

Thread_running：未休眠的线程数

3.2.2 优化 Threads_created

Variable_name	Value
Threads_cached	9
Threads_connected	11
Threads_created	3980503
Threads_running	1

threads_created 表示创建过的线程数，很明显，threads_created 过大，表明 mysql 服务器一直在创建线程，这也是比较耗资源，说明服务器不健康。

3.2.2.1 解决方案

配置 thread_cache_size：

thread_cache_size 作用是当客户端断开之后，服务器处理此客户的线程将会缓存起来以响应下一个客户而不是销毁（前提是缓存数未达上限）。MySQL<= 5.6.7 默认为 0，MySQL>= 5.6.8 默认值为-1（自动调整）。可设置为 0-16384。

查看 thread_cache_size：

show variables like 'thread_cache_size'

设置规则：

如果是短连接，适当设置大一点，因为短连接往往需要不停创建，不停销毁，如果大一点，连接线程都处于取用状态，不需要重新创建和销毁，所以对性能肯定是比较大的提升。

如果是长连接，可以设置成小一点，一般在 50-100 左右。

根据物理内存设置规则：

1G ---> 8
2G ---> 16
3G ---> 32
≥4G ---> 64

set global thread_cache_size=64 //重启 mysql 后失效。

3.2.3 查看连接超时时间

3.2.3.1 交互式

交互式：是指使用客户端工具连接数据库服务端。如：Navicat、SQLyog 等。

全局：

show global variables like 'interactive_timeout'

会话：

show session variables like 'interactive_timeout'

3.2.3.2 非交互式

非交互式：是指使用 JDBC 方式连接数据服务端。

全局：

show global variables like 'wait_timeout'

会话：

```
show session variables like 'wait_timeout'
```

3.2.4 优化连接超时时间

3.2.4.1 交互式

全局

```
set global interactive_timeout=600;
```

会话

```
set session interactive_timeout=600;
```

3.2.4.2 非交互式

全局

```
set global wait_timeout=600;
```

会话

```
set session wait_timeout=600;
```

3.2.4.3 总结

wait_timeout 的作用是，设置非交互连接（就是指那些连接池方式、非客户端方式连接的）的超时时间，默认是 28800，就是 8 小时，超过这个时间，mysql 服务器会主动切断那些已经连接的，但是状态是 sleep 的连接。

3.2.5 查看最大连接数

对客户端而言，MySQL 服务端的最大连接数是指服务端允许的最大连接并发数。

```
show variables like 'max_connections';
```

3.2.6 优化最大连接数

该参数设置过小的最明显特征是出现“Too many connections”异常。

如果服务器的并发连接请求量比较大，建议调高该值，以增加并行连接数量，当然这建立在机器能支撑的情况下，因为如果连接数越多，介于 MySQL 会为每个连接提供连接缓冲区，就会开销越多的内存，所以要适当调整该值，不能盲目提高设置。

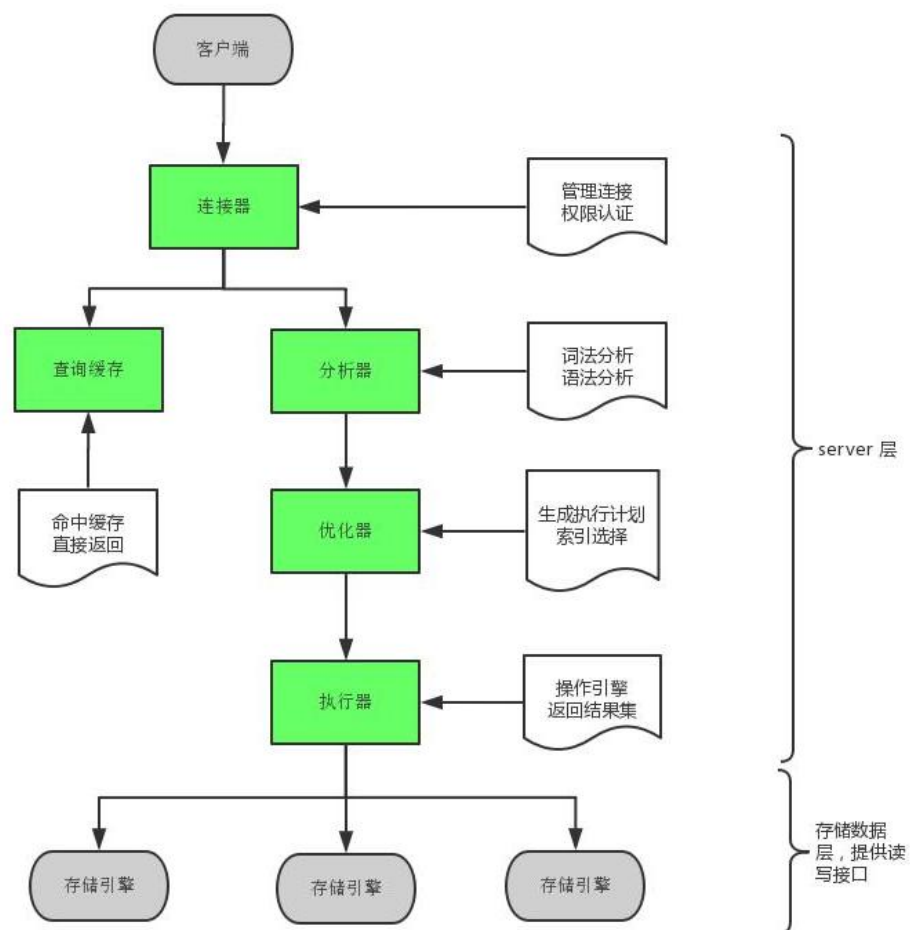
通过命令修改

```
set global max_connections = 200 ;
```

通过配置文件修改

```
max_connections=100;
```

4 MySQL 的架构以及内部模块



4.1 连接器

主要负责用户登录数据库，进行用户的身份认证，包括校验账户密码，权限等操作，如果用户账户密码已通过，连接器会到权限表中查询该用户的所有权限。

如果账号密码不对，就会抛出 Access denied for user 的异常。

如果账号密码正确，连接器就会读取当前用户此时所拥有的的权限，值得注意的是，在连接过程中，即使你用管理员账号修改当前用户的权限，丝毫不会影响它在本次连接的权限，你的修改需要等到下次连接才会生效。

如果你长时间没有操作数据库，这个连接自动断开，这个时间默认是 8 小时。这个时候你要操作数据库就必须重连。

4.2 查询缓存

连接建立后，执行查询语句的时候，会先查询缓存，Mysql 会先校验这个 sql 是否执行过，以 Key-Value 的形式缓存在内存中，Key 是查询预计，Value 是结果集。如果缓存 key 被命中，就会直接返回给客户端，如果没有命中，就会执行后续的操作，完成后也会把结果缓存起来，方便下一次调用。当然在真正执行缓存查询的时候还是会校验用户的权限，是否有该表的查询条件。

Mysql 查询不建议使用缓存，因为对于经常更新的数据来说，缓存的有效时间太短了，往往带来的效果并不好，对于不经常更新的数据来说，使用缓存还是可以的，Mysql 8.0 版本后删除了缓存的功能，官方也是认为该功能在实际的应用场景比较少，所以干脆直接删掉了。

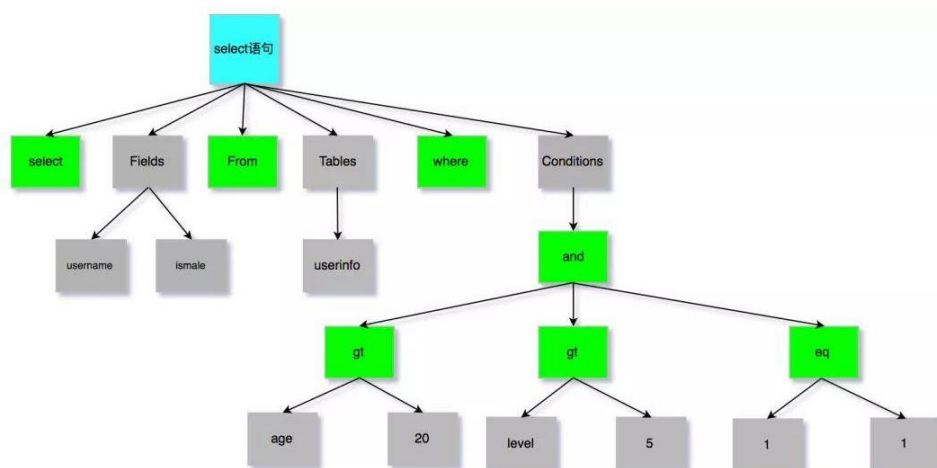
4.3 分析器

Mysql 没有命中缓存，那么就会进入分析器，分析器主要是用来分析 SQL 语句是来干嘛

的，分析器也会分为几步：

第一步，词法分析，一条 SQL 语句有多个字符串组成，首先要提取关键字，比如 select，提出查询的表，提出字段名，提出查询条件等等。做完这些操作后，就会进入第二步。

第二步，语法分析，主要就是判断你输入的 sql 是否正确，是否符合 mysql 的语法。



4.4 优化器

优化器的作用就是它认为的最优的执行方案去执行（虽然有时候也不是最优），比如多个索引的时候该如何选择索引，多表查询的时候如何选择关联顺序等。

4.5 执行器

当选择了执行方案后，mysql 就准备开始执行了，首先执行前会校验该用户有没有权限，如果没有权限，就会返回错误信息，如果有权限，就会去调用引擎的接口，返回接口执行的结果。

5 MySQL 中查询缓存优化

开启 mysql 缓存后，数据没有更新的情况下，相同的查询 sql 会使用缓存数据返回结果。在数据更新较少，类似查询较多的情况下，使用 mysql 缓存可以显著提升查询效率。

5.1 查询缓存配置信息

show variables like '%query_cache%';

信息	结果1	概况	状态
Variable_name	Value		
▶ have_query_cache	YES		
query_cache_limit	1048576		
query_cache_min_res_unit	4096		
query_cache_size	0		
query_cache_type	OFF		
query_cache_wlock_invalidate	OFF		

- have_query_cache 表示是否支持查询缓存，YES 表示支持
- query_cache_type 表示缓存类型，OFF 表示关闭查询缓存，ON 表示开启查询缓存，DEMAND 表示用户自定义查询缓存
- query_cache_limit 表示支持的最大单条查询 sql 数据量
- query_cache_min_res_unit 表示查询缓存最小单位
- query_cache_size 表示查询缓存空间大小
- query_cache_wlock_invalidate 表示查询缓存是否支持写锁，OFF 表示不支持，即读取数据不考虑写锁，ON 表示支持，即读取数据会被写锁阻塞

5.2 开启查询缓存

查看当前的 MySQL 数据库是否支持查询缓存

show variables like 'have_query_cache';

查看当前 MySQL 是否开启了查询缓存

show variables like 'query_cache_type';

在 Mysql 配置文件文件中开启查询缓存

- query_cache_type=0(OFF)关闭

- query_cache_type=1(ON)缓存所有结果，除非 select 语句使用 SQL_NO_CACHE 禁用查询缓存
- query_cache_type=2(DEMAND)，只缓存 select 语句中通过 SQL_CACHE 指定需要缓存的查询

配置查询缓存的占用空间单位 M

query_cache_size=10M

5.3 查询缓存使用

只有字符串相等查询的 sql 才使用相同缓存，即 select name from users 与 SELECT name FROM users 不使用同一个缓存。

在 query_cache_type 为 ON 的情况下，默认所有查询都使用缓存，我们可以使用 sql_no_cache 显示指定某个查询不使用缓存 select sql_no_cache name from users;

在 query_cache_type 为 DEMAND 的情况下，需要使用 sql_cache 指定某个查询使用缓存 select sql_cache name from users;

查看查询缓存的状态变量

show status like 'Qcache%';

信息	结果1	概况	状态
Variable_name Value			
Qcache_free_blocks	1		
Qcache_free_memory	10467280		
Qcache_hits	2		
Qcache_inserts	1		
Qcache_lowmem_prunes	0		
Qcache_not_cached	10		

- Qcache_free_blocks 表示已分配内存块中空闲块数量；

- Qcache_total_blocks 表示当前查询缓存占用的内存块数量；
- Qcache_free_memory 表示缓存空闲空间大小；
- Qcache_hits 表示缓存命中次数；
- Qcache_inserts 表示缓存未命中时，数据写入缓存次数；
- Qcache_queries_in_cache 表示缓存查询语句数量；
- Qcache_lowmem_prunes 表示缓存修剪次数，缓存满时，会使用 LRU 算法移除最久未被使用缓存，此值较大，说明缓存空间太小；
- Qcache_not_cached 表示没有被缓存的查询 sql 数量；

5.4 查询缓存失效的情况

SQL 语句不一致的情况，要想命中查询缓存，查询的 SQL 语句必须一致。

SQL1 : select count(*) from users;

SQL2 : Select count(*) from users;

当查询语句中有一些不确定的操作时，则不会缓存。如：`now()`，`current_date()`，

`curdate()`，`curtime()`，`rand()`，`uuid()`，`user()`，`database()`。

SQL1 : select * from tb_item where updatetime < now() limit 1;

SQL2 : select user();

SQL3 : select database();

不使用任何表查询语句。

select 'a'

查询 mysql，information_schema 或 performance_schema 数据库中的表时，不会走查询缓存。

select * from information_schema.engines;

在存储的函数，触发器或事件的主体内执行的查询。

如果表更改，则使用该表的所有高速缓存查询都将变为无效并从高速缓存中删除。这包括使用 MERGE 映射到已更改表的表的查询。一个表可以被许多类型的语句，如被改变 INSERT，UPDATE，DELETE，TRUNCATE TABLE，ALTER TABLE，DROP TABLE，或 DROP DATABASE

二、SQL 语句优化

1 SQL 语句优化简介

1.1 SQL 语句优化解释

- 对于特定的要求，使用更优的 SQL 策略或索引策略，以达到让结果呈现的时间更短，从而提升操作效率的过程就是 SQL 优化。
- SQL 优化包含在数据库级别优化中。我们平常所说的 SQL 优化就是指优化 SQL 语句和索引。
- SQL 优化是伴随着业务而进行优化的，并不是下面的所有操作就必须都达到才是好的优化。

1.2 常规调优思路

- 查看 slow-log，分析 slow-log，分析出查询慢的语句。
- 按照一定优先级，进行一个一个的排查所有慢语句。
- 分析 top sql，进行 explain 调试，查看语句执行时间。
- 调整索引或语句本身。

2 MySQL 日志支持

2.1 MySQL 5.7 中日志分类

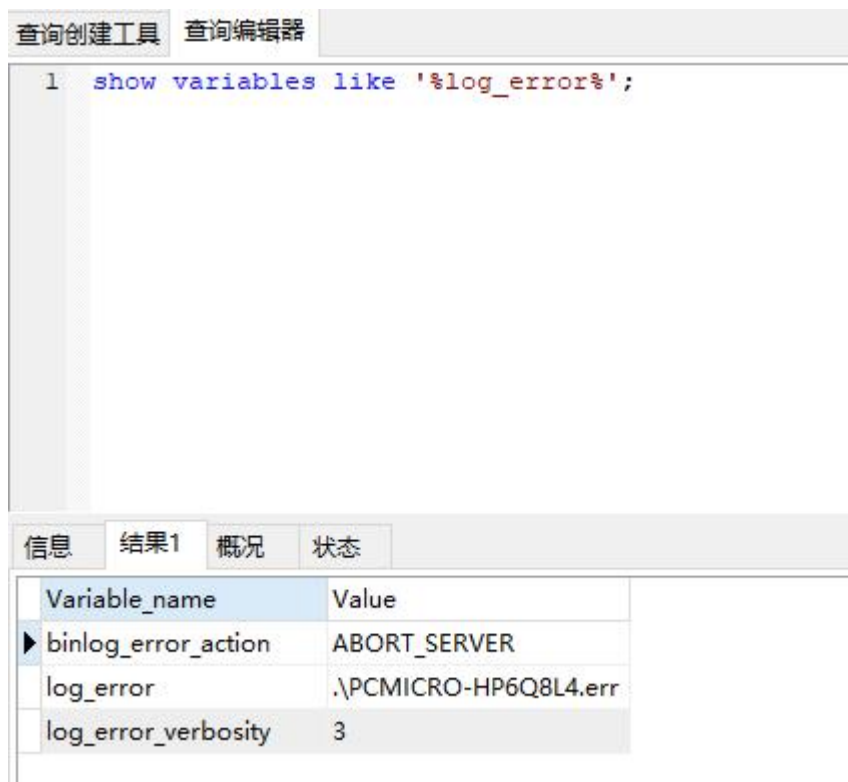
MySQL 中日志分为四类：错误日志、二进制日志、通用查询日志、慢查询日志。

2.2 错误日志

MySQL 错误日志是记录 MySQL 运行过程中较为严重的警告和错误信息，以及 MySQL 每次启动和关闭的详细信息。错误日志的命名通常为 hostname.err。其中，hostname 表示服务器主机名。

查看错误日志相关信息命令

```
show variables like '%log_error%';
```



The screenshot shows a MySQL query tool interface. At the top, there are tabs for '查询创建工具' and '查询编辑器'. The '查询编辑器' tab is active, showing the SQL command: `1 show variables like '%log_error%';`. Below the editor, there is a table with the results of the query. The table has two columns: 'Variable_name' and 'Value'. The results are as follows:

Variable_name	Value
binlog_error_action	ABORT_SERVER
log_error	.\PCMICRO-HP6Q8L4.err
log_error_verbosity	3

- binlog_error_action 错误处理方式。ABORT_SERVER 出现问题终止服务，IGNORE_ERROR 忽略错误。

- `log_error` 错误日志文件名及路径。
- `log_error_verbosity` 记录级别。取值 1 表示记录警告信息。大于 1 表示所有警告信息都记录。

修改错误日志路径

通过 MySQL 配置文件修改错误日志路径。

`log_error=[路径]/[日志文件名].err`

2.3 二进制日志

包含所有更新数据（新增、删除、修改、改表等）SQL 信息的记录。MySQL 主从配置就依赖这个日志文件。

查看二进制日志信息命令

```
show variables like '%log_bin%';
```

查询创建工具

查询编辑器

```

1  show variables like '%log_bin%';
2
3

```

信息

结果1

概况

状态

Variable_name	Value
log_bin	OFF
log_bin_basename	
log_bin_index	
log_bin_trust_function_calls	OFF
log_bin_use_v1_row_events	OFF
sql_log_bin	ON

二进制日志不可以通过修改全局参数开启。全局配置文件（my.ini）中该参数是注释的。

```

# Binary Logging.
# log-bin

```

直接设置 log-bin 的值为日志文件名。

```

# Binary Logging.
log-bin=mylogbin

```

设置后重启 MySQL 服务会发现 log_bin 参数值为 ON

Variable_name	Value
log_bin	ON
log_bin_basename	C:\ProgramData\MySQL\MySQL Server 5.7\binlog
log_bin_index	C:\ProgramData\MySQL\MySQL Server 5.7\binlog
log_bin_trust_function	OFF
log_bin_use_v1_row	OFF
sql_log_bin	ON

查看当前服务器使用的二进制文件及大小

```
show binary logs;
```

开启后二进制文件存储在 C:\ProgramData\MySQL\MySQL Server 5.7\Data。里面有个 xxx.index 文件（这个文件称为二进制文件索引）里面存储了所有二进制文件清单。当重启 MySQL 服务或过一定时间后会自动增加一个二进制文件。增加的二进制文件编号递增。也可以使用 flush logs;命令生成一个新的二进制文件。

由于是二进制日志文件是二进制文件，所以无法直接使用文本编辑器查看日志文件内容，可以通过命令：show binlog events in 'mylogbin.000001';或者借助工具来查看二进制日志文件中的内容。

在 navicat 或 sql 命令中输入

```
show binlog events in 'mylogbin.000001';
```

借助 mysqlbinlog 工具

```
mysqlbinlog mylogbin.000001
```

binlog 中除了删除表、创建表的 SQL 都是加密的，如果希望看见可以使用下面命令

```
mysqlbinlog --base64-output=decode-rows -v mylogbin.000003
```

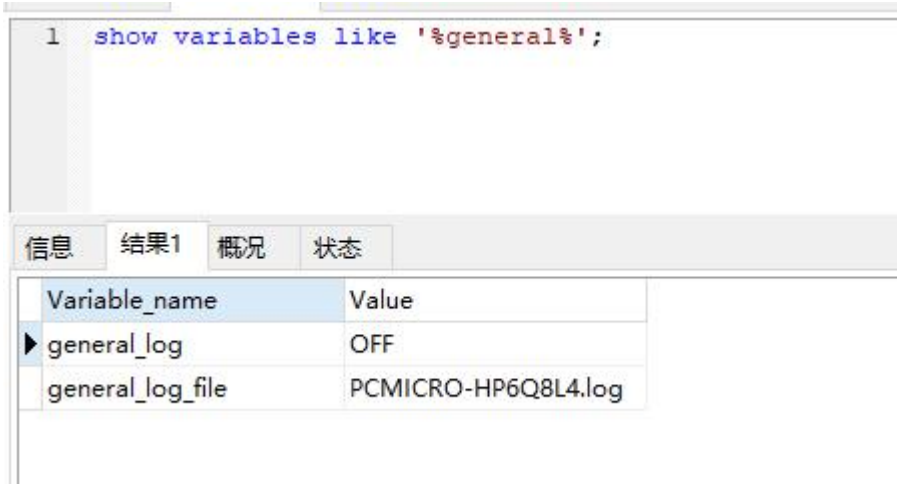
直接打印到控制台会出现中文乱码。

2.4 通用查询日志

通用查询日志是记录建立的客户端连接和执行的语句。

查看通用查询日志是否开启

show variables like '%general%';




Variable_name	Value
general_log	OFF
general_log_file	PCMICRO-HP6Q8L4.log

general_log 属性取值：OFF 表示关闭（默认关闭），ON 表示打开。

通过查看日志输出格式

show variables like '%log_output%';



Variable_name	Value
log_output	FILE

FILE：

存储在 C:\ProgramData\MySQL\MySQL Server 5.7\Data 中，主机名.log

TABLE：

存储在数据库中的 mysql/general_log

临时开启/关闭通用日志（重启失效）

```
# 开启
set global general_log=on;

# 关闭
set global general_log=off;
```

临时设置输出格式（重启失效）

```
# mysql/general_log
set global log_output='TABLE';
# 主机名-slow.log
set global log_output='FILE';
# 两者都输出
set global log_output='FILE,TABLE';
```

永久设置，修改全局配置文件 my.ini 配置文件

```
# General and Slow logging.
log-output=FILE
general-log=0
general_log_file="PCMICRO-HP6Q8L4.log"
```

日志文件中时间和系统时间不一致问题

查看系统日志文件格式

```
show variables like '%log_timestamps%';
```

修改日志文件时间格式为系统时间

```
set global log_timestamps = SYSTEM;
```

2.5 慢查询日志

- 记录所有执行时间超过 long_query_time 秒的所有查询或不适用于索引的查询。
- long_query_time 默认时间为 10 秒。即超过 10 秒的查询都认为是慢查询。
- 慢查询日志默认名称：主机名-slow.log。

```
# General and Slow logging.
log-output=FILE
general-log=1
general_log_file="PCMICRO-HP6Q8L4.log"
slow-query-log=1
slow_query_log_file="PCMICRO-HP6Q8L4-slow.log"
long_query_time=10
```

除了查看 my.ini 文件以外通过 show variables like '%quer%';命令查看

查询创建工具 查询编辑器	
1 show variables like '%quer%';	
信息 结果1 概况 状态	
Variable_name	Value
binlog_rows_query_log_e	OFF
ft_query_expansion_limit	20
have_query_cache	YES
log_queries_not_using_inc	OFF
log_throttle_queries_not_i	0
long_query_time	10.000000
query_alloc_block_size	8192
query_cache_limit	1048576
query_cache_min_res_unit	4096
query_cache_size	10485760
query_cache_type	DEMAND
query_cache_wlock_invali	OFF
query_prealloc_size	8192
slow_query_log	ON
slow_query_log_file	PCMICRO-HP6Q8L4-slow

- slow_query_log 表示是否开启慢查询日志。（默认开启）
- slow_query_log_file 慢查询日志文件名
- long_query_time 慢查询阈值设置，查出为慢查询。此值直接设置全局参数可能无效，建议测试时直接修改配置文件。
- log_queries_not_using_indexes 是否记录不适用于索引的查询（前提 slow_query_log 开启）

查看慢查询日志内容

```
# Time: 2021-04-25T03:01:37.640859Z
# User@Host: root[root] @ localhost [127.0.0.1] Id: 2
# Query_time: 1.646116 Lock_time: 0.000000 Rows_sent: 368247 Rows_examined: 736494
SET timestamp=1619319697;
select * from users u1,users u2 where u1.userid = u2.userid;
```

第一行,SQL 查询执行的时间。

第二行,执行 SQL 查询的连接信息,用户和连接 IP。

第三行,记录了一些我们比较有用的信息,如下解析:

- Query_time,这条 SQL 执行的时间,越长则越慢;
- Lock_time,在 MySQL 服务器阶段(不是在存储引擎阶段)等待表锁时间;
- Rows_sent,查询返回的行数;
- Rows_examined,查询检查的行数,越长就当然越费时间;

第四行,设置时间戳,没有实际意义,只是和第一行对应执行时间。

第五行及后面所有行(第二个# Time:之前),执行的 sql 语句记录信息。

3 执行计划

3.1 执行计划简介

我们可以利用 explain 关键字可以模拟优化器执行 SQL 查询语句,来分析 sql 慢查询语句。

3.2 语法结构

EXPLAIN SELECT 投影列 FROM 表名 WHERE 条件

3.3 字段解释

3.3.1 select_type

查询类型，主要用于区分普通查询、联合查询、子查询等。

3.3.2 table

查询的哪个表。

3.3.3 type

查询类型，SQL 优化中非常重要的指标。

从好到坏排序。一般要求至少是 range 级别，最好能达到 ref 级别

system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL

3.3.4 key

实际使用了的索引。

3.3.5 rows

需要扫描的行数。

4 慢查询优化

4.1 索引失效

a) 使用 LIKE 关键字的查询语句。

在使用 LIKE 关键字进行查询的查询语句中，如果匹配字符串的第一个字符为“%”，索引不会起作用。只有“%”不在第一个位置索引才会起作用。

SELECT * FROM users WHERE username LIKE 'old%' -- 使用索引

```
SELECT * FROM users WHERE username LIKE '%lu' -- 不使用索引
```

b) 索引列参与计算

如果对条件中的列进行了计算，则不会使用该列的索引。

```
SELECT * FROM users WHERE userage=20;-- 使用索引
```

```
SELECT * FROM users WHERE userage=10+20;-- 使用索引
```

```
SELECT * FROM users WHERE userage+10=30;-- 不会使用索引
```

c) 索引列使用了函数

如果对条件中的列使用了函数，则不会使用该列的索引。

```
SELECT * FROM users WHERE username=concat('old','lu'); -- 使用索引
```

```
SELECT * FROM users WHERE concat(username,'lu')='oldlu'; -- 不会使用索引
```

d) 尽量避免 OR 操作

```
SELECT * FROM users WHERE username='oldlu' or userage=20 or usersex='male'
```

--如果条件中有 or 则要求使用的所有字段都必须建立索引，否则不使用索引。

除非每个列都建立了索引，否则不建议使用 OR，在多列 OR 中，可以考虑用 UNION 替换。

```
SELECT * FROM users WHERE username='oldlu' UNION
```

```
SELECT * FROM users WHERE userage=20 UNION
```

```
SELECT * FROM users WHERE usersex='male'
```

4.2 分解关联查询

将一个大的查询分解为多个小查询是很有必要的。

很多高性能的应用都会对关联查询进行分解，就是可以对每一个表进行一次单表查询，

然后将查询结果在应用程序中进行关联，很多场景下这样会更高效。

```
SELECT * FROM users u
JOIN orders o ON u.userid = o.user_id
JOIN item i ON o.orderid = i.order_id
WHERE u.userid= 200;
```

分解为：

```
SELECT * FROM users WHERE userid = 200;
SELECT * FROM order WHERE user_id = 200;
SELECT * FROM item WHERE order_id in (12321,32412,32321);
```

4.3 优化 LIMIT 分页

在系统中需要分页的操作通常会使用 limit 加上偏移量的方法实现，同时加上合适的 order by 子句。如果有对应的索引，通常效率会不错。

如果当偏移量非常大的时候，例如可能是 limit 1000000,20 这样的查询，这是 mysql 需要查询 1000020 条然后只返回最后 20 条，前面的 1000000 条记录都将被舍弃，这样的代价很高。

使用子查询优化大数据量分页查询

```
select * from users where username = 'oldlu' limit 1000000,100;
select * from users where username = 'oldlu' and userid >= (select userid from users where
username = 'oldlu' LIMIT 1000000,1) LIMIT 100;
```

使用 id 限定优化大数据量分页查询

使用这种方式需要先假设数据表的 id 是连续递增的，我们根据查询的页数和查询的记录数可以算出查询的 id 的范围，可以使用 id between and 来查询。

```
select * from users where username = 'oldlu' and (userid between 1000001 and 1000100)
LIMIT 100;
```


三、 MySQL 中的索引

1 索引概述

1.1 索引的优点

为什么要创建索引？这是因为，创建索引可以大大提高系统的查询性能。如果不使用索引，查询时从第一行开始查询。如果使用了索引就可以更加快速的找到希望的数据

第一、通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二、可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。

第三、可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四、在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五、通过使用索引，可以在查询的过程中，使用查询优化器，提高系统的性能。

1.2 索引的缺点

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？这种想法固然有其合理性，然而也有其片面性。虽然，索引有许多优点，但是，为表中的每一个列都增加索引，是非常不明智的。这是因为，增加索引也有许多不利的一个方面：

第一、创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二、索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间。如果要建立聚簇索引，那么需要的空间就会更大。

第三、当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

2 索引选择

2.1 什么样的字段适合创建索引

索引是建立在数据库表中的某些列的上面。因此，在创建索引的时候，应该仔细考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，应该在具备下述特性的列上创建索引：

- 第一、在经常需要搜索的列上，可以加快搜索的速度；
- 第二、在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；
- 第三、在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；
- 第四、在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；
- 第五、在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；
- 第六、在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

建立索引，一般按照 select 的 where 条件来建立，比如：select 的条件是 where f1 and f2，那么如果我们在字段 f1 或字段 f2 上建立索引是没有用的，只有在字段 f1 和 f2 上同时建立索引才有用等。

2.2 什么样的字段不适合创建索引

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的这些列具有下述特点：

- 第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反

而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

3 MySQL 中的索引类型

3.1 单列索引

就是给一个列添加索引。

- 普通索引：不考虑过多情况，主要是为了让查询更快一些。
- 唯一索引：列中值不可以重复，可以是 null
- 主键索引：列中值不可以重复，又不可以为 null

3.2 组合索引

给表中大于等于两个列添加索引。但是需要满足最左前缀。

3.3 全文索引 (full-text)

- 只能在 char、varchar、text 等字段才可以使用全文索引
- 全文索引是抽取一列内容的关键字，通过关键字建立索引。例如：我们都在尚学堂学习。

当搜索尚学堂时就可以搜索这句话。所以全文索引适用于含有 like 的查询。可以解决
'xxxx%' 模糊查询低效的问题。

4 索引管理

建立索引可以在建立表时直接指定索引,也可以后期添加索引,注意主键约束自动带有主键索引,唯一约束自动带有唯一索引。

在 MySQL 中,对索引的查看和删除操作是所有索引类型通用的。

4.1 普通索引

这是最基本的索引,它没有任何限制 MySQL 中默认的 BTREE 类型的索引,也是我们大多数情况下用到的索引。

直接创建索引

```
CREATE INDEX index_name ON table(column(length));
```

修改表结构的方式添加索引

```
ALTER TABLE table_name ADD INDEX index_name ON (column(length));
```

创建表的时候同时创建索引

```
CREATE TABLE `table` (
  `id` int(11) NOT NULL AUTO_INCREMENT ,
  `title` char(255) CHARACTER NOT NULL ,
  `content` text CHARACTER NULL ,
  `time` int(10) NULL DEFAULT NULL ,
  PRIMARY KEY (`id`),
  INDEX index_name (title(length))
);
```

查看索引

```
SHOW INDEX FROM table_name;
```

```
SHOW KEYS FROM table_name; # 只在 MySQL 中可以使用 keys 关键字。
```

删除索引

```
DROP INDEX index_name ON table;
ALTER TABLE table_name DROP INDEX index_name;
ALTER TABLE table_name DROP PRIMARY KEY;
```

4.2 唯一索引

与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

创建唯一索引

```
CREATE UNIQUE INDEX indexName ON table(column(length));
```

修改表结构

```
ALTER TABLE table_name ADD UNIQUE indexName ON (column(length));
```

创建表的时候直接指定

```
CREATE TABLE `table` (
  `id` int(11) NOT NULL AUTO_INCREMENT ,
  `title` char(255) CHARACTER NOT NULL ,
  `content` text CHARACTER NULL ,
  `time` int(10) NULL DEFAULT NULL ,
  UNIQUE indexName (title(length))
);
```

4.3 主键索引

是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值。一般是在建表的时候同时创建主键索引。

```
CREATE TABLE `table` (
  `id` int(11) NOT NULL AUTO_INCREMENT ,
  `title` char(255) NOT NULL ,
```

```
PRIMARY KEY (`id`)  
);
```

4.4 组合索引

指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用组合索引时遵循最左前缀匹配原则。

```
ALTER TABLE table_name ADD INDEX index_name (name,city,age);
```

4.5 全文索引

MySQL 从 3.23 版开始支持全文索引和全文检索，FULLTEXT 索引在 MySQL5.6 之前仅可用于 MyISAM 表，在 MySQL5.7 后 InnoDB 也支持；他们可以从 CHAR、VARCHAR 或 TEXT 列中作为 CREATE TABLE 语句的一部分被创建，或是随后使用 ALTER TABLE 或 CREATE INDEX 被添加。

对于较大的数据集，将你的资料输入一个没有 FULLTEXT 索引的表中，然后创建索引，其速度比把资料输入现有 FULLTEXT 索引的速度更为快。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。

全文索引对中文支持不好，如果搜索中文就只能按照最左对照进行搜索。如果是英文就可以匹配中间。

直接创建全文索引

```
CREATE FULLTEXT INDEX index_content ON table(column);
```

修改表结构添加全文索引

```
ALTER TABLE table ADD FULLTEXT index_content(column);
```

创建表时添加全文索引

```
CREATE TABLE `table` (
```

```
`id` int(11) NOT NULL AUTO_INCREMENT ,
`title` char(255) CHARACTER NOT NULL ,
`content` text CHARACTER NULL ,
`time` int(10) NULL DEFAULT NULL ,
PRIMARY KEY (`id`),
FULLTEXT (content)
);
```

5 索引优化

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点。虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行 INSERT、UPDATE 和 DELETE 次数大于查询次数时，放弃索引。因为更新表时，MySQL 不仅要保存数据，还要保存一下索引文件。建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的 MySQL 有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。

5.1 使用短索引（前缀索引）

对字符串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个 CHAR(255) 的列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和 I/O 操作。

```
CREATE INDEX index_name ON table_name (column(length));
```

5.2 索引列排序

MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 order by 中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建组合索引。

5.3 like 语句操作

一般情况下不鼓励使用 like 操作 如果非使用不可 如何使用也是一个问题。like “%aaa%” 不会使用索引，而 like “aaa%”（非前导模糊查询）可以使用索引。

5.4 不要在列上进行运算

例如：select * from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select * from users where adddate<'2007-01-01'。

应该把计算放在业务代码上完成，而不是交给数据库

5.5 范围列可以使用索引

范围条件有：<、<=、>、>=、between 等。

范围列可以用到索引（联合索引必须是最左前缀），但是范围列后面的列无法用到索引，索引最多用于一个范围列，如果查询条件中有两个范围列则无法全用到索引。所以 where 中把最主要的查询条件放在第一个。

5.6 类型转换会导致索引无效

当列是文本类型时，把数值类型当作列的条件会弃用索引

```
explain select * from teacher where name = 20;
```

5.7 优化总结

最后总结一下，MySQL 只对以下操作符才使用索引：<,<=,>,>=,between,in,以及某些时候的 like(不以通配符%或_开头的情形)。而理论上每张表里面最多可创建 16 个索引，不过除非是数据量真的很多，否则过多的使用索引也不是那么好玩的。

建议：一个表的索引数最好不要超过 6 个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。