1.MySQL环境

1.1.环境安装

```
# 查看Linux服务器上是否安装过MySQL
rpm -qa | grep -i mysql # 查询出所有mysql依赖包

# 1、拉取镜像
docker pull mysql:5.7

# 2、创建实例并启动
docker run -p 3306:3306 --name mysql \
-v /root/mysql/log:/var/log/mysql \
-v /root/mysql/data:/var/lib/mysql \
-v /root/mysql/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=333 \
-d mysql:5.7

# 3、mysql配置 /root/mysql/conf/my.conf
[client]
#mysqlde utf8字符集默认为3位的，不支持emoji表情及部分不常见的汉字，故推荐使用utf8mb4
default-character-set=utf8

[mysql]
default-character-set=utf8

[mysqld]
#设置client连接mysql时的字符集,防止乱码
init_connect='SET collation_connection = utf8_general_ci'
init_connect='SET NAMES utf8'

#数据库默认字符集
character-set-server=utf8

#数据库字符集对应一些排序等规则，注意要和character-set-server对应
collation-server=utf8_general_ci

# 跳过mysql程序起动时的字符参数设置 ，使用服务器端字符集设置
skip-character-set-client-handshake

# 禁止MySQL对外部连接进行DNS解析，使用这一选项可以消除MySQL进行DNS解析的时间。但需要注意，如果开启该选项，则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求!
skip-name-resolve

# 4、重启mysql容器
docker restart mysql

# 5、进入到mysql容器
docker exec -it mysql /bin/bash

# 6、查看修改的配置文件
cat /etc/mysql/my.conf
```

1.2.安装位置

`Docker`容器就是一个小型的`Linux`环境，进入到`MySQL`容器中。

```
docker exec -it mysql /bin/bash
```

`Linux`环境下`MySQL`的安装目录。

路径	解释
`/var/lib/mysql`	MySQL数据库文件存放位置
`/usr/share/mysql`	错误消息和字符集文件配置
`/usr/bin`	客户端程序和脚本
`/etc/init.d/mysql`	启停脚本相关

1.3.修改字符集

```
# 1、进入到mysql数据库并查看字符集
# show variables like 'character%';
# show variables like '%char%';

mysql> show variables like 'character%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | utf8                       |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | utf8                       |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.00 sec)

mysql> show variables like '%char%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | utf8                       |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | utf8                       |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.01 sec)
```

`MySQL5.7`配置文件位置是`/etc/my.cnf`或者`/etc/mysql/my.cnf`，如果字符集不是`utf-8`直接进入配置文件修改即可。

```
[client]
default-character-set=utf8

[mysql]
default-character-set=utf8

[mysqld]
# 设置client连接mysql时的字符集,防止乱码
init_connect='SET NAMES utf8'
init_connect='SET collation_connection = utf8_general_ci'

# 数据库默认字符集
character-set-server=utf8

#数据库字符集对应一些排序等规则，注意要和character-set-server对应
collation-server=utf8_general_ci

# 跳过mysql程序起动时的字符参数设置 ，使用服务器端字符集设置
skip-character-set-client-handshake

# 禁止MySQL对外部连接进行DNS解析，使用这一选项可以消除MySQL进行DNS解析的时间。但需要注意，如果开启该选项，则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求！
skip-name-resolve
```

注意：安装`MySQL`完毕之后，第一件事就是修改字符集编码。

1.4.配置文件

`MySQL`配置文件讲解：https://www.cnblogs.com/gaoyuechen/p/10273102.html

1、二进制日志`log-bin`：主从复制。

```
# my,cnf
# 开启mysql binlog功能
log-bin=mysql-bin
```

2、错误日志`log-error`：默认是关闭的，记录严重的警告和错误信息，每次启动和关闭的详细信息等。

```
# my,cnf
# 数据库错误日志文件
log-error = error.log
```

3、查询日志`log`：默认关闭，记录查询的`sql`语句，如果开启会降低`MySQL`整体的性能，因为记录日志需要消耗系统资源。

```
# my,cnf
# 慢查询sql日志设置
slow_query_log = 1
slow_query_log_file = slow.log
```

4、数据文件。

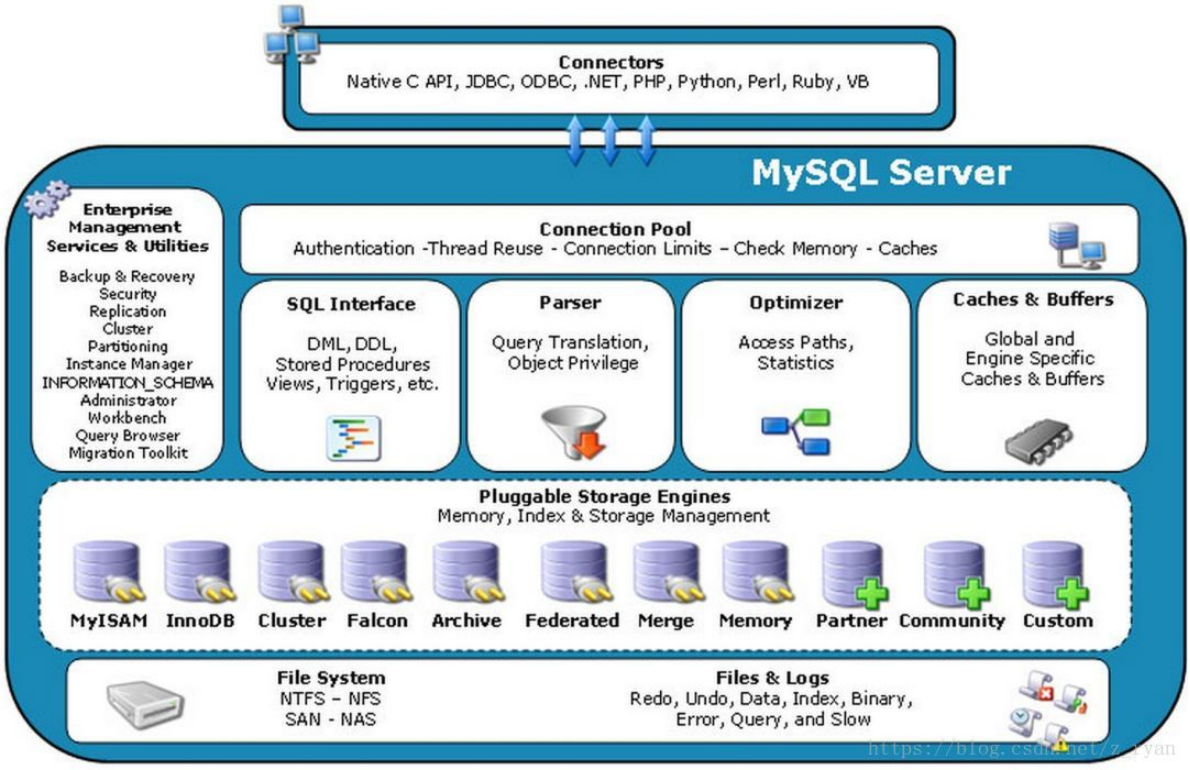
- `frm文件`：存放表结构。
- `myd`文件：存放表数据。
- `myi`文件：存放表索引。

```
# mysql5.7 使用.frm文件来存储表结构
# 使用 .ibd文件来存储表索引和表数据
-rw-r-----  1 mysql mysql   8988 Jun 25 09:31 pms_category.frm
-rw-r-----  1 mysql mysql 245760 Jul 21 10:01 pms_category.ibd
```

`MySQL5.7`的`Innodb`存储引擎可将所有数据存放于`ibdata*`的共享表空间，也可将每张表存放于独立的`.ibd`文件的独立表空间。 共享表空间以及独立表空间都是针对数据的存储方式而言的。

- 共享表空间: 某一个数据库的所有的表数据，索引文件全部放在一个文件中，默认这个共享表空间的文件路径在`data`目录下。 默认的文件名为`:ibdata1` 初始化为`10M`。
- 独立表空间: 每一个表都将会生成以独立的文件方式来进行存储，每一个表都有一个`.frm`表描述文件，还有一个`.ibd`文件。 其中这个文件包括了单独一个表的数据内容以及索引内容，默认情况下它的存储位置也是在表的位置之中。在配置文件`my.cnf`中设置： `innodb_file_per_table`。

2.MySQL逻辑架构



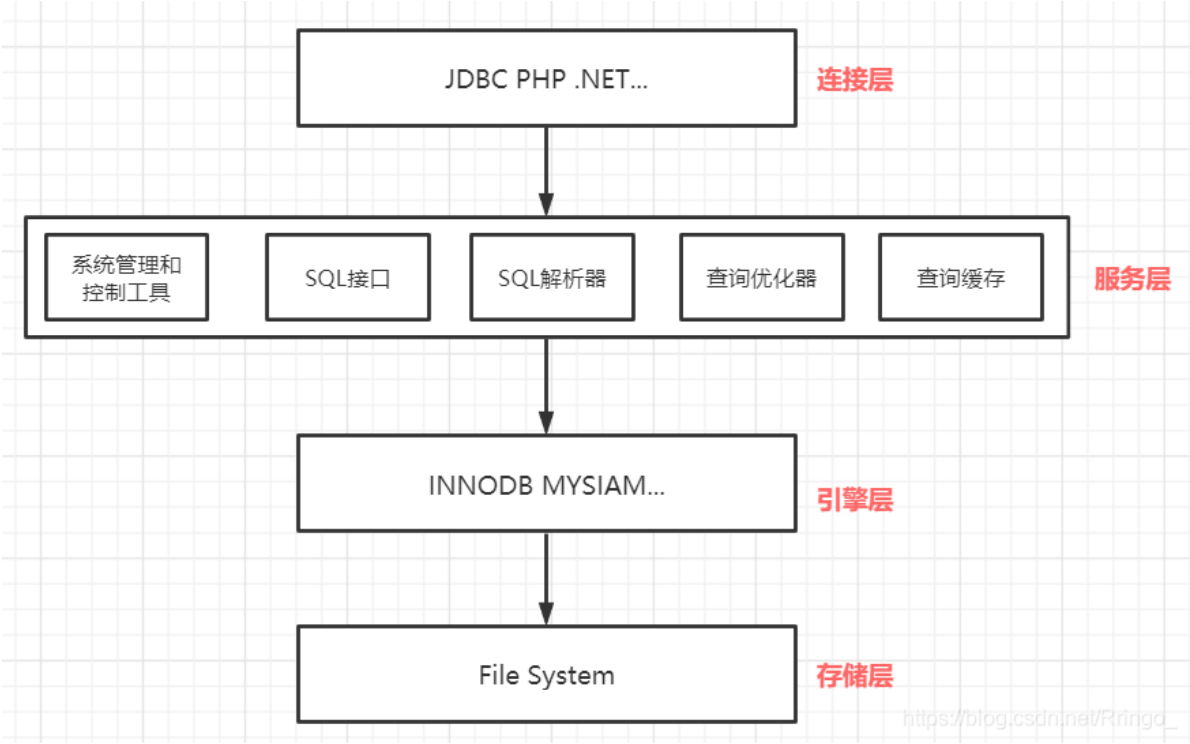
- `Connectors`：指的是不同语言中与SQL的交互。
- `Connection Pool`：管理缓冲用户连接，线程处理等需要缓存的需求。**MySQL数据库的连接层。**
- `Management Serveices & Utilities`：系统管理和控制工具。备份、安全、复制、集群等等。。
- `SQL Interface`：接受用户的SQL命令，并且返回用户需要查询的结果。
- `Parser`：SQL语句解析器。
- `Optimizer`：查询优化器，SQL语句在查询之前会使用查询优化器对查询进行优化。**就是优化客户端请求query**，根据客户端请求的 query 语句，和数据库中的一些统计信息，在一系列算法的基础上进行分析，得出一个最优的策略，告诉后面的程序如何取得这个 query 语句的结果。**For Example**： `select uid,name from user where gender = 1;`这个`select`查询先根据`where`

语句进行选取，而不是先将表全部查询出来以后再进行`gender`过滤；然后根据`uid`和`name`进行属性投影，而不是将属性全部取出以后再进行过滤。最后将这两个查询条件联接起来生成最终查询结果。

- `Caches & Buffers`：查询缓存。
- `Pluggable Storage Engines`：**存储引擎接口。MySQL区别于其他数据库的最重要的特点就是其插件式的表存储引擎(注意：存储引擎是基于表的，而不是数据库)。**
- `File System`：数据落地到磁盘上，就是文件的存储。

MySQL数据库和其他数据库相比，MySQL有点与众不同，主要体现在存储引擎的架构上，**插件式的存储引擎架构将查询处理和其他的系统任务以及数据的存储提取相分离。**这种架构可以根据业务的需求和实际需求选择合适的存储引擎。

> 逻辑架构分层

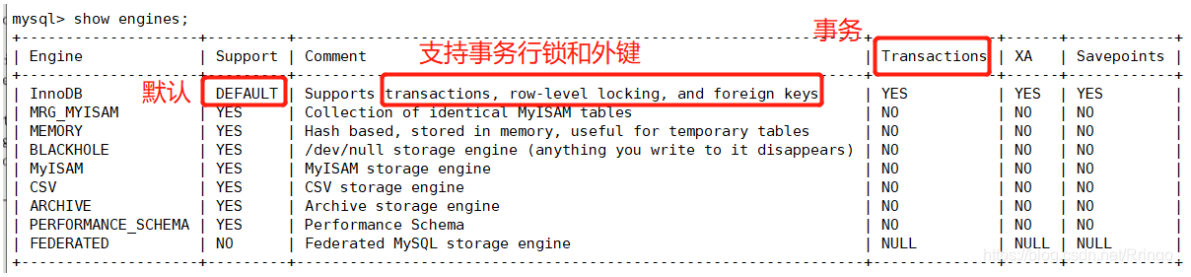


- 连接层：最上层是一些客户端和连接服务，包含本地sock通信和大多数基于客户端/服务端工具实现的类似于`tcp/ip`的通信。主要完成一些类似于连接处理、授权认证、及相关的安全方案。在该层上引入了线程池的概念，为通过认证安全接入的客户端提供线程。同样在该层上可以实现基于`SSL`的安全链接。服务器也会为安全接入的每个客户端验证它所具有的操作权限。
- 服务层：MySQL的核心服务功能层，该层是MySQL的核心，包括查询缓存，解析器，解析树，预处理器，查询优化器。主要进行查询解析、分析、查询缓存、内置函数、存储过程、触发器、视图等，select操作会先检查是否命中查询缓存，命中则直接返回缓存数据，否则解析查询并创建对应的解析树。
- 引擎层：存储引擎层，存储引擎真正的负责了MySQL中数据的存储和提取，服务器通过API与存储引擎进行通信。不同的存储引擎具有的功能不同，这样我们可以根据自己的实际需要进行选取。
- 存储层：数据存储层，主要是将数据存储在运行于裸设备的文件系统之上，并完成与存储引擎的交互。

3.存储引擎

`show engines;`命令查看MySQL5.7支持的存储引擎。

```
mysql> show engines;
```



`show variables like 'default_storage_engine%';`查看当前数据库正在使用的存储引擎。

```
mysql> show variables like 'default_storage_engine%';
+------------------------+--------+
| Variable_name          | Value  |
+------------------------+--------+
| default_storage_engine | InnoDB |
+------------------------+--------+
1 row in set (0.01 sec)
```

> InnoDB和MyISAM对比

对比项	MyISAM	InnoDB
主外键	不支持	支持
事务	不支持	支持
行表锁	表锁，即使操作一条记录也会锁住整张表，**不适合高并发操作**	行锁，操作时只锁某一行，不对其他行有影响，**适合高并发操作**
缓存	只缓存索引，不缓存真实数据	不仅缓存索引还要缓存真实数据，对内存要求较高，而且内存大小对性能有决定性影响
表空间	小	大
关注点	性能	事务
默认安装	Y	Y

4.SQL性能下降的原因

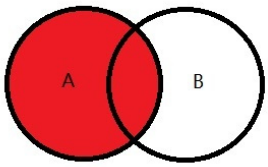
- 查询语句写的差。
- 索引失效：索引建了，但是没有用上。

- 关联 查询太多`join`（设计缺陷或者不得已的需求）。
- 服务器调优以及各个参数的设置（缓冲、线程数等）。

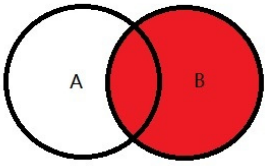
5.SQL执行顺序

```
select              # 5
    ...
from                # 1
    ...
where               # 2
    ....
group by            # 3
    ...
having              # 4
    ...
order by            # 6
    ...
limit               # 7
    [offset]
```

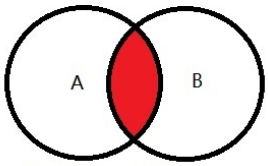
6.七种JOIN理论



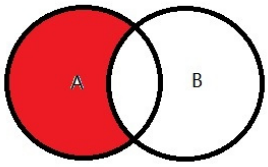
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key;



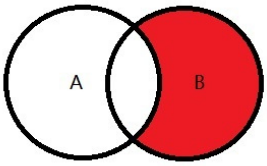
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key;



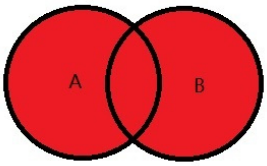
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key;



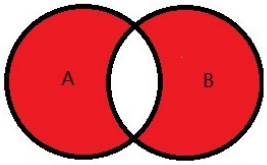
SELECT <select_list>
FROM TableA A LEFT
JOIN TableB B ON
A.Key = B.Key WHERE
B.Key IS NULL;



SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL;



SELECT <select_list>
FROM TableA A FULL
OUTER JOIN TableB B
ON A.Key = B.Key;



SELECT <select_list> FROM
TableA A FULL OUTER JOIN
TableB B ON A.Key = B.Key
WHERE A.Key IS NULL OR
B.Key IS NULL;


```
/* 1 */
SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key;

/* 2 */
SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key;

/* 3 */
SELECT <select_list> FROM TableA A INNER JOIN TableB B ON A.Key = B.Key;

/* 4 */
SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key WHERE
B.Key IS NULL;
```

```
/* 5 */
SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key WHERE
A.Key IS NULL;

/* 6 */
SELECT <select_list> FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key;
/* MySQL不支持FULL OUTER JOIN这种语法 可以改成 1+2 */
SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key
UNION
SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key;

/* 7 */
SELECT <select_list> FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL;
/* MySQL不支持FULL OUTER JOIN这种语法 可以改成 4+5 */
SELECT <select_list> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key WHERE
B.Key IS NULL;
UNION
SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key WHERE
A.Key IS NULL;
```

7.索引

7.1.索引简介

> 索引是什么?

MySQL官方对索引的定义为：**索引（INDEX）是帮助MySQL高效获取数据的数据结果。**

从而可以获得索引的本质：**索引是排好序的快速查找数据结构。**

索引的目的在于提高查询效率，可以类比字典的目录。如果要查`mysql`这个这个单词，我们肯定要先定位到`m`字母，然后从上往下找`y`字母，再找剩下的`sql`。如果没有索引，那么可能需要`a---z`，这样全字典扫描，如果我想找`Java`开头的单词呢？如果我想找`Oracle`开头的单词呢？？？

重点：索引会影响到MySQL查找(WHERE的查询条件)和排序(ORDER BY)两大功能！

除了数据本身之外，数据库还维护着一个满足特定查找算法的数据结构，这些数据结构以某种方式指向数据，这样就可以在这些数据结构的基础上实现高级查找算法，这种数据结构就是索引。

一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储在磁盘上。

```
# Linux下查看磁盘空间命令 df -h
[root@Ringo ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G   16G   23G  41% /
devtmpfs        911M     0  911M   0% /dev
tmpfs           920M     0  920M   0% /dev/shm
tmpfs           920M  480K  920M   1% /run
tmpfs           920M     0  920M   0% /sys/fs/cgroup
overlay          40G   16G   23G  41%
```

我们平时所说的索引，如果没有特别指明，都是指B树（多路搜索树，并不一定是二叉的）结构组织的索引。其中聚集索引，次要索引，覆盖索引，复合索引，前缀索引，唯一索引默认都是使用B+树索引，统称索引。当然，除了B+树这种数据结构的索引之外，还有哈希索引（Hash Index）等。

> 索引的优势和劣势

优势：

- 查找：类似大学图书馆的书目索引，提高数据检索的效率，降低数据库的IO成本。
- 排序：通过索引对数据进行排序，降低数据排序的成本，降低了CPU的消耗。

劣势：

- 实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占用空间的。
- 虽然索引大大提高了查询速度，但是同时会降低表的更新速度，例如对表频繁的进行`INSERT`、`UPDATE`和`DELETE`。因为更新表的时候，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加的索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。
- 索引只是提高效率的一个因素，如果MySQL有大数据量的表，就需要花时间研究建立最优秀的索引。

7.2.MySQL索引分类

索引分类：

- 单值索引：一个索引只包含单个列，一个表可以有多个单列索引。
- 唯一索引：索引列的值必须唯一，但是允许空值。
- 复合索引：一个索引包含多个字段。

建议：一张表建的索引最好不要超过5个！

```
/* 基本语法 */

/* 1、创建索引 [UNIQUE]可以省略*/
/* 如果只写一个字段就是单值索引，写多个字段就是复合索引 */
CREATE [UNIQUE] INDEX indexName ON tabName(columnName(length));

/* 2、删除索引 */
DROP INDEX [indexName] ON tabName;

/* 3、查看索引 */
/* 加上\G就可以以列的形式查看了 不加\G就是以表的形式查看 */
SHOW INDEX FROM tabName \G;
```

使用`ALTER`命令来为数据表添加索引

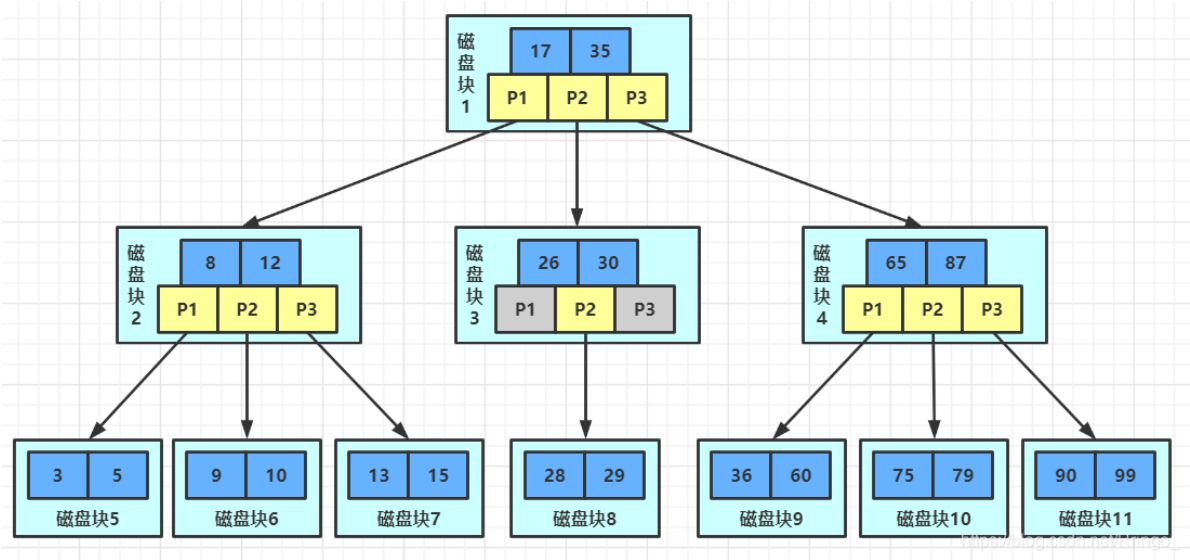
```
/* 1、该语句添加一个主键，这意味着索引值必须是唯一的，并且不能为NULL */
ALTER TABLE tabName ADD PRIMARY KEY(column_list);

/* 2、该语句创建索引的键值必须是唯一的(除了NULL之外，NULL可能会出现多次) */
ALTER TABLE tabName ADD UNIQUE indexName(column_list);

/* 3、该语句创建普通索引，索引值可以出现多次 */
ALTER TABLE tabName ADD INDEX indexName(column_list);

/* 4、该语句指定了索引为FULLTEXT，用于全文检索 */
ALTER TABLE tabName ADD FULLTEXT indexName(column_list);
```

7.3MySQL索引数据结构

索引数据结构：

- `BTree`索引。
- `Hash`索引。
- `Full-text`全文索引。
- `R-Tree`索引。

`BTree`索引检索原理：



7.4.哪些情况需要建索引

- 主键自动建立主键索引（唯一 + 非空）。
- 频繁作为查询条件的字段应该创建索引。
- 查询中与其他表关联的字段，外键关系建立索引。
- 查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度。
- 查询中统计或者分组字段（group by也和索引有关）。

7.5.那些情况不要建索引

- 记录太少的表。
- 经常增删改的表。
- 频繁更新的字段不适合创建索引。
- Where条件里用不到的字段不创建索引。
- 假如一个表有10万行记录，有一个字段A只有true和false两种值，并且每个值的分布概率大约为50%，那么对A字段建索引一般不会提高数据库的查询速度。索引的选择性是指索引列中不同值的数目与表中记录数的比。如果一个表中有2000条记录，表索引列有1980个不同的值，那么这个索引的选择性就是1980/2000=0.99。一个索引的选择性越接近于1，这个索引的效率就越高。

8.性能分析

8.1.EXPLAIN简介

> EXPLAIN是什么?

EXPLAIN：SQL的执行计划，使用EXPLAIN关键字可以模拟优化器执行SQL查询语句，从而知道MySQL是如何处理SQL语句的。

> EXPLAIN怎么使用?

语法：`explain` + `SQL`。

```
mysql> explain select * from pms_category \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: pms_category
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1425
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

> EXPLAIN能干嘛?

可以查看以下信息：

- `id`：表的读取顺序。
- `select_type`：数据读取操作的操作类型。
- `possible_keys`：哪些索引可以使用。
- `key`：哪些索引被实际使用。
- `ref`：表之间的引用。
- `rows`：每张表有多少行被优化器查询。

8.2.EXPLAIN字段

> id

`id`：表的读取和加载顺序。

值有以下三种情况：

- `id`相同，执行顺序由上至下。
- `id`不同，如果是子查询，id的序号会递增，**id值越大优先级越高，越先被执行。**
- `id`相同不同，同时存在。**永远是id大的优先级最高，id相等的时候顺序执行。**

> select_type

`select_type`：数据查询的类型，主要是用于区别，普通查询、联合查询、子查询等的复杂查询。

- `SIMPLE`：简单的`SELECT`查询，查询中不包含子查询或者`UNION`。
- `PRIMARY`：查询中如果包含任何复杂的子部分，最外层查询则被标记为`PRIMARY`。
- `SUBQUERY`：在`SELECT`或者`WHERE`子句中包含了子查询。
- `DERIVED`：在`FROM`子句中包含的子查询被标记为`DERIVED(衍生)`，MySQL会递归执行这些子查询，把结果放在临时表中。
- `UNION`：如果第二个`SELECT`出现在`UNION`之后，则被标记为`UNION`；若`UNION`包含在`FROM`子句的子查询中，外层`SELECT`将被标记为`DERIVED`。
- `UNION RESULT`：从`UNION`表获取结果的`SELECT`。

> type

`type`：访问类型排列。

从最好到最差依次是： `system` > `const` > `eq_ref` > `ref` > `range` > `index` > `ALL`。除了`ALL`没有用到索引，其他级别都用到索引了。

一般来说，得保证查询至少达到`range`级别，最好达到`ref`。

- `system`：表只有一行记录（等于系统表），这是`const`类型的特例，平时不会出现，这个也可以忽略不计。
- `const`：表示通过索引一次就找到了，`const`用于比较`primary key`或者`unique`索引。因为只匹配一行数据，所以很快。如将主键置于`where`列表中，MySQL就能将该查询转化为一个常量。
- `eq_ref`：唯一性索引扫描，读取本表中和关联表表中的每行组合成的一行，查出来只有一条记录。除 了 `system` 和`const` 类型之外, 这是最好的联接类型。
- `ref`：非唯一性索引扫描，返回本表和关联表某个值匹配的所有行，查出来有多条记录。
- `range`：只检索给定范围的行，一般就是在`WHERE`语句中出现了`BETWEEN`、`< >`、`in`等的查询。这种范围扫描索引比全表扫描要好，因为它只需要开始于索引树的某一点，而结束于另一点，不用扫描全部索引。
- `index`：`Full Index Scan`，全索引扫描，`index`和`ALL`的区别为`index`类型只遍历索引树。**也就是说虽然`ALL`和`index`都是读全表，但是`index`是从索引中读的，`ALL`是从磁盘中读取的。**
- `ALL`：`Full Table Scan`，没有用到索引，全表扫描。

> possible_keys 和 key

`possible_keys`：显示可能应用在这张表中的索引，一个或者多个。查询涉及到的字段上若存在索引，则该索引将被列出，**但不一定被查询实际使用。**

`key`：实际使用的索引。如果为`NULL`，则没有使用索引。查询中如果使用了覆盖索引，则该索引仅仅出现在`key`列表中。

> key_len

`key_len`：表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。`key_len`显示的值为索引字段的最大可能长度，并非实际使用长度，即`key_len`是根据表定义计算而得，不是通过表内检索出的。在不损失精度的情况下，长度越短越好。

`key_len`计算规则：https://blog.csdn.net/qq_34930488/article/details/102931490

```
mysql> desc pms_category;
+---------------+------------+------+-----+---------+----------------+
| Field         | Type       | Null | Key | Default | Extra          |
+---------------+------------+------+-----+---------+----------------+
| cat_id        | bigint(20) | NO   | PRI | NULL    | auto_increment |
| name          | char(50)   | YES  |     | NULL    |                |
| parent_cid    | bigint(20) | YES  |     | NULL    |                |
| cat_level     | int(11)    | YES  |     | NULL    |                |
| show_status   | tinyint(4) | YES  |     | NULL    |                |
| sort          | int(11)    | YES  |     | NULL    |                |
| icon          | char(255)  | YES  |     | NULL    |                |
| product_unit  | char(50)   | YES  |     | NULL    |                |
| product_count | int(11)    | YES  |     | NULL    |                |
+---------------+------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)


mysql> explain select cat_id from pms_category where cat_id between 10 and 20
\G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
```

```
        table: pms_category
   partitions: NULL
         type: range
possible_keys: PRIMARY
          key: PRIMARY   # 用到了主键索引，通过查看表结构知道，cat_id是bigint类型，占用8个字节
      key_len: 8         # 这里只用到了cat_id主键索引，所以长度就是8!
          ref: NULL
         rows: 11
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

> ref

`ref`：显示索引的哪一列被使用了，如果可能的话，是一个常数。哪些列或常量被用于查找索引列上的值。

> rows

`rows`：根据表统计信息及索引选用情况，大致估算出找到所需的记录需要读取的行数。

> Extra

`Extra`：包含不适合在其他列中显示但十分重要的额外信息。

- `Using filesort`：说明MySQL会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。**MySQL中无法利用索引完成的排序操作成为"文件内排序"。**

```
# 排序没有使用索引
mysql> explain select name from pms_category where name='Tangs' order by cat_level \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: pms_category
   partitions: NULL
         type: ref
possible_keys: idx_name_parentCid_catLevel
          key: idx_name_parentCid_catLevel
      key_len: 201
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using where; Using index; Using filesort
1 row in set, 1 warning (0.00 sec)

#~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
# 排序使用到了索引

mysql> explain select name from pms_category where name='Tangs' order by parent_cid,cat_level\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: pms_category
   partitions: NULL
```

```
         type: ref
possible_keys: idx_name_parentCid_catLevel
          key: idx_name_parentCid_catLevel
      key_len: 201
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.00 sec)
```

- `Using temporary`：使用了临时表保存中间结果，MySQL在对查询结果排序时使用了临时表。常见于排序`order by`和分组查询`group by`。**临时表对系统性能损耗很大。**
- `Using index`：表示相应的`SELECT`操作中使用了覆盖索引，避免访问了表的数据行，效率不错！如果同时出现`Using where`，表示索引被用来执行索引键值的查找；如果没有同时出现`Using where`，表明索引用来读取数据而非执行查找动作。

```
# 覆盖索引
# 就是select的数据列只用从索引中就能够取得，不必从数据表中读取，换句话说查询列要被所使用的索引覆盖。
# 注意：如果要使用覆盖索引，一定不能写SELECT *，要写出具体的字段。
mysql> explain select cat_id from pms_category \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: pms_category
   partitions: NULL
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 8
          ref: NULL
         rows: 1425
     filtered: 100.00
        Extra: Using index   # select的数据列只用从索引中就能够取得，不必从数据表中读取
1 row in set, 1 warning (0.00 sec)
```

- `Using where`：表明使用了`WHERE`过滤。
- `Using join buffer`：使用了连接缓存。
- `impossible where`：`WHERE`子句的值总是false，不能用来获取任何元组。

```
mysql> explain select name from pms_category where name = 'zs' and name = 'ls'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Impossible WHERE   # 不可能字段同时查到两个名字
1 row in set, 1 warning (0.00 sec)
```

9.索引分析

9.1.单表索引分析

> 数据准备

```
DROP TABLE IF EXISTS `article`;

CREATE TABLE IF NOT EXISTS `article`(
`id` INT(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT '主键',
`author_id` INT(10) UNSIGNED NOT NULL COMMENT '作者id',
`category_id` INT(10) UNSIGNED NOT NULL COMMENT '分类id',
`views` INT(10) UNSIGNED NOT NULL COMMENT '被查看的次数',
`comments` INT(10) UNSIGNED NOT NULL COMMENT '回帖的备注',
`title` VARCHAR(255) NOT NULL COMMENT '标题',
`content` VARCHAR(255) NOT NULL COMMENT '正文内容'
) COMMENT '文章';

INSERT INTO `article`(`author_id`, `category_id`, `views`, `comments`, `title`,
`content`) VALUES(1,1,1,1,'1','1');
INSERT INTO `article`(`author_id`, `category_id`, `views`, `comments`, `title`,
`content`) VALUES(2,2,2,2,'2','2');
INSERT INTO `article`(`author_id`, `category_id`, `views`, `comments`, `title`,
`content`) VALUES(3,3,3,3,'3','3');
INSERT INTO `article`(`author_id`, `category_id`, `views`, `comments`, `title`,
`content`) VALUES(1,1,3,3,'3','3');
INSERT INTO `article`(`author_id`, `category_id`, `views`, `comments`, `title`,
`content`) VALUES(1,1,4,4,'4','4');
```

> 案例：查询`category_id`为1且`comments`大于1的情况下，`views`最多的`article_id`。

1、编写SQL语句并查看SQL执行计划。

```
# 1、sql语句
SELECT id,author_id FROM article WHERE category_id = 1 AND comments > 1 ORDER BY
views DESC LIMIT 1;

# 2、sql执行计划
mysql> EXPLAIN SELECT id,author_id FROM article WHERE category_id = 1 AND
comments > 1 ORDER BY views DESC LIMIT 1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: article
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5
     filtered: 20.00
        Extra: Using where; Using filesort  # 产生了文件内排序，需要优化SQL
1 row in set, 1 warning (0.00 sec)
```

2、创建索引`idx_article_ccv`。

```
CREATE INDEX idx_article_ccv ON article(category_id,comments,views);
```

3、查看当前索引。

```
mysql> SHOW INDEX FROM article;
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| Table   | Non_unique | Key_name       | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
| article |          0 | PRIMARY        |            1 | id          | A         |           4 |     NULL | NULL   |      | BTREE      |
| article |          1 | idx_article_ccv |            1 | category_id | A         |           3 |     NULL | NULL   |      | BTREE      |
| article |          1 | idx_article_ccv |            2 | comments    | A         |           5 |     NULL | NULL   |      | BTREE      |
| article |          1 | idx_article_ccv |            3 | views       | A         |           5 |     NULL | NULL   |      | BTREE      |
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+
4 rows in set (0.00 sec)
```


4、查看现在SQL语句的执行计划。

```
mysql> EXPLAIN SELECT id,author_id FROM article WHERE category_id = 1 AND comments > 1 ORDER BY views DESC LIMIT 1;
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
| id | select_type | table   | partitions | type | possible_keys | key            | key_len | ref  | rows | filtered | Extra                                        |
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
|  1 | SIMPLE      | article | NULL       | range | idx_article_ccv | idx_article_ccv | 8       | NULL |    2 |   100.00 | Using index condition; Using filesort |
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

我们发现，创建符合索引`idx_article_ccv`之后，虽然解决了全表扫描的问题，但是在`order by`排序的时候没有用到索引，MySQL居然还是用的`Using filesort`，为什么？

5、我们试试把SQL修改为`SELECT id,author_id FROM article WHERE category_id = 1 AND comments = 1 ORDER BY views DESC LIMIT 1;`看看SQL的执行计划。

```
mysql> EXPLAIN SELECT id,author_id FROM article WHERE category_id = 1 AND comments = 1 ORDER BY views DESC LIMIT 1;
+----+-------------+---------+------------+------+---------------+----------------+---------+-------------+------+----------+-------------+
| id | select_type | table   | partitions | type | possible_keys | key            | key_len | ref         | rows | filtered | Extra       |
+----+-------------+---------+------------+------+---------------+----------------+---------+-------------+------+----------+-------------+
|  1 | SIMPLE      | article | NULL       | ref  | idx_article_ccv | idx_article_ccv | 8       | const,const |    1 |   100.00 | Using where |
+----+-------------+---------+------------+------+---------------+----------------+---------+-------------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

推论：当`comments > 1`的时候`order by`排序`views`字段索引就用不上，但是当`comments = 1`的时候`order by`排序`views`字段索引就可以用上！！！**所以，范围之后的索引会失效。**

6、我们现在知道**范围之后的索引会失效**，原来的索引`idx_article_ccv`最后一个字段`views`会失效，那么我们如果删除这个索引，创建`idx_article_cv`索引呢？？？？

```
/* 创建索引 idx_article_cv */
CREATE INDEX idx_article_cv ON article(category_id,views);
```

查看当前的索引

```
mysql> SHOW INDEX FROM article;
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table   | Non_unique | Key_name       | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| article |          0 | PRIMARY        |            1 | id          | A         |           4 |     NULL | NULL   |      | BTREE      |         |               |
| article |          1 | idx_article_cv |            1 | category_id | A         |           3 |     NULL | NULL   |      | BTREE      |         |               |
| article |          1 | idx_article_cv |            2 | views       | A         |           5 |     NULL | NULL   |      | BTREE      |         |               |
+---------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
3 rows in set (0.00 sec)
```

7、当前索引是`idx_article_cv`，来看一下SQL执行计划。

```
mysql> EXPLAIN SELECT id,author_id FROM article WHERE category_id = 1 AND comments > 1 ORDER BY views DESC LIMIT 1;
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
| id | select_type | table   | partitions | type | possible_keys | key            | key_len | ref  | rows | filtered | Extra                                        |
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
|  1 | SIMPLE      | article | NULL       | range | idx_article_cv | idx_article_cv | 4       | NULL |    3 |    33.33 | Using index condition; Using where |
+----+-------------+---------+------------+------+---------------+----------------+---------+------+------+----------+----------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

9.2.两表索引分析

> 数据准备

```
DROP TABLE IF EXISTS `class`;
DROP TABLE IF EXISTS `book`;

CREATE TABLE IF NOT EXISTS `class`(
`id` INT(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT '主键',
`card` INT(10) UNSIGNED NOT NULL COMMENT '分类'
) COMMENT '商品类别';

CREATE TABLE IF NOT EXISTS `book`(
`bookid` INT(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT '主键',
`card` INT(10) UNSIGNED NOT NULL COMMENT '分类'
) COMMENT '书籍';
```

> 两表连接查询的SQL执行计划

1、不创建索引的情况下，SQL的执行计划。

```
mysql> EXPLAIN SELECT * FROM book LEFT JOIN class ON book.card = class.card;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                                              |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
|  1 | SIMPLE      | book  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   24 |   100.00 | NULL                                               |
|  1 | SIMPLE      | class | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   22 |   100.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)
```

`book` 和 `class` 两张表都是没有使用索引，全表扫描，那么如果进行优化，索引是创建在 `book` 表还是创建在 `class` 表呢？下面进行大胆的尝试！

2、左表(`book` 表)创建索引。

创建索引 `idx_book_card`

```
/* 在book表创建索引 */
CREATE INDEX idx_book_card ON book(card);
```

在 `book` 表中有 `idx_book_card` 索引的情况下，查看SQL执行计划

```
mysql> EXPLAIN SELECT * FROM book LEFT JOIN class ON book.card = class.card;
+----+-------------+-------+------------+-------+---------------+--------------+---------+------+------+----------+----------------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key          | key_len | ref  | rows | filtered | Extra                                              |
+----+-------------+-------+------------+-------+---------------+--------------+---------+------+------+----------+----------------------------------------------------+
|  1 | SIMPLE      | book  | NULL       | index | NULL          | idx_book_card | 4       | NULL |   24 |   100.00 | Using index                                        |
|  1 | SIMPLE      | class | NULL       | ALL   | NULL          | NULL         | NULL    | NULL |   22 |   100.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+-------+---------------+--------------+---------+------+------+----------+----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)
```

3、删除 `book` 表的索引，右表(`class` 表)创建索引。

创建索引 `idx_class_card`

```
/* 在class表创建索引 */
CREATE INDEX idx_class_card ON class(card);
```

在 `class` 表中有 `idx_class_card` 索引的情况下，查看SQL执行计划

```
mysql> EXPLAIN SELECT * FROM book LEFT JOIN class ON book.card = class.card;
+----+-------------+-------+------------+------+---------------+---------------+---------+-------------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key           | key_len | ref                     | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+---------------+---------+-------------------------+------+----------+-------------+
|  1 | SIMPLE      | book  | NULL       | ALL  | NULL          | NULL          | NULL    | NULL                    |   24 |   100.00 | NULL        |
|  1 | SIMPLE      | class | NULL       | ref  | idx_class_card | idx_class_card | 4       | sql_analysis.book.card |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------------+---------+-------------------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

由此可见，左连接将索引创建在右表上更合适，右连接将索引创建在左表上更合适。

9.3.三张表索引分析

> 数据准备

```
DROP TABLE IF EXISTS `phone`;

CREATE TABLE IF NOT EXISTS `phone`(
`phone_id` INT(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT '主键',
`card` INT(10) UNSIGNED NOT NULL COMMENT '分类'
) COMMENT '手机';
```

> 三表连接查询SQL优化

1、不加任何索引，查看SQL执行计划。

```
mysql> EXPLAIN SELECT * FROM class LEFT JOIN book ON class.card = book.card LEFT JOIN phone ON book.card = phone.card;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                                              |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
|  1 | SIMPLE      | class | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   22 |   100.00 | NULL                                               |
|  1 | SIMPLE      | book  | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   24 |   100.00 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | phone | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   15 |   100.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------------------------------------------+
3 rows in set, 1 warning (0.00 sec)
```

2、根据两表查询优化的经验，左连接需要在右表上添加索引，所以尝试在`book`表和`phone`表上添加索引。

```
/* 在book表创建索引 */
CREATE INDEX idx_book_card ON book(card);

/* 在phone表上创建索引 */
CREATE INDEX idx_phone_card ON phone(card);
```

再次执行SQL的执行计划

```
mysql> EXPLAIN SELECT * FROM class LEFT JOIN book ON class.card = book.card LEFT JOIN phone ON book.card = phone.card;
+----+-------------+-------+------------+------+---------------+---------------+---------+--------------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key           | key_len | ref                      | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+---------------+---------+--------------------------+------+----------+-------------+
|  1 | SIMPLE      | class | NULL       | ALL  | NULL          | NULL          | NULL    | NULL                     |   22 |   100.00 | NULL        |
|  1 | SIMPLE      | book  | NULL       | ref  | idx_book_card | idx_book_card | 4       | sql_analysis.class.card  |    1 |   100.00 | Using index |
|  1 | SIMPLE      | phone | NULL       | ref  | idx_phone_card| idx_phone_card| 4       | sql_analysis.book.card   |    1 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------------+---------+--------------------------+------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)
```

9.4.结论

`JOIN`语句的优化：

- 尽可能减少`JOIN`语句中的`NestedLoop`（嵌套循环）的总次数：**永远都是小的结果集驱动大的结果集。**
- 优先优化`NestedLoop`的内层循环。
- 保证`JOIN`语句中被驱动表上`JOIN`条件字段已经被索引。
- 当无法保证被驱动表的`JOIN`条件字段被索引且内存资源充足的前提下，不要太吝惜`Join Buffer`的设置。

10.索引失效

> 数据准备

```
CREATE TABLE `staffs`(
`id` INT(10) PRIMARY KEY AUTO_INCREMENT,
`name` VARCHAR(24) NOT NULL DEFAULT '' COMMENT '姓名',
`age` INT(10) NOT NULL DEFAULT 0 COMMENT '年龄',
`pos` VARCHAR(20) NOT NULL DEFAULT '' COMMENT '职位',
`add_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '入职时间'
)COMMENT '员工记录表';
```

```
INSERT INTO `staffs`(`name`,`age`,`pos`) VALUES('Ringo', 18, 'manager');
INSERT INTO `staffs`(`name`,`age`,`pos`) VALUES('张三', 20, 'dev');
INSERT INTO `staffs`(`name`,`age`,`pos`) VALUES('李四', 21, 'dev');

/* 创建索引 */
CREATE INDEX idx_staffs_name_age_pos ON `staffs`(`name`,`age`,`pos`);
```

10.1.索引失效的情况

- 全值匹配我最爱。
- 最佳左前缀法则。
- 不在索引列上做任何操作（计算、函数、(自动or手动)类型转换），会导致索引失效而转向全表扫描。
- 索引中范围条件右边的字段会全部失效。
- 尽量使用覆盖索引（只访问索引的查询，索引列和查询列一致），减少`SELECT *`。
- MySQL在使用`!=`或者`<>`的时候无法使用索引会导致全表扫描。
- `is null`、`is not null`也无法使用索引。
- `like`以通配符开头`%abc`索引失效会变成全表扫描。
- 字符串不加单引号索引失效。
- 少用`or`，用它来连接时会索引失效。

10.2.最佳左前缀法则

> 案例

```
/* 用到了idx_staffs_name_age_pos索引中的name字段 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo';

/* 用到了idx_staffs_name_age_pos索引中的name, age字段 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `age` = 18;

/* 用到了idx_staffs_name_age_pos索引中的name，age，pos字段 这是属于全值匹配的情况！！！*/
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `age` = 18 AND `pos` =
'manager';

/* 索引没用上，ALL全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `age` = 18 AND `pos` = 'manager';

/* 索引没用上，ALL全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `pos` = 'manager';

/* 用到了idx_staffs_name_age_pos索引中的name字段，pos字段索引失效 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `pos` = 'manager';
```

> 概念

最佳左前缀法则：如果索引是多字段的复合索引，要遵守最佳左前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的字段。

口诀：带头大哥不能死，中间兄弟不能断。

10.3.索引列上不计算

> 案例

```
# 现在要查询`name` = 'Ringo'的记录下面有两种方式来查询!

# 1、直接使用 字段 = 值的方式来计算
mysql> SELECT * FROM `staffs` WHERE `name` = 'Ringo';
+----+-------+-----+---------+---------------------+
| id | name  | age | pos     | add_time            |
+----+-------+-----+---------+---------------------+
|  1 | Ringo |  18 | manager | 2020-08-03 08:30:39 |
+----+-------+-----+---------+---------------------+
1 row in set (0.00 sec)

# 2、使用MySQL内置的函数
mysql> SELECT * FROM `staffs` WHERE LEFT(`name`, 5) = 'Ringo';
+----+-------+-----+---------+---------------------+
| id | name  | age | pos     | add_time            |
+----+-------+-----+---------+---------------------+
|  1 | Ringo |  18 | manager | 2020-08-03 08:30:39 |
+----+-------+-----+---------+---------------------+
1 row in set (0.00 sec)
```

我们发现以上两条SQL的执行结果都是一样的，但是执行效率有没有差距呢？？？

通过分析两条SQL的执行计划来分析性能。

```
mysql> EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo';
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type | possible_keys         | key                   | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | staffs | NULL       | ref  | idx_staffs_name_age_pos | idx_staffs_name_age_pos | 74      | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM `staffs` WHERE LEFT(`name`, 5) = 'Ringo';
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | staffs | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |   100.00 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

索引没有用上

由此可见，在索引列上进行计算，会使索引失效。

口诀：索引列上不计算。

10.4.范围之后全失效

> 案例

```
/* 用到了idx_staffs_name_age_pos索引中的name，age，pos字段 这是属于全值匹配的情况！！！*/
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `age` = 18 AND `pos` =
'manager';


/* 用到了idx_staffs_name_age_pos索引中的name，age字段，pos字段索引失效 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = '张三' AND `age` > 18 AND `pos` =
'dev';
```

查看上述SQL的执行计划

```
mysql> EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `age` = 18 AND `pos` = 'manager';
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------------------+------+----------+-------+
| id | select_type | table  | partitions | type | possible_keys         | key                   | key_len | ref               | rows | filtered | Extra |
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------------------+------+----------+-------+
|  1 | SIMPLE      | staffs | NULL       | ref  | idx_staffs_name_age_pos | idx_staffs_name_age_pos | 140     | const,const,const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+------+-----------------------+-----------------------+---------+-------------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'zs' AND `age` > 18 AND `pos` = 'dev';
+----+-------------+--------+------------+-------+-----------------------+-----------------------+---------+------+------+----------+-----------------------+
| id | select_type | table  | partitions | type  | possible_keys         | key                   | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+--------+------------+-------+-----------------------+-----------------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | staffs | NULL       | range | idx_staffs_name_age_pos | idx_staffs_name_age_pos | 78      | NULL |    1 |    33.33 | Using index condition |
+----+-------------+--------+------------+-------+-----------------------+-----------------------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

name age pos 三个字段都用到了

pos索引字段失效

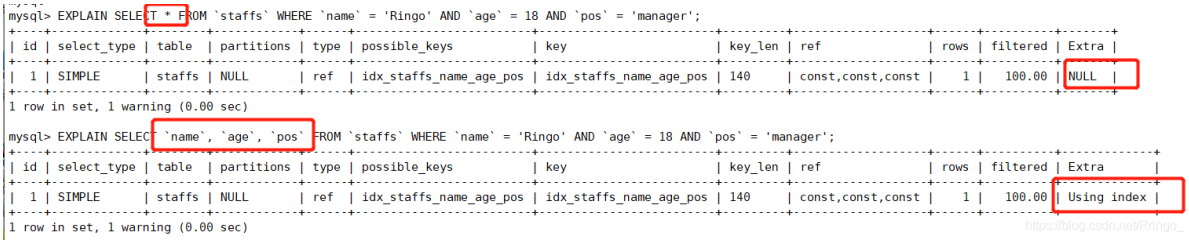
由此可知，查询范围的字段使用到了索引，但是范围之后的索引字段会失效。

口诀：范围之后全失效。

10.5.覆盖索引尽量用

在写SQL的不要使用`SELECT *`，用什么字段就查询什么字段。

```
/* 没有用到覆盖索引 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 'Ringo' AND `age` = 18 AND `pos` =
'manager';

/* 用到了覆盖索引 */
EXPLAIN SELECT `name`, `age`, `pos` FROM `staffs` WHERE `name` = 'Ringo' AND
`age` = 18 AND `pos` = 'manager';
```



口诀：查询一定不用`\*`。

10.6.不等有时会失效

```
/* 会使用到覆盖索引 */
EXPLAIN SELECT `name`, `age`, `pos` FROM `staffs` WHERE `name` != 'Ringo';

/* 索引失效 全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` != 'Ringo';
```

10.7.like百分加右边

```
/* 索引失效 全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` LIKE '%ing%';

/* 索引失效 全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` LIKE '%ing';

/* 使用索引范围查询 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` LIKE 'Rin%';
```

口诀：`like`百分加右边。

如果一定要使用`%like`，而且还要保证索引不失效，那么使用覆盖索引来编写SQL。

```
/* 使用到了覆盖索引 */
EXPLAIN SELECT `id` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `name` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
```

```
EXPLAIN SELECT `age` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `pos` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`, `name` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`, `age` FROM `staffs` WHERE `name` LIKE '%in%';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`,`name`, `age`, `pos` FROM `staffs` WHERE `name` LIKE '%in';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`, `name` FROM `staffs` WHERE `pos` LIKE '%na';

/* 索引失效 全表扫描 */
EXPLAIN SELECT `name`, `age`, `pos`, `add_time` FROM `staffs` WHERE `name` LIKE
'%in';
```

```
mysql> EXPLAIN SELECT `name`, `age`, `pos` FROM `staffs` WHERE `name` LIKE '%in%';
+----+-------------+--------+------------+-------+---------------+-----------------------+---------+------+------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys | key                   | key_len | ref  | rows | filtered | Extra                    |
+----+-------------+--------+------------+-------+---------------+-----------------------+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | staffs | NULL       | index | NULL          | idx_staffs_name_age_pos | 140     | NULL |    3 |    33.33 | Using where; Using index |
+----+-------------+--------+------------+-------+---------------+-----------------------+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

口诀：覆盖索引保两边。

10.8.字符要加单引号

```
/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`, `name` FROM `staffs` WHERE `name` = 'Ringo';

/* 使用到了覆盖索引 */
EXPLAIN SELECT `id`, `name` FROM `staffs` WHERE `name` = 2000;

/* 索引失效 全表扫描 */
EXPLAIN SELECT * FROM `staffs` WHERE `name` = 2000;
```

这里name = 2000在MySQL中会发生强制类型转换，将数字转成字符串。

口诀：字符要加单引号。

10.9.索引相关题目

假设index(a,b,c)

Where语句	索引是否被使用
where a = 3	Y，使用到a
where a = 3 and b = 5	Y，使用到a，b
where a = 3 and b = 5	Y，使用到a，b，c
where b = 3 或者 where b = 3 and c = 4 或者 where c = 4	N，没有用到a字段
where a = 3 and c = 5	使用到a，但是没有用到c，因为b断了
where a = 3 and b > 4 and c = 5	使用到a，b，但是没有用到c，因为c在范围之后
where a = 3 and b like 'kk%' and c = 4	Y，a，b，c都用到
where a = 3 and b like '%kk' and c = 4	只用到a
where a = 3 and b like '%kk%' and c = 4	只用到a
where a = 3 and b like 'k%kk%' and c = 4	Y，a，b，c都用到

10.10.面试题分析

> 数据准备

```
/* 创建表 */
CREATE TABLE `test03`(
`id` INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
`c1` CHAR(10),
`c2` CHAR(10),
`c3` CHAR(10),
`c4` CHAR(10),
`c5` CHAR(10)
);

/* 插入数据 */
INSERT INTO `test03`(`c1`,`c2`,`c3`,`c4`,`c5`) VALUES('a1','a2','a3','a4','a5');
INSERT INTO `test03`(`c1`,`c2`,`c3`,`c4`,`c5`)
VALUES('b1','b22','b3','b4','b5');
INSERT INTO `test03`(`c1`,`c2`,`c3`,`c4`,`c5`) VALUES('c1','c2','c3','c4','c5');
INSERT INTO `test03`(`c1`,`c2`,`c3`,`c4`,`c5`) VALUES('d1','d2','d3','d4','d5');
INSERT INTO `test03`(`c1`,`c2`,`c3`,`c4`,`c5`) VALUES('e1','e2','e3','e4','e5');

/* 创建复合索引 */
CREATE INDEX idx_test03_c1234 ON `test03`(`c1`,`c2`,`c3`,`c4`);
```

> 题目

```
/* 最好索引怎么创建的，就怎么用，按照顺序使用，避免让MySQL再自己去翻译一次 */

/* 1.全值匹配 用到索引c1 c2 c3 c4全字段 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c3` = 'a3'
AND `c4` = 'a4';
```

```
/* 2.用到索引c1 c2 c3 c4全字段 MySQL的查询优化器会优化SQL语句的顺序*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c4` = 'a4'
AND `c3` = 'a3';

/* 3.用到索引c1 c2 c3 c4全字段 MySQL的查询优化器会优化SQL语句的顺序*/
EXPLAIN SELECT * FROM `test03` WHERE `c4` = 'a4' AND `c3` = 'a3' AND `c2` = 'a2'
AND `c1` = 'a1';

/* 4.用到索引c1 c2 c3字段，c4字段失效，范围之后全失效 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c3` > 'a3'
AND `c4` = 'a4';

/* 5.用到索引c1 c2 c3 c4全字段 MySQL的查询优化器会优化SQL语句的顺序*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c4` > 'a4'
AND `c3` = 'a3';

/*
   6.用到了索引c1 c2 c3三个字段，c1和c2两个字段用于查找， c3字段用于排序了但是没有统计到
key_len中，c4字段失效
*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c4` = 'a4'
ORDER BY `c3`;

/* 7.用到了索引c1 c2 c3三个字段，c1和c2两个字段用于查找，c3字段用于排序了但是没有统计到
key_len中*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' ORDER BY `c3`;

/*
   8.用到了索引c1 c2两个字段，c4失效，c1和c2两个字段用于查找，c4字段排序产生了Using
filesort说明排序没有用到c4字段
*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' ORDER BY `c4`;

/* 9.用到了索引c1 c2 c3三个字段，c1用于查找，c2和c3用于排序 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c5` = 'a5' ORDER BY `c2`,
`c3`;

/* 10.用到了c1一个字段，c1用于查找，c3和c2两个字段索引失效，产生了Using filesort */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c5` = 'a5' ORDER BY `c3`,
`c2`;

/* 11.用到了c1 c2 c3三个字段，c1 c2用于查找，c2 c3用于排序 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND  `c2` = 'a2' ORDER BY c2,
c3;

/* 12.用到了c1 c2 c3三个字段，c1 c2用于查找，c2 c3用于排序 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND  `c2` = 'a2' AND `c5` = 'a5'
ORDER BY c2, c3;

/*
   13.用到了c1 c2 c3三个字段，c1 c2用于查找，c2 c3用于排序 没有产生Using filesort
      因为之前c2这个字段已经确定了是'a2'了，这是一个常量，再去ORDER BY c3,c2 这时候c2已经不
用排序了!
      所以没有产生Using filesort 和(10)进行对比学习!
*/
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c2` = 'a2' AND `c5` = 'a5'
ORDER BY c3, c2;
```

```
/* GROUP BY 表面上是叫做分组，但是分组之前必定排序。 */

/* 14.用到c1 c2 c3三个字段，c1用于查找，c2 c3用于排序，c4失效 */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c4` = 'a4' GROUP BY
`c2`,`c3`;

/* 15.用到c1这一个字段，c4失效，c2和c3排序失效产生了Using filesort */
EXPLAIN SELECT * FROM `test03` WHERE `c1` = 'a1' AND `c4` = 'a4' GROUP BY
`c3`,`c2`;
```

`GROUP BY`基本上都需要进行排序，索引优化几乎和`ORDER BY`一致，但是`GROUP BY`会有临时表的产生。

10.11.总结

索引优化的一般性建议：

- 对于单值索引，尽量选择针对当前`query`过滤性更好的索引。
- 在选择复合索引的时候，当前`query`中过滤性最好的字段在索引字段顺序中，位置越靠前越好。
- 在选择复合索引的时候，尽量选择可以能够包含当前`query`中的`where`子句中更多字段的索引。
- 尽可能通过分析统计信息和调整`query`的写法来达到选择合适索引的目的。

口诀：

- 带头大哥不能死。
- 中间兄弟不能断。
- 索引列上不计算。
- 范围之后全失效。
- 覆盖索引尽量用。
- 不等有时会失效。
- like百分加右边。
- 字符要加单引号。
- 一般SQL少用or。

11.分析慢SQL的步骤

分析：

1、观察，至少跑1天，看看生产的慢SQL情况。

2、开启慢查询日志，设置阈值，比如超过5秒钟的就是慢SQL，并将它抓取出来。

3、explain + 慢SQL分析。

4、show Profile。

5、运维经理 OR DBA，进行MySQL数据库服务器的参数调优。

总结（大纲）：

1、慢查询的开启并捕获。

2、explain + 慢SQL分析。

3、show Profile查询SQL在MySQL数据库中的执行细节和生命周期情况。

4、MySQL数据库服务器的参数调优。

12.查询优化

12.1.小表驱动大表

> 优化原则：对于MySQL数据库而言，永远都是小表驱动大表。

```
/**
* 举个例子：可以使用嵌套的for循环来理解小表驱动大表。
* 以下两个循环结果都是一样的，但是对于MySQL来说不一样，
* 第一种可以理解为，和MySQL建立5次连接每次查询1000次。
* 第一种可以理解为，和MySQL建立1000次连接每次查询5次。
*/
for(int i = 1; i <= 5; i ++){
    for(int j = 1; j <= 1000; j++){

    }
}
// ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
for(int i = 1; i <= 1000; i ++){
    for(int j = 1; j <= 5; j++){

    }
}
```

> IN和EXISTS

```
/* 优化原则：小表驱动大表，即小的数据集驱动大的数据集 */

/* IN适合B表比A表数据小的情况*/
SELECT * FROM `A` WHERE `id` IN (SELECT `id` FROM `B`)

/* EXISTS适合B表比A表数据大的情况 */
SELECT * FROM `A` WHERE EXISTS (SELECT 1 FROM `B` WHERE `B`.id = `A`.id);
```

EXISTS：

- 语法：`SELECT....FROM tab WHERE EXISTS(subquery);`该语法可以理解为：
- 该语法可以理解为：将主查询的数据，放到子查询中做条件验证，根据验证结果（`true`或是`false`）来决定主查询的数据结果是否得以保留。

提示：

- `EXISTS(subquery)`子查询只返回`true`或者`false`，因此子查询中的`SELECT *`可以是`SELECT 1 OR SELECT X`，它们并没有区别。
- `EXISTS(subquery)`子查询的实际执行过程可能经过了优化而不是我们理解上的逐条对比，如果担心效率问题，可进行实际检验以确定是否有效率问题。
- `EXISTS(subquery)`子查询往往也可以用条件表达式，其他子查询或者`JOIN`替代，何种最优需要具体问题具体分析。

12.2.ORDER BY优化

> 数据准备

```
CREATE TABLE `talA`(
```

```
`age` INT,
`birth` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO `talA`(`age`) VALUES(18);
INSERT INTO `talA`(`age`) VALUES(19);
INSERT INTO `talA`(`age`) VALUES(20);
INSERT INTO `talA`(`age`) VALUES(21);
INSERT INTO `talA`(`age`) VALUES(22);
INSERT INTO `talA`(`age`) VALUES(23);
INSERT INTO `talA`(`age`) VALUES(24);
INSERT INTO `talA`(`age`) VALUES(25);

/* 创建索引 */
CREATE INDEX idx_talA_age_birth ON `talA`(`age`, `birth`);
```

> 案例

```
/* 1.使用索引进行排序了 不会产生Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `age` > 20 ORDER BY `age`;

/* 2.使用索引进行排序了 不会产生Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `age` > 20 ORDER BY `age`,`birth`;

/* 3.没有使用索引进行排序 产生了Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `age` > 20 ORDER BY `birth`;

/* 4.没有使用索引进行排序 产生了Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `age` > 20 ORDER BY `birth`,`age`;

/* 5.没有使用索引进行排序 产生了Using filesort */
EXPLAIN SELECT * FROM `talA` ORDER BY `birth`;

/* 6.没有使用索引进行排序 产生了Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `birth` > '2020-08-04 07:42:21' ORDER BY
`birth`;

/* 7.使用索引进行排序了 不会产生Using filesort */
EXPLAIN SELECT * FROM `talA` WHERE `birth` > '2020-08-04 07:42:21' ORDER BY
`age`;

/* 8.没有使用索引进行排序 产生了Using filesort */
EXPLAIN SELECT * FROM `talA` ORDER BY `age` ASC, `birth` DESC;
```

`ORDER BY`子句，尽量使用索引排序，避免使用`Using filesort`排序。

MySQL支持两种方式的排序，`FileSort`和`Index`，`Index`的效率高，它指MySQL扫描索引本身完成排序。`FileSort`方式效率较低。

`ORDER BY`满足两情况，会使用`Index`方式排序：

- `ORDER BY`语句使用索引最左前列。
- 使用`WHERE`子句与`ORDER BY`子句条件列组合满足索引最左前列。

结论：尽可能在索引列上完成排序操作，遵照索引建的最佳左前缀原则。

> 如果不在索引列上，File Sort有两种算法：MySQL就要启动双路排序算法和单路排序算法

1、双路排序算法：MySQL4.1之前使用双路排序，字面意思就是两次扫描磁盘，最终得到数据，读取行指针和`ORDER BY`列，对他们进行排序，然后扫描已经排序好的列表，按照列表中的值重新从列表中读取对应的数据输出。**一句话，从磁盘取排序字段，在`buffer`中进行排序，再从磁盘取其他字段。**

取一批数据，要对磁盘进行两次扫描，众所周知，IO是很耗时的，所以在MySQL4.1之后，出现了改进的算法，就是单路排序算法。

2、单路排序算法：从磁盘读取查询需要的所有列，按照`ORDER BY`列在`buffer`对它们进行排序，然后扫描排序后的列表进行输出，它的效率更快一些，避免了第二次读取数据。并且把随机IO变成了顺序IO，但是它会使用更多的空间，因为它把每一行都保存在内存中了。

由于单路排序算法是后出的，总体而言效率好过双路排序算法。

但是单路排序算法有问题：如果`SortBuffer`缓冲区太小，导致从磁盘中读取所有的列不能完全保存在`SortBuffer`缓冲区中，这时候单路复用算法就会出现问题，反而性能不如双路复用算法。

单路复用算法的优化策略：

- 增大`sort_buffer_size`参数的设置。
- 增大`max_length_for_sort_data`参数的设置。

提高ORDER BY排序的速度：

- `ORDER BY`时使用`SELECT *`是大忌，查什么字段就写什么字段，这点非常重要。在这里的影响是：
  - 当查询的字段大小总和小于`max_length_for_sort_data`而且排序字段不是`TEXT|BLOB`类型时，会使用单路排序算法，否则使用多路排序算法。
  - 两种排序算法的数据都有可能超出`sort_buffer`缓冲区的容量，超出之后，会创建`tmp`临时文件进行合并排序，导致多次IO，但是单路排序算法的风险会更大一些，所以要增大`sort_buffer_size`参数的设置。
- 尝试提高`sort_buffer_size`：不管使用哪种算法，提高这个参数都会提高效率，当然，要根据系统的能力去提高，因为这个参数是针对每个进程的。
- 尝试提高`max_length_for_sort_data`：提高这个参数，会增加用单路排序算法的概率。但是如果设置的太高，数据总容量`sort_buffer_size`的概率就增大，明显症状是高的磁盘IO活动和低的处理器使用率。

12.3.GORUP BY优化

- `GROUP BY`实质是先排序后进行分组，遵照索引建的最佳左前缀。
- 当无法使用索引列时，会使用`Using filesort`进行排序，增大`max_length_for_sort_data`参数的设置和增大`sort_buffer_size`参数的设置，会提高性能。
- `WHERE`执行顺序高于`HAVING`，能写在`WHERE`限定条件里的就不要写在`HAVING`中了。

12.4.总结

为排序使用索引

- MySQL两种排序方式：`Using filesort`和`Index`扫描有序索引排序。
- MySQL能为排序与查询使用相同的索引，创建的索引既可以用于排序也可以用于查询。

```
/* 创建a b c三个字段的索引 */
idx_table_a_b_c(a, b, c)

/* 1.ORDER BY 能使用索引最左前缀 */
ORDER BY a;
ORDER BY a, b;
```

```
ORDER BY a, b, c;
ORDER BY a DESC, b DESC, c DESC;

/* 2.如果WHERE子句中使用索引的最左前缀定义为常量，则ORDER BY能使用索引 */
WHERE a = 'Ringo' ORDER BY b, c;
WHERE a = 'Ringo' AND b = 'Tangs' ORDER BY c;
WHERE a = 'Ringo' AND b > 2000 ORDER BY b, c;

/* 3.不能使用索引进行排序 */
ORDER BY a ASC, b DESC, c DESC;  /* 排序不一致 */
WHERE g = const ORDER BY b, c;   /* 丢失a字段索引 */
WHERE a = const ORDER BY c;      /* 丢失b字段索引 */
WHERE a = const ORDER BY a, d;   /* d字段不是索引的一部分 */
WHERE a IN (...) ORDER BY b, c;  /* 对于排序来说，多个相等条件(a=1 or a=2)也是范围查询
*/
```

13.慢查询日志

13.1.基本介绍

> 慢查询日志是什么?

- MySQL的慢查询日志是MySQL提供的一种日志记录，它用来记录在MySQL中响应时间超过阈值的语句，具体指运行时间超过`long_query_time`值的SQL，则会被记录到慢查询日志中。
- `long_query_time`的默认值为10，意思是运行10秒以上的语句。
- 由慢查询日志来查看哪些SQL超出了我们的最大忍耐时间值，比如一条SQL执行超过5秒钟，我们就算慢SQL，希望能收集超过5秒钟的SQL，结合之前`explain`进行全面分析。

> 特别说明

默认情况下，MySQL数据库没有开启慢查询日志，需要我们手动来设置这个参数。

当然，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件。

> 查看慢查询日志是否开以及如何开启

- 查看慢查询日志是否开启：`SHOW VARIABLES LIKE '%slow_query_log%';`。
- 开启慢查询日志：`SET GLOBAL slow_query_log = 1;`。**使用该方法开启MySQL的慢查询日志只对当前数据库生效，如果MySQL重启后会失效。**

```
# 1、查看慢查询日志是否开启
mysql> SHOW VARIABLES LIKE '%slow_query_log%';
+---------------------+---------------------------------------+
| Variable_name       | Value                                 |
+---------------------+---------------------------------------+
| slow_query_log      | OFF                                   |
| slow_query_log_file | /var/lib/mysql/1dcb5644392c-slow.log |
+---------------------+---------------------------------------+
2 rows in set (0.01 sec)

# 2、开启慢查询日志
mysql> SET GLOBAL slow_query_log = 1;
Query OK, 0 rows affected (0.00 sec)
```

如果要使慢查询日志永久开启，需要修改`my.cnf`文件，在`[mysqld]`下增加修改参数。

```
# my.cnf
[mysqld]
# 1.这个是开启慢查询。注意ON需要大写
slow_query_log=ON

# 2.这个是存储慢查询的日志文件。这个文件不存在的话，需要自己创建
slow_query_log_file=/var/lib/mysql/slow.log
```

> 开启了慢查询日志后，什么样的SQL才会被记录到慢查询日志里面呢?

这个是由参数`long_query_time`控制的，默认情况下`long_query_time`的值为10秒。

MySQL中查看`long_query_time`的时间：`SHOW VARIABLES LIKE 'long_query_time%';`。

```
# 查看long_query_time 默认是10秒
# 只有SQL的执行时间>10才会被记录
mysql> SHOW VARIABLES LIKE 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

修改`long_query_time`的时间，需要在`my.cnf`修改配置文件

```
[mysqld]
# 这个是设置慢查询的时间，我设置的为1秒
long_query_time=1
```

查新慢查询日志的总记录条数：`SHOW GLOBAL STATUS LIKE '%Slow_queries%';`。

```
mysql> SHOW GLOBAL STATUS LIKE '%Slow_queries%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Slow_queries  | 3     |
+---------------+-------+
1 row in set (0.00 sec)
```

13.2.日志分析工具

日志分析工具`mysqldumpslow`：在生产环境中，如果要手工分析日志，查找、分析SQL，显然是个体力活，MySQL提供了日志分析工具`mysqldumpslow`。

```
# 1、mysqldumpslow --help 来查看mysqldumpslow的帮助信息
root@1dcb5644392c:/usr/bin# mysqldumpslow --help
Usage: mysqldumpslow [ OPTS... ] [ LOGS... ]

Parse and summarize the MySQL slow query log. Options are

  --verbose    verbose
  --debug      debug
  --help       write this text to standard output
```

```
  -v           verbose
  -d           debug
  -s ORDER     what to sort by (al, at, ar, c, l, r, t), 'at' is default  # 按照何种方式排序
                al: average lock time # 平均锁定时间
                ar: average rows sent # 平均返回记录数
                at: average query time # 平均查询时间
                 c: count  # 访问次数
                 l: lock time  # 锁定时间
                 r: rows sent  # 返回记录
                 t: query time  # 查询时间
  -r           reverse the sort order (largest last instead of first)
  -t NUM       just show the top n queries  # 返回前面多少条记录
  -a           don't abstract all numbers to N and strings to 'S'
  -n NUM       abstract numbers with at least n digits within names
  -g PATTERN   grep: only consider stmts that include this string
  -h HOSTNAME  hostname of db server for *-slow.log filename (can be wildcard),
               default is '*', i.e. match all
  -i NAME      name of server instance (if using mysql.server startup script)
  -l           don't subtract lock time from total time

# 2、 案例
# 2.1、得到返回记录集最多的10个SQL
mysqldumpslow -s r -t 10 /var/lib/mysql/slow.log

# 2.2、得到访问次数最多的10个SQL
mysqldumpslow -s c -t 10 /var/lib/mysql/slow.log

# 2.3、得到按照时间排序的前10条里面含有左连接的查询语句
mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/slow.log

# 2.4、另外建议使用这些命令时结合|和more使用，否则出现爆屏的情况
mysqldumpslow -s r -t 10 /var/lib/mysql/slow.log | more
```

14.批量插入数据脚本

14.1.环境准备

> 1、建表SQL。

```
/* 1.dept表 */
CREATE TABLE `dept` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键',
  `deptno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '部门id',
  `dname` varchar(20) NOT NULL DEFAULT '' COMMENT '部门名字',
  `loc` varchar(13) NOT NULL DEFAULT '' COMMENT '部门地址',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='部门表'

/* 2.emp表 */
CREATE TABLE `emp` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '主键',
  `empno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '员工编号',
  `ename` varchar(20) NOT NULL DEFAULT '' COMMENT '员工名字',
  `job` varchar(9) NOT NULL DEFAULT '' COMMENT '职位',
```

```
  `mgr` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '上级编号',
  `hiredata` date NOT NULL COMMENT '入职时间',
  `sal` decimal(7,2) NOT NULL COMMENT '薪水',
  `comm` decimal(7,2) NOT NULL COMMENT '分红',
  `deptno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '部门id',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='员工表'
```

> 2、由于开启过慢查询日志，开启了`bin-log`，我们就必须为`function`指定一个参数，否则使用函数会报错。

```
# 在mysql中设置
# log_bin_trust_function_creators 默认是关闭的 需要手动开启
mysql> SHOW VARIABLES LIKE 'log_bin_trust_function_creators';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| log_bin_trust_function_creators | OFF   |
+---------------------------------+-------+
1 row in set (0.00 sec)

mysql> SET GLOBAL log_bin_trust_function_creators=1;
Query OK, 0 rows affected (0.00 sec)
```

上述修改方式MySQL重启后会失败，在`my.cnf`配置文件下修改永久有效。

```
[mysqld]
log_bin_trust_function_creators=ON
```

14.2.创建函数

```
# 1、函数：随机产生字符串
DELIMITER $$
CREATE FUNCTION rand_string(n INT) RETURNS VARCHAR(255)
BEGIN
    DECLARE chars_str VARCHAR(100) DEFAULT
'abcdefghijklmnopqrstuvwsyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    DECLARE return_str VARCHAR(255) DEFAULT '';
    DECLARE i INT DEFAULT 0;
    WHILE i < n DO
    SET return_str =
CONCAT(return_str,SUBSTRING(chars_str,FLOOR(1+RAND()*52),1));
    SET i = i + 1;
    END WHILE;
    RETURN return_str;
END $$

# 2、函数：随机产生部门编号
DELIMITER $$
CREATE FUNCTION rand_num() RETURNS INT(5)
BEGIN
    DECLARE i INT DEFAULT 0;
    SET i = FLOOR(100 + RAND() * 10);
    RETURN i;
END $$
```

14.3.创建存储过程

```
# 1、函数：向dept表批量插入
DELIMITER $$
CREATE PROCEDURE insert_dept(IN START INT(10),IN max_num INT(10))
BEGIN
DECLARE i INT DEFAULT 0;
    SET autocommit = 0;
    REPEAT
    SET i = i + 1;
    INSERT INTO dept(deptno,dname,loc) VALUES((START +
i),rand_string(10),rand_string(8));
    UNTIL i = max_num
    END REPEAT;
    COMMIT;
END $$

# 2、函数：向emp表批量插入
DELIMITER $$
CREATE PROCEDURE insert_emp(IN START INT(10),IN max_num INT(10))
BEGIN
DECLARE i INT DEFAULT 0;
    SET autocommit = 0;
    REPEAT
    SET i = i + 1;
    INSERT INTO emp(empno,ename,job,mgr,hiredata,sal,comm,deptno) VALUES((START +
i),rand_string(6),'SALESMAN',0001,CURDATE(),2000,400,rand_num());
    UNTIL i = max_num
    END REPEAT;
    COMMIT;
END $$
```

14.4.调用存储过程

```
# 1、调用存储过程向dept表插入10个部门。
DELIMITER ;
CALL insert_dept(100,10);

# 2、调用存储过程向emp表插入50万条数据。
DELIMITER ;
CALL insert_emp(100001,500000);
```

15.Show Profile

> Show Profile是什么?

`Show Profile`：MySQL提供可以用来分析当前会话中语句执行的资源消耗情况。可以用于SQL的调优的测量。**默认情况下，参数处于关闭状态，并保存最近15次的运行结果。**

> 分析步骤

1、是否支持，看看当前的MySQL版本是否支持。

```
# 查看Show Profile功能是否开启
mysql> SHOW VARIABLES LIKE 'profiling';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| profiling     | OFF   |
+---------------+-------+
1 row in set (0.00 sec)
```

2、开启`Show Profile`功能，默认是关闭的，使用前需要开启。

```
# 开启Show Profile功能
mysql> SET profiling=ON;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

3、运行SQL

```
SELECT * FROM `emp` GROUP BY `id`%10 LIMIT 150000;

SELECT * FROM `emp` GROUP BY `id`%20 ORDER BY 5;
```

4、查看结果，执行`SHOW PROFILES;`

`Duration`：持续时间。

```
mysql> SHOW PROFILES;
+----------+------------+---------------------------------------------------+
| Query_ID | Duration   | Query                                             |
+----------+------------+---------------------------------------------------+
|        1 | 0.00156100 | SHOW VARIABLES LIKE 'profiling'                   |
|        2 | 0.56296725 | SELECT * FROM `emp` GROUP BY `id`%10 LIMIT 150000 |
|        3 | 0.52105825 | SELECT * FROM `emp` GROUP BY `id`%10 LIMIT 150000 |
|        4 | 0.51279775 | SELECT * FROM `emp` GROUP BY `id`%20 ORDER BY 5   |
+----------+------------+---------------------------------------------------+
4 rows in set, 1 warning (0.00 sec)
```

5、诊断SQL，`SHOW PROFILE cpu,block io FOR QUERY Query_ID;`

```
# 这里的3是第四步中的Query_ID。
# 可以在SHOW PROFILE中看到一条SQL中完整的生命周期。
mysql> SHOW PROFILE cpu,block io FOR QUERY 3;
+----------------------+----------+----------+------------+--------------+-------
--------+
| Status               | Duration | CPU_user | CPU_system | Block_ops_in |
Block_ops_out |
+----------------------+----------+----------+------------+--------------+-------
--------+
| starting             | 0.000097 | 0.000090 |   0.000002 |            0 |
       0 |
| checking permissions | 0.000010 | 0.000009 |   0.000000 |            0 |
       0 |
| Opening tables       | 0.000039 | 0.000058 |   0.000000 |            0 |
       0 |
| init                 | 0.000046 | 0.000046 |   0.000000 |            0 |
       0 |
```

```
| System lock          | 0.000011 | 0.000000 |   0.000000 |            0 |
      0 |
| optimizing           | 0.000005 | 0.000000 |   0.000000 |            0 |
      0 |
| statistics           | 0.000023 | 0.000037 |   0.000000 |            0 |
      0 |
| preparing            | 0.000014 | 0.000000 |   0.000000 |            0 |
      0 |
| Creating tmp table   | 0.000041 | 0.000053 |   0.000000 |            0 |
      0 |
| Sorting result       | 0.000005 | 0.000000 |   0.000000 |            0 |
      0 |
| executing            | 0.000003 | 0.000000 |   0.000000 |            0 |
      0 |
| Sending data         | 0.520620 | 0.516267 |   0.000000 |            0 |
      0 |
| Creating sort index  | 0.000060 | 0.000051 |   0.000000 |            0 |
      0 |
| end                  | 0.000006 | 0.000000 |   0.000000 |            0 |
      0 |
| query end            | 0.000011 | 0.000000 |   0.000000 |            0 |
      0 |
| removing tmp table   | 0.000006 | 0.000000 |   0.000000 |            0 |
      0 |
| query end            | 0.000004 | 0.000000 |   0.000000 |            0 |
      0 |
| closing tables       | 0.000009 | 0.000000 |   0.000000 |            0 |
      0 |
| freeing items        | 0.000032 | 0.000064 |   0.000000 |            0 |
      0 |
| cleaning up          | 0.000019 | 0.000000 |   0.000000 |            0 |
      0 |
+----------------------+----------+----------+------------+--------------+-------
--------+
20 rows in set, 1 warning (0.00 sec)
```

`Show Profile`查询参数备注：

- `ALL`：显示所有的开销信息。
- `BLOCK IO`：显示块IO相关开销（通用）。
- `CONTEXT SWITCHES`：上下文切换相关开销。
- `CPU`：显示CPU相关开销信息（通用）。
- `IPC`：显示发送和接收相关开销信息。
- `MEMORY`：显示内存相关开销信息。
- `PAGE FAULTS`：显示页面错误相关开销信息。
- `SOURCE`：显示和Source_function。
- `SWAPS`：显示交换次数相关开销的信息。

6、`Show Profile`查询列表，日常开发需要注意的结论：

- `converting HEAP to MyISAM`：查询结果太大，内存都不够用了，往磁盘上搬了。
- `Creating tmp table`：创建临时表（拷贝数据到临时表，用完再删除），非常耗费数据库性能。
- `Copying to tmp table on disk`：把内存中的临时表复制到磁盘，危险！！！
- `locked`：死锁。

mysql 全局查询日志

```
set global general_log = 1;
mysql> set global log_output = 'TABLE';
select * from mysql.general_log;


mysql> show variables like  '%general_log%';
+------------------+-----------------------------------+
| Variable_name    | Value                             |
+------------------+-----------------------------------+
| general_log      | OFF                               |
| general_log_file | /var/lib/mysql/VM-0-16-centos.log |
+------------------+-----------------------------------+
2 rows in set (0.00 sec)


mysql> set global general_log = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like  '%general_log%';
+------------------+-----------------------------------+
| Variable_name    | Value                             |
+------------------+-----------------------------------+
| general_log      | ON                                |
| general_log_file | /var/lib/mysql/VM-0-16-centos.log |
+------------------+-----------------------------------+
2 rows in set (0.00 sec)

mysql> set global log_output = 'TABLE';
Query OK, 0 rows affected (0.00 sec)


mysql> select * from mysql.general_log;
+----------------------------+---------------------------+-----------+-----------
+--------------+-----------------------------------+
| event_time                 | user_host                 | thread_id | server_id
| command_type | argument                          |
+----------------------------+---------------------------+-----------+-----------
+--------------+-----------------------------------+
| 2021-02-21 20:38:44.517918 | root[root] @ localhost [] |        46 |         0
| Query        | select * from mysql.general_log |
| 2021-02-21 20:38:53.428638 | root[root] @ localhost [] |        46 |         0
| Query        | select * from emp limit 100     |
| 2021-02-21 20:38:56.427454 | root[root] @ localhost [] |        46 |         0
| Query        | select * from mysql.general_log |
+----------------------------+---------------------------+-----------+-----------
+--------------+-----------------------------------+
3 rows in set (0.00 sec)
```

16.表锁(偏读)

表锁特点:

- 表锁偏向`MyISAM`存储引擎，开销小，加锁快，无死锁，锁定粒度大，发生锁冲突的概率最高，并发度最低。

16.1.环境准备

```
# 1、创建表
CREATE TABLE `mylock`(
`id` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
`name` VARCHAR(20)
)ENGINE=MYISAM DEFAULT CHARSET=utf8 COMMENT='测试表锁';

# 2、插入数据
INSERT INTO `mylock`(`name`) VALUES('ZhangSan');
INSERT INTO `mylock`(`name`) VALUES('LiSi');
INSERT INTO `mylock`(`name`) VALUES('WangWu');
INSERT INTO `mylock`(`name`) VALUES('ZhaoLiu');
```

16.2.锁表的命令

> 1、查看数据库表锁的命令。

```
# 查看数据库表锁的命令
SHOW OPEN TABLES;
```

> 2、给`mylock`表上读锁，给`book`表上写锁。

```
# 给mylock表上读锁，给book表上写锁
LOCK TABLE `mylock` READ, `book` WRITE;

# 查看当前表的状态
mysql> SHOW OPEN TABLES;
+--------------------+-------------------------------------------------------+----
----+-------------+
| Database           | Table                                                 |
In_use | Name_locked |
+--------------------+-------------------------------------------------------+----
----+-------------+
| sql_analysis       | book                                                  |
 1 |           0 |
| sql_analysis       | mylock                                                |
 1 |           0 |
+--------------------+-------------------------------------------------------+----
----+-------------+
```

> 3、释放表锁。

```
# 释放给表添加的锁
UNLOCK TABLES;

# 查看当前表的状态
mysql> SHOW OPEN TABLES;
+--------------------+-------------------------------------------------------------+----
----+-------------+
| Database           | Table                                                       |
In_use | Name_locked |
+--------------------+-------------------------------------------------------------+----
----+-------------+
| sql_analysis       | book                                                        |
 0 |           0 |
| sql_analysis       | mylock                                                      |
 0 |           0 |
+--------------------+-------------------------------------------------------------+----
----+-------------+
```

16.3.读锁案例

> 1、打开两个会话，`SESSION1`为`mylock`表添加读锁。

```
# 为mylock表添加读锁
LOCK TABLE `mylock` READ;
```

> 2、打开两个会话，`SESSION1`是否可以读自己锁的表？是否可以修改自己锁的表？是否可以读其他的表？那么`SESSION2`呢？

```
# SESSION1

# 问题1：SESSION1为mylock表加了读锁，可以读mylock表！
mysql> SELECT * FROM `mylock`;
+----+----------+
| id | name     |
+----+----------+
|  1 | ZhangSan |
|  2 | LiSi     |
|  3 | WangWu   |
|  4 | ZhaoLiu  |
+----+----------+
4 rows in set (0.00 sec)

# 问题2：SESSION1为mylock表加了读锁，不可以修改mylock表！
mysql> UPDATE `mylock` SET `name` = 'abc' WHERE `id` = 1;
ERROR 1099 (HY000): Table 'mylock' was locked with a READ lock and can't be
updated

# 问题3：SESSION1为mylock表加了读锁，不可以读其他的表！
mysql> SELECT * FROM `book`;
ERROR 1100 (HY000): Table 'book' was not locked with LOCK TABLES


# SESSION2

# 问题1：SESSION1为mylock表加了读锁，SESSION2可以读mylock表！
```

```
mysql> SELECT * FROM `mylock`;
+----+----------+
| id | name     |
+----+----------+
|  1 | ZhangSan |
|  2 | LiSi     |
|  3 | WangWu   |
|  4 | ZhaoLiu  |
+----+----------+
4 rows in set (0.00 sec)

# 问题2：SESSION1为mylock表加了读锁，SESSION2修改mylock表会被阻塞，需要等待SESSION1释放mylock表!
mysql> UPDATE `mylock` SET `name` = 'abc' WHERE `id` = 1;
^C^C -- query aborted
ERROR 1317 (70100): Query execution was interrupted

# 问题3：SESSION1为mylock表加了读锁，SESSION2可以读其他表!
mysql> SELECT * FROM `book`;
+--------+------+
| bookid | card |
+--------+------+
|      1 |    1 |
|      7 |    4 |
|      8 |    4 |
|      9 |    5 |
|      5 |    6 |
|     17 |    6 |
|     15 |    8 |
+--------+------+
24 rows in set (0.00 sec)
```

16.4.写锁案例

> 1、打开两个会话，`SESSION1`为`mylock`表添加写锁。

```
# 为mylock表添加写锁
LOCK TABLE `mylock` WRITE;
```

> 2、打开两个会话，`SESSION1`是否可以读自己锁的表？是否可以修改自己锁的表？是否可以读其他的表？那么`SESSION2`呢？

```
# SESSION1

# 问题1：SESSION1为mylock表加了写锁，可以读mylock的表!
mysql> SELECT * FROM `mylock`;
+----+----------+
| id | name     |
+----+----------+
|  1 | ZhangSan |
|  2 | LiSi     |
|  3 | WangWu   |
|  4 | ZhaoLiu  |
+----+----------+
4 rows in set (0.00 sec)
```

```
# 问题2：SESSION1为mylock表加了写锁，可以修改mylock表!
mysql> UPDATE `mylock` SET `name` = 'abc' WHERE `id` = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

# 问题3：SESSION1为mylock表加了写锁，不能读其他表!
mysql> SELECT * FROM `book`;
ERROR 1100 (HY000): Table 'book' was not locked with LOCK TABLES

# SESSION2

# 问题1：SESSION1为mylock表加了写锁，SESSION2读mylock表会阻塞，等待SESSION1释放!
mysql> SELECT * FROM `mylock`;
^C^C -- query aborted
ERROR 1317 (70100): Query execution was interrupted

# 问题2：SESSION1为mylock表加了写锁，SESSION2读mylock表会阻塞，等待SESSION1释放!
mysql> UPDATE `mylock` SET `name` = 'abc' WHERE `id` = 1;
^C^C -- query aborted
ERROR 1317 (70100): Query execution was interrupted

# 问题3：SESSION1为mylock表加了写锁，SESSION2可以读其他表!
mysql> SELECT * FROM `book`;
+--------+------+
| bookid | card |
+--------+------+
|      1 |    1 |
|      7 |    4 |
|      8 |    4 |
|      9 |    5 |
|      5 |    6 |
|     17 |    6 |
|     15 |    8 |
+--------+------+
24 rows in set (0.00 sec)
```

16.5.案例结论

`MyISAM`引擎在执行查询语句`SELECT`之前，会自动给涉及到的所有表加读锁，在执行增删改之前，会自动给涉及的表加写锁。

MySQL的表级锁有两种模式：

- 表共享读锁（Table Read Lock）。
- 表独占写锁（Table Write Lock）。

对`MyISAM`表进行操作，会有以下情况：

- 对`MyISAM`表的读操作（加读锁），不会阻塞其他线程对同一表的读操作，但是会阻塞其他线程对同一表的写操作。只有当读锁释放之后，才会执行其他线程的写操作。
- 对`MyISAM`表的写操作（加写锁），会阻塞其他线程对同一表的读和写操作，只有当写锁释放之后，才会执行其他线程的读写操作。

16.6.表锁分析

```
mysql> SHOW STATUS LIKE 'table%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Table_locks_immediate      | 173   |
| Table_locks_waited         | 0     |
| Table_open_cache_hits      | 5     |
| Table_open_cache_misses    | 8     |
| Table_open_cache_overflows | 0     |
+----------------------------+-------+
5 rows in set (0.00 sec)
```

可以通过`Table_locks_immediate`和`Table_locks_waited`状态变量来分析系统上的表锁定。具体说明如下：

`Table_locks_immediate`：产生表级锁定的次数，表示可以立即获取锁的查询次数，每立即获取锁值加1。

`Table_locks_waited`：出现表级锁定争用而发生等待的次数（不能立即获取锁的次数，每等待一次锁值加1），此值高则说明存在较严重的表级锁争用情况。

此外，`MyISAM`的读写锁调度是写优先，这也是`MyISAM`不适合作为主表的引擎。因为写锁后，其他线程不能进行任何操作，大量的写操作会使查询很难得到锁，从而造成永远阻塞。

17.行锁(偏写)

行锁特点：

- 偏向`InnoDB`存储引擎，开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度最高。

`InnoDB`存储引擎和`MyISAM`存储引擎最大不同有两点：一是支持事务，二是采用行锁。

事务的ACID：

- `Atomicity [ˌætəˈmɪsəti]`。
- `Consistency [kənˈsɪstənsi]`。
- `Isolation [ˌaɪsəˈleɪʃn]`。
- `Durability [ˌdjʊərəˈbɪlɪti]`。

17.1.环境准备

```
# 建表语句
CREATE TABLE `test_innodb_lock`(
`a` INT,
`b` VARCHAR(16)
)ENGINE=INNODB DEFAULT CHARSET=utf8 COMMENT='测试行锁';

# 插入数据
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(1, 'b2');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(2, '3');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(3, '4000');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(4, '5000');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(5, '6000');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(6, '7000');
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(7, '8000');
```

```
INSERT INTO `test_innodb_lock`(`a`, `b`) VALUES(8, '9000');

# 创建索引
CREATE INDEX idx_test_a ON `test_innodb_lock`(a);
CREATE INDEX idx_test_b ON `test_innodb_lock`(b);
```

17.2.行锁案例

> 1、开启手动提交

打开`SESSION1`和`SESSION2`两个会话，都开启手动提交。

```
# 开启MySQL数据库的手动提交
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
```

> 2、读几知所写

```
# SESSION1

# SESSION1对test_innodb_lock表做写操作，但是没有commit。
# 执行修改SQL之后，查询一下test_innodb_lock表，发现数据被修改了。
mysql> UPDATE `test_innodb_lock` SET `b` = '88' WHERE `a` = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM `test_innodb_lock`;
+------+------+
| a    | b    |
+------+------+
|    1 | 88   |
|    2 | 3    |
|    3 | 4000 |
|    4 | 5000 |
|    5 | 6000 |
|    6 | 7000 |
|    7 | 8000 |
|    8 | 9000 |
+------+------+
8 rows in set (0.00 sec)

# SESSION2

# SESSION2这时候来查询test_innodb_lock表。
# 发现SESSION2是读不到SESSION1未提交的数据的。
mysql> SELECT * FROM `test_innodb_lock`;
+------+------+
| a    | b    |
+------+------+
|    1 | b2   |
|    2 | 3    |
|    3 | 4000 |
|    4 | 5000 |
|    5 | 6000 |
|    6 | 7000 |
|    7 | 8000 |
```

```
|    8 | 9000 |
+------+------+
8 rows in set (0.00 se
```

> 3、行锁两个SESSION同时对一条记录进行写操作

```
# SESSION1 对test_innodb_lock表的`a`=1这一行进行写操作，但是没有commit
mysql> UPDATE `test_innodb_lock` SET `b` = '99' WHERE `a` = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

# SESSION2 也对test_innodb_lock表的`a`=1这一行进行写操作，但是发现阻塞了！！！
# 等SESSION1执行commit语句之后，SESSION2的SQL就会执行了
mysql> UPDATE `test_innodb_lock` SET `b` = 'asdasd' WHERE `a` = 1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

> 4、行锁两个SESSION同时对不同记录进行写操作

```
# SESSION1 对test_innodb_lock表的`a`=6这一行进行写操作，但是没有commit
mysql> UPDATE `test_innodb_lock` SET `b` = '8976' WHERE `a` = 6;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

# SESSION2 对test_innodb_lock表的`a`=4这一行进行写操作，没有阻塞！！！
# SESSION1和SESSION2同时对不同的行进行写操作互不影响
mysql> UPDATE `test_innodb_lock` SET `b` = 'Ringo' WHERE `a` = 4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

17.3.索引失效行锁变表锁

```
# SESSION1 执行SQL语句，没有执行commit。
# 由于`b`字段是字符串，但是没有加单引号导致索引失效
mysql> UPDATE `test_innodb_lock` SET `a` = 888 WHERE `b` = 8000;
Query OK, 1 row affected, 1 warning (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 1

# SESSION2 和SESSION1操作的并不是同一行，但是也被阻塞了？？？
# 由于SESSION1执行的SQL索引失效，导致行锁升级为表锁。
mysql> UPDATE `test_innodb_lock` SET `b` = '1314' WHERE `a` = 1;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

17.4.间隙锁的危害

> 什么是间隙锁?

当我们用范围条件而不是相等条件检索数据，并请求共享或者排他锁时，`InnoDB`会给符合条件的已有数据记录的索引项加锁，对于键值在条件范文内但并不存在的记录，叫做"间隙(GAP)"。

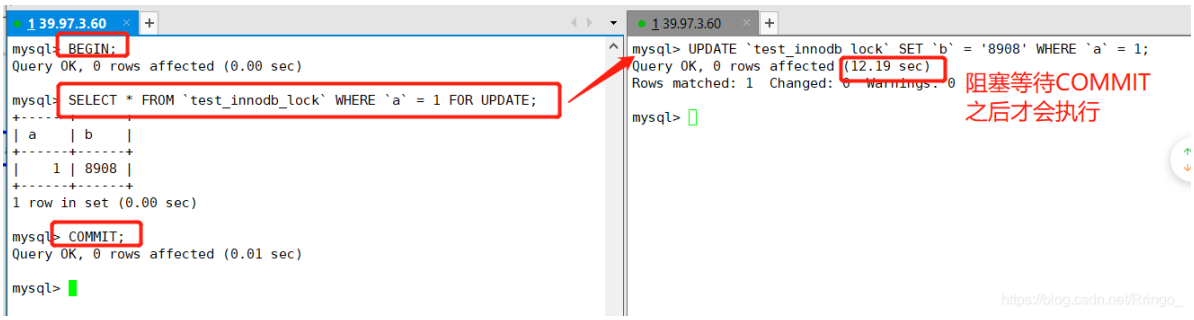
`InnoDB`也会对这个"间隙"加锁，这种锁的机制就是所谓的"间隙锁"。

> 间隙锁的危害

因为`Query`执行过程中通过范围查找的话，他会锁定整个范围内所有的索引键值，即使这个键值不存在。

间隙锁有一个比较致命的缺点，就是**当锁定一个范围的键值后，即使某些不存在的键值也会被无辜的锁定，而造成在锁定的时候无法插入锁定键值范围内的任何数据。**在某些场景下这可能会对性能造成很大的危害。

17.5.如何锁定一行



`SELECT .....FOR UPDATE`在锁定某一行后，其他写操作会被阻塞，直到锁定的行被`COMMIT`。

17.6.案例结论

`InnoDB`存储引擎由于实现了行级锁定，虽然在锁定机制的实现方面所带来的性能损耗可能比表级锁定会要更高一些，但是在整体并发处理能力方面要远远优于`MyISAM`的表级锁定的。当系统并发量较高的时候，`InnoDB`的整体性能和`MyISAM`相比就会有比较明显的优势了。

但是，`InnoDB`的行级锁定同样也有其脆弱的一面，当我们使用不当的时候，可能会让`InnoDB`的整体性能表现不仅不能比`MyISAM`高，甚至可能会更差。

17.7.行锁分析

```
mysql> SHOW STATUS LIKE 'innodb_row_lock%';
+-------------------------------+--------+
| Variable_name                 | Value  |
+-------------------------------+--------+
| Innodb_row_lock_current_waits | 0      |
| Innodb_row_lock_time          | 124150 |
| Innodb_row_lock_time_avg      | 31037  |
| Innodb_row_lock_time_max      | 51004  |
| Innodb_row_lock_waits         | 4      |
+-------------------------------+--------+
5 rows in set (0.00 sec)
```

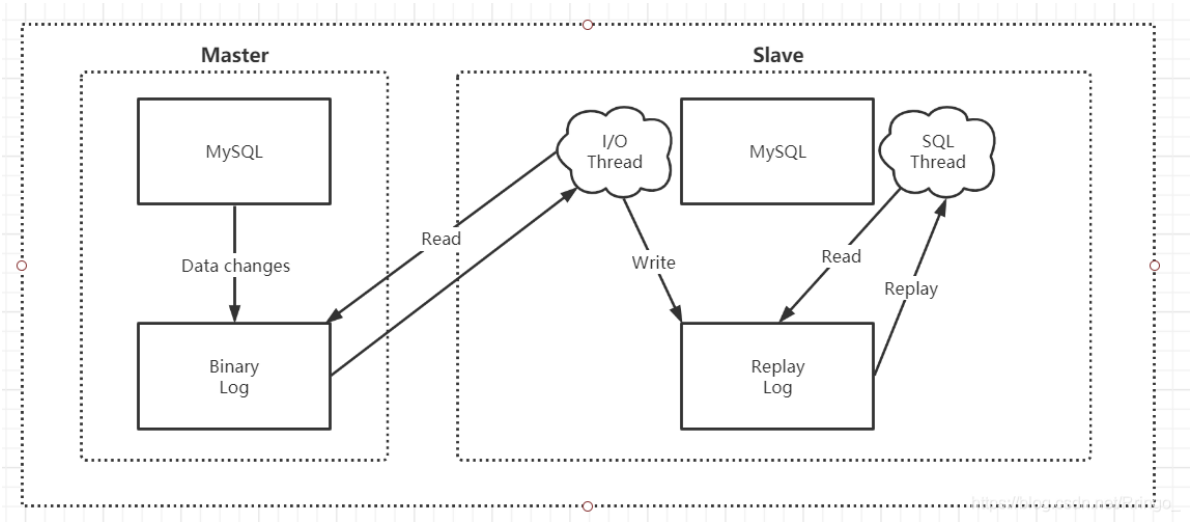
对各个状态量的说明如下：

- `Innodb_row_lock_current_waits`：当前正在等待锁定的数量。
- `Innodb_row_lock_time`：从系统启动到现在锁定总时间长度（重要）。
- `Innodb_row_lock_time_avg`：每次等待所花的平均时间（重要）。
- `Innodb_row_lock_time_max`：从系统启动到现在等待最长的一次所花的时间。
- `Innodb_row_lock_waits`：系统启动后到现在总共等待的次数（重要）。

尤其是当等待次数很高，而且每次等待时长也不小的时候，我们就需要分析系统中为什么会有如此多的等待，然后根据分析结果着手制定优化策略。

18.主从复制

18.1.复制基本原理



MySQL复制过程分为三步：

- Master将改变记录到二进制日志(Binary Log)。这些记录过程叫做二进制日志事件，`Binary Log Events`；
- Slave将Master的`Binary Log Events`拷贝到它的中继日志(Replay Log);
- Slave重做中继日志中的事件，将改变应用到自己的数据库中。MySQL复制是异步且串行化的。

18.2.复制基本原则

- 每个Slave只有一个Master。
- 每个Slave只能有一个唯一的服务器ID。
- 每个Master可以有多个Salve。

18.3.一主一从配置

> 1、基本要求：Master和Slave的MySQL服务器版本一致且后台以服务运行。

```
# 创建mysql-slave1实例
docker run -p 3307:3306 --name mysql-slave1 \
-v /root/mysql-slave1/log:/var/log/mysql \
-v /root/mysql-slave1/data:/var/lib/mysql \
-v /root/mysql-slave1/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=333 \
-d mysql:5.7
```

> 2、主从配置都是配在[mysqld]节点下，都是小写

```
# Master配置
[mysqld]
server-id=1 # 必须
log-bin=/var/lib/mysql/mysql-bin # 必须
read-only=0
binlog-ignore-db=mysql
# Slave配置
[mysqld]
server-id=2 # 必须
log-bin=/var/lib/mysql/mysql-bin
```

> 3、Master配置

```
# 1、GRANT REPLICATION SLAVE ON *.* TO 'username'@'从机IP地址' IDENTIFIED BY
'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'zhangsan'@'172.18.0.3' IDENTIFIED BY
'123456';
Query OK, 0 rows affected, 1 warning (0.01 sec)

# 2、刷新命令
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

# 3、记录下File和Position
# 每次配从机的时候都要SHOW MASTER STATUS;查看最新的File和Position
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+----------------
---+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
Executed_Gtid_Set |
+------------------+----------+--------------+------------------+----------------
---+
| mysql-bin.000001 |      602 |              | mysql            |
  |
+------------------+----------+--------------+------------------+----------------
---+
1 row in set (0.00 sec)
```

> 4、Slave从机配置

```
CHANGE MASTER TO MASTER_HOST='172.18.0.4',
MASTER_USER='zhangsan',
MASTER_PASSWORD='123456',
MASTER_LOG_FILE='mysql-bin.File的编号',
MASTER_LOG_POS=Position的最新值;
# 1、使用用户名密码登录进Master
mysql> CHANGE MASTER TO MASTER_HOST='172.18.0.4',
    -> MASTER_USER='zhangsan',
    -> MASTER_PASSWORD='123456',
    -> MASTER_LOG_FILE='mysql-bin.000001',
    -> MASTER_LOG_POS=602;
Query OK, 0 rows affected, 2 warnings (0.02 sec)

# 2、开启Slave从机的复制
mysql> START SLAVE;
Query OK, 0 rows affected (0.00 sec)

# 3、查看Slave状态
# Slave_IO_Running 和 Slave_SQL_Running 必须同时为Yes 说明主从复制配置成功!
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event # Slave待命状态
                  Master_Host: 172.18.0.4
                  Master_User: zhangsan
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 602
               Relay_Log_File: b030ad25d5fe-relay-bin.000002
                Relay_Log_Pos: 320
```

```
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 602
              Relay_Log_Space: 534
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: bd047557-b20c-11ea-9961-0242ac120002
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more
updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
1 row in set (0.00 sec)
```

> 5、测试主从复制

```
# Master创建数据库
mysql> create database test_replication;
Query OK, 1 row affected (0.01 sec)
```

```
# Slave查询数据库
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test_replication   |
+--------------------+
5 rows in set (0.00 sec)
```

6、停止主从复制功能

```
# 1、停止Slave
mysql> STOP SLAVE;
Query OK, 0 rows affected (0.00 sec)

# 2、重新配置主从
# MASTER_LOG_FILE 和 MASTER_LOG_POS一定要根据最新的数据来配
mysql> CHANGE MASTER TO MASTER_HOST='172.18.0.4',
    -> MASTER_USER='zhangsan',
    -> MASTER_PASSWORD='123456',
    -> MASTER_LOG_FILE='mysql-bin.000001',
    -> MASTER_LOG_POS=797;
Query OK, 0 rows affected, 2 warnings (0.01 sec)

mysql> START SLAVE;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 172.18.0.4
                  Master_User: zhangsan
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 797
               Relay_Log_File: b030ad25d5fe-relay-bin.000002
                Relay_Log_Pos: 320
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 797
              Relay_Log_Space: 534
              Until_Condition: None
```

```
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: bd047557-b20c-11ea-9961-0242ac120002
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more
updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
1 row in set (0.00 sec)
```

▶