## 1.MYSQL的架构介绍

### MYSQL简介

#### MYSQL概述

MYSQL是一个关系型数据库管理系统，由瑞典MSYQL AB公司开发，目前属于Oracle公司。

MYSQL是开源的，所以不需要支付额外的费用。

#### 高级MSQL

完整的MYSQL优化需要很深的功底,SQL易学难精，要写出性能稳定高效可靠简洁的SQL是很有难度的。

主要包含了MYSQL内核、SQL优化、MYSQL服务器的优化、各种参数常量设定、查询语句优化、主从复制、软硬件升级、容灾备份、SQL编程等内容。

### MYSQL Linux版安装

#### RPM方式安装MYSQL

1. 下载对应的MYSQL RPM包;
2. 检查当前系统是否安装过MYSQL

查看命令：rpm -qa |grep -I mysql

删除命令：rmp -e 软件名称

1. 安装MYSQL服务端

安装命令：rpm -ivh MYSQL服务端rpm包名称

1. 安装MYSQL客户端

安装命令：rpm -ivh MYSQL客户端rpm包名称

1. 查看MYSQL是否安装成功

ps -ef |grep mysql

或执行mysqladmin --version

1. 查看MYSQL安装时创建的mysql用户和mysql组

cat /etc/passwd | grep mysql

cat /etc/group |grep mysql

1. Mysql服务器的启动和停止

service mysql start

service mysql stop

使用ps -ef |grep mysql查看是否开启了或停用

1. MYSQ启动后连接MYSQL

Mysql默认没有密码，需要修改密码

/user/bin/mysqladmin -u root password 123456

mysql -uroot -p123456

1. 自启动MYSQL服务

设置开机自启：chkconfig mysql on

chkconfig –list |grep mysql

ntsysv 命令看到[\*]mysql这一行，表示开机后会自动启动mysql

1. 修改配置文件位置

在linux上查看MYSQL的安装目录 ：ps -ef |grep mysql

1. 修改字符编码

Show variables like ‘character%’;

Show variables like ‘%char%’;

需要在配置文件中修改；

### MYSQL配置文件

二进制日志文件log-bin（主从复制）

错误日志log-err:

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；

查询日志log

默认是关闭的，记录查询的sql语句，如果开启会降低MYSQL的整体性能，因为记录日志是需要消耗系统资源的。

数据文件

两系统：

Window：/data目录下

Linux：默认路径：/var/lib/mysql

frm文件：存放表结构

myd文件：存放表数据；

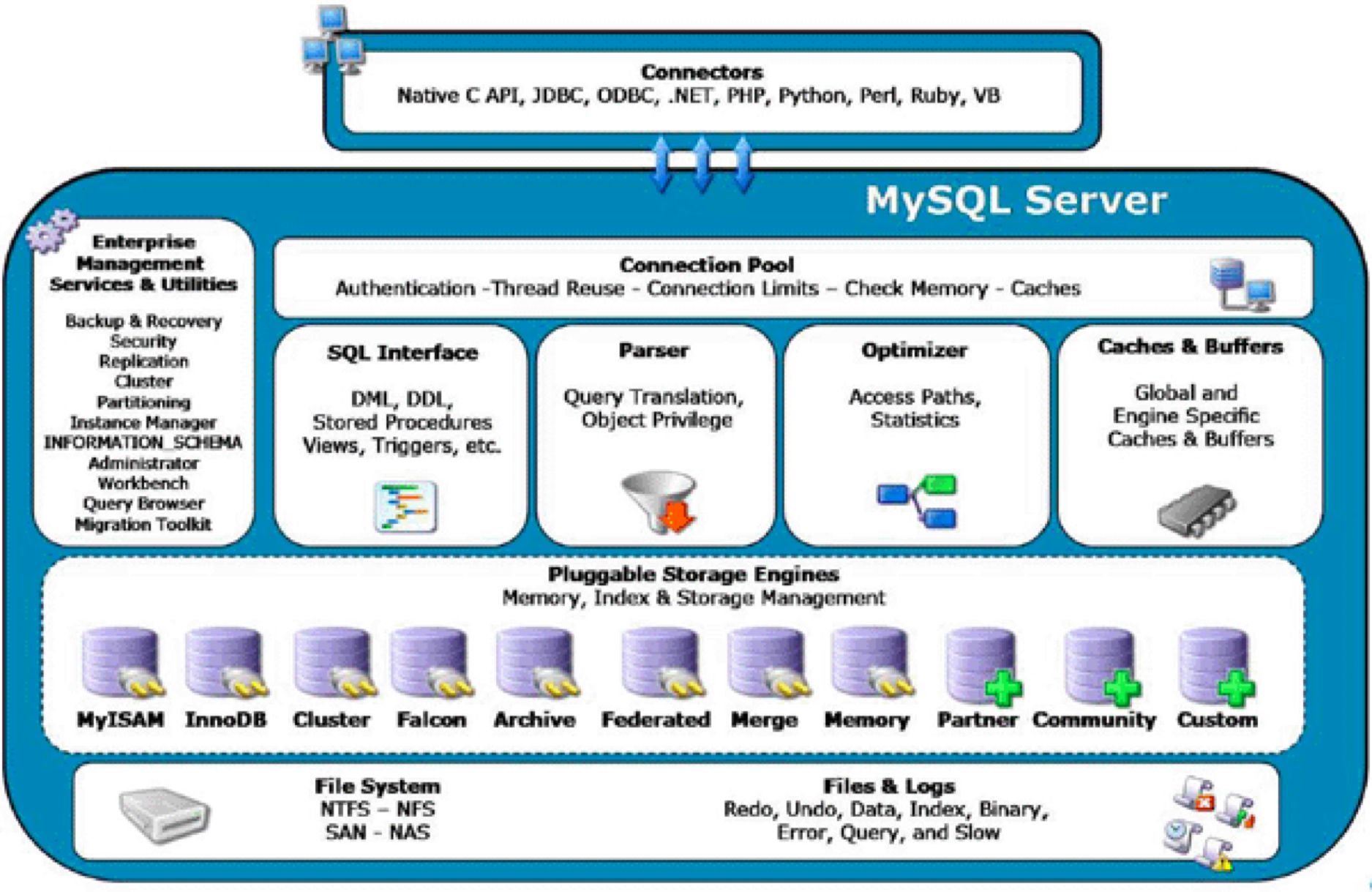
myi文件：存放索引

如何配置

Window my.ini

Linux my.cnf

### MYSQL逻辑架构介绍



连接层：最上层是由一些客户端和连接服务，包含了本地socket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。

服务层：第二层架构完成大多数核心服务功能，如SQL接口，并完成缓存的查询，SQL的分析及内置函数的执行。

引擎层：存储引擎层，负责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不同的功能，可以根据实际情况选择合适的存储引擎。

存储层：将数据存储在运行与设备的文件系统中，并完成与存储引擎的交互。

### MYSQL存储引擎

1. 查看MYSQL已提供的存储引擎

Show engines;

1. 查看当前MYSQL默认的存储引擎；

Show variables like ‘%storage\_engines%’；

|  |  |  |
| --- | --- | --- |
| 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整个表，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。  适合高并发的操作 |
| 缓存 | 只缓存索引，不缓存真实数据 | 不仅缓存索引，还缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响 |
| 表空间 | 小 | 大 |
| 关注点 | 性能 | 事务 |
| 默认安装 | Yes | Yes |

## 2.索引优化分析

### 性能下降SQL慢执行时间长等待时间长

#### 解决问题的思路：

1. 复现
2. 缩小范围

#### 可能的问题

1. SQL语句写得不好
2. 索引失效(单值（给某表的某一个字段加索引）和复合（给某表的多个字段加索引）索引)
3. 关联查询太多join(设计缺陷或不得已的需求)
4. 服务器调优及各个参数设置（缓冲、线程数等）

### 常见通用的join查询

#### SQL的执行顺序

1.FROM <left\_table>

2.ON <join\_condition>

3.<join\_type> join <right\_table>

4.WHERE <where\_condition>

5.GROUP BY <group\_by\_list>

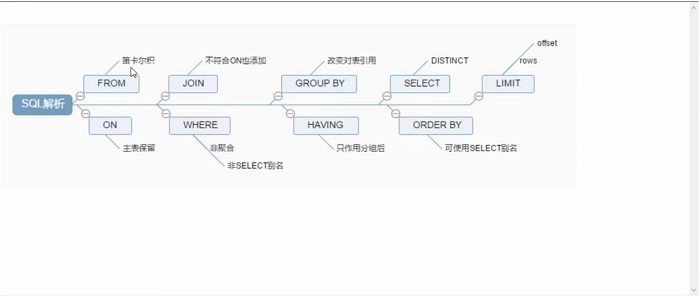
6.HAVING <having\_condition>

7.SELECT

8.DISTINCT <select\_list>

9.ORDER BY <order\_by\_condition>

10.LIMIT <limit\_number>



#### SQL语法：

SELECT DISTINCT

<select\_list>

FROM

<left\_table><join\_type>

JOIN <right\_table> ON <join\_condition>

WHERE

<where\_condition>

GROUP BY

<group\_by\_list>

HAVEING

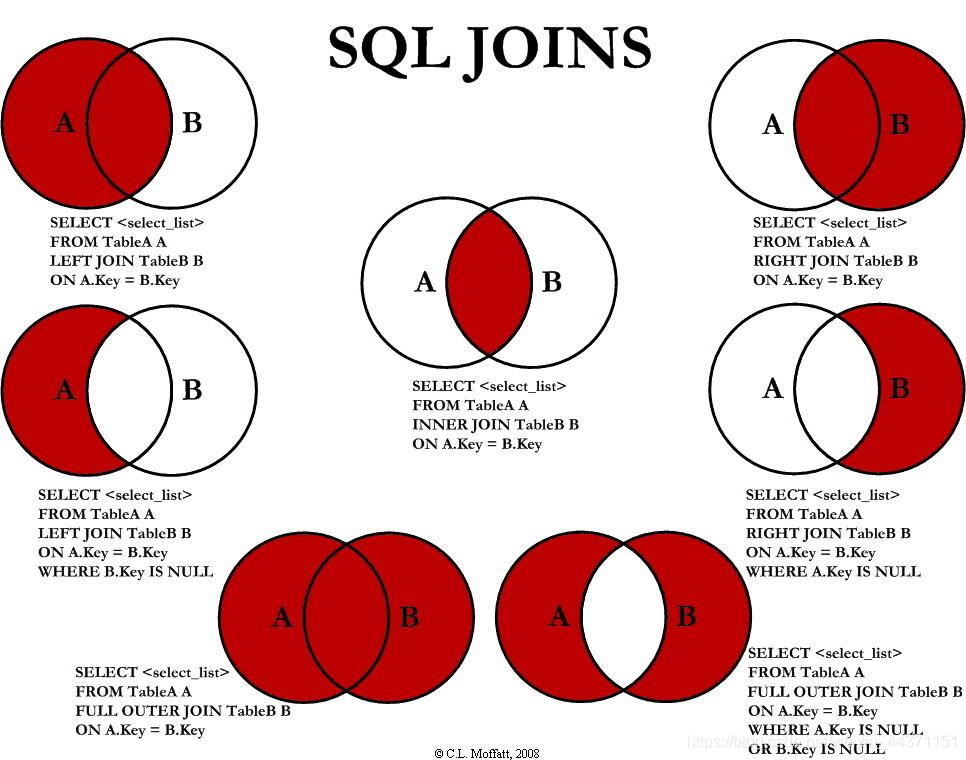
<having\_condition>

ORDER BY

<order\_by\_condition>

LIMIT <limit\_number>

#### Join图解：



MYSQL不支持 full outer join 语法;

MYSQL解决方案：

可以是用union关键字解决；

### 索引简介

#### 什么是索引？

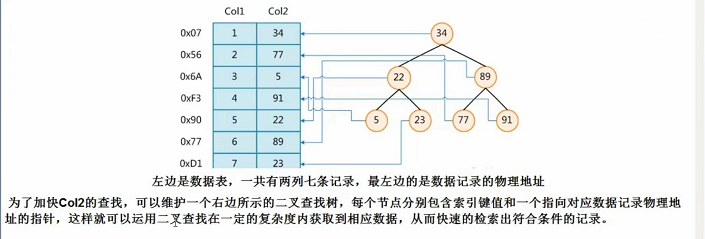
MYSQL官方对索引的定义为：索引（Index）是帮助MYSQL高效获取数据的数据结构。

可以得到索引的本质：索引是数据结构。索引的目的在于提高查询效率。可以类比字典。

如要查“mysql”这个单词，需要先定位到m字母，然后往下找到y字母，再找到剩下的sql。

**可以简单的理解为“排序好的快速查找数据结构”。**在数据本身之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以实现在这些数据结构上实现高级查找算法。这种数据结构就叫索引。

下图是一种可能的索引方式：



一般来说索引本身很大，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。我们平常所说的索引，如果没有特别指明，就是指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，统称为索引。当然，除了B+树这种类型的索引之外，还有哈希索引（hash index）等。

#### 索引优势

类似图书馆建立书目录索引，提高数据检索效率，降低数据库的IO成本。

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗。

#### 索引的劣势

实际上索引也是一张表，该表保存了主键和索引字段，并指向实体表的记录，所以索引列也是占用空间的；

虽然索引提高了查询速度，同时却降低了更新表的速度，如对表进行INSERT,UPDATE和DELETE。因为更新表时，MYSQL不仅要保存数据，还要保存一下索引文件，每次更新添加了索引列的字 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的索引。

#### 索引的分类

##### 单值索引

即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，但允许有空值。

##### 复合索引

即一个索引包含了多个列。

##### 基本语法

创建

CREATE [UNIQUE] INDEX indexName ON mytable(columnName(length));

ALTER mytable ADD [UNIQUE] INDEX [indexName] ON (columnName(length));

删除

DROP INDEX [indexName] on mytable;

查看

SHOW INDEX FROM table\_name\G;

四种方式添加数据表索引

添加主键索引(非空+唯一)

ALTER TABLE tbl\_name ADD PRIMARY KEY(column\_list)

添加唯一索引

ALTER TABLE tbl\_name ADD UNIQUE index\_name(column\_list)

添加普通索引

ALTER TABLE tbl\_name ADD INDEX index\_name(column\_list)

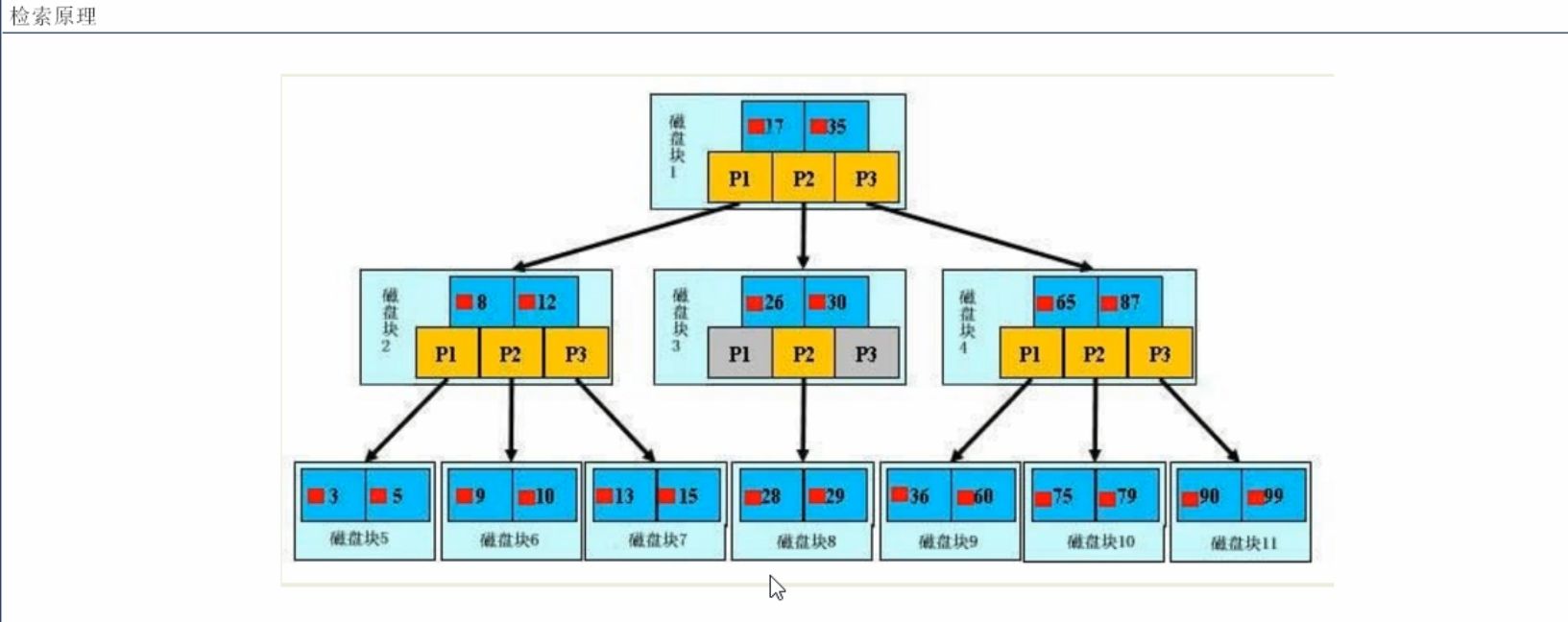
添加FULLTEXT索引,用于全文搜索

ALTER TABLE tbl\_name ADD FULLTEXT index\_name(column\_list)

#### MYSQL索引的结构

##### BTree索引

索引的原理：



##### Hash索引

##### Full-text索引

##### R-Tree索引

#### 什么情况下需要创建索引

1. 主键自动创建唯一索引
2. 频繁作为查询条件的字段应该创建索引
3. 查询中于其他表管理的字段，外键关系建立索引
4. 频繁更新的字段不适合建立索引，因为每次更新不单单更新了记录还会更新索引。
5. Where条件里用不到的字段不要创建索引。
6. 单键/组合索引的选择问题？（在高并发下倾向创建组合索引）
7. 查询中排序的字段，排序的字段若通过索引去访问将大大提高排序速度
8. 查询中统计或分组字段

#### 那些情况要建索引

1. 表记录太少
2. 经常增删改的表，因为提高了查询的速度的同时却降低了更新表的速度，如对表进行INSERT,UPDATE,DELETE.因为更新表时，MYSQL不仅要保存数据，还要保存一下索引文件。
3. 数据重复且分布平均的表字段，因此应该只为经常查询和最经常排序的数据建立索引。（性别）

索引的选择性是指索引列中不同值的数目与表记录数的比。如果一个表中有2000条记录，表索引中有1980个不同的值，那么索引的选择性是1980/2000=0.99,一个索引的选择性越接近1，这个索引的效率就越高。

1. 如果某个数据列包含许多重发的内容，为它建立索引就没有太大的实际效果

### 性能分析

#### Mysql Query Optimizer

MYSQL中有专门负责优惠SELECT语句的优化器模块。主要功能是：通过计算分享系统中收集到的统计信息，为客户端请求中的Query提供MYSQL认为最优的执行计划。

当客户端想MYSQL请求一条Query，命令解析器模块完成请求分类，区别出事SELECT并发给MySQL Query Optimeizer时，MySQL Query Optimeizer首先会对整条Query语句进行优化，处理掉一些常量表达式的预算，直接换算成常量值。并对Query中的查询条件进行简化和转换，如去掉一些无用或者显而易见的条件、结构调整等。然后分析Query中的Hint信息（如果有），看显示的Hint信息是否可以完全确定该Query的执行计划。如果没有Hint或Hint信息还不足以完全确定执行计划，则会读取所涉及对象的统计信息，根据Query进行写相应的计算分析，然后再得出最后的执行计划。

#### MYSQL常见瓶颈

CPU：CPU在饱和的时候一般发生在数据装入内存或从磁盘上读取数据的时候；

IO:磁盘I/O瓶颈发生在装入数据大于内存容量的时候；

服务器硬件性能的瓶颈：top,free,iostat和vmstat来查看系统的性能状态；

#### Explain

##### 什么是Explain（查看执行计划）

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，分析查询语句或是表结构的性能瓶颈

官网地址：<https://dev.mysql.com/doc/refman/8.0/en/>

##### Explain能干什么？

1. SQL语句中表的读取顺序 **id**
2. SQL语句中数据读取操作的操作类型 **select\_type**
3. 有那些索引可以使用 **possible\_keys**
4. 有那些索引被实际使用 **key**
5. 表之间的引用 ref
6. 每张表有多少行被优化查询 rows

##### 如何使用

###### 语法：

explain + SQL语句；

###### 执行计划包含的信息

**id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |**

##### 各字段解释（id，type，key,rows,Extra）

###### Id

SELECT查询的序列号，包含一组数字，表示查询中执行SELECT子句或操作表的顺序

三种情况：

id相同，执行顺序由上而下

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（Derived =衍生）

如果ID相同，可以认为是一组，从上往下顺序执行；

在所有组中，id越大，优先级越高，越先执行；

###### select\_type

1. 常见的值

SIMPLE | PRIMARY | SUBQUERY | DERIVED | UNION | UNION RESULT

查询的类型，主要用于区别普通查询、联合查询、子查询等的复制条件

**SIMPLE**：简单的select查询，查询中不包含子查询或UNION;

**PRIMARY**:查询中包含复杂的子查询,最外层查询则被标记为PRIMARY;

**SUBQUERY**:在select或where列表中包含了子查询

**DERIVED**：在FROM列表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，把结果放到临时表里。

**UNION**：若第二个SELECT出现在UNION之后，则被标记为UNION；

若UNION包含在FROM子句的查询中，外层SELECT被标记为：DERIVED；

**UNION RESULT**：从UNION表获取结果的SELECT。

###### Table

显示这一行的数据是关于那张表的；

###### Type

常见值：**ALL | index | range | ref | eq\_ref | const ,system |NULL**

访问类型排列

type显示的是访问类型，是一个较为重要的指标。结果值从最好到最差依次是 system>const>eq\_ref>ref>fulltext>ref\_or\_null>index\_merge>unique\_subquery>index\_subquery>range>index>ALL

**system>const>eq\_ref>ref>range>index>ALL**

一般来说，得保证查询至少达到range级别，最好能达到ref级别。

显示查询使用了何种类型，从最好到最差：

**system>const>eq\_ref>ref>range>index>ALL**

**system**:表只有一行记录(等于系统表)，这个是const类型的特例，平时不会出现，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，如果主键置于列表中，MYSQ就能将该查询转换为一个常量。

**eq\_ref**:唯一索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描。

**ref**:非唯一索引扫描，返回匹配某个单独的所有行。本质上是一种索引访问，它返回所有的匹配某个单独值的行，然而，他可能会找到多个符合条件的额行，所以属于查找和扫描的混合体。

**range**:只检索给定范围的行，使用一个索引来选择行，key列显示使用了那个索引。一般就是你在where语句中出现了between、<、>、in等的查询。这种范围扫描索引扫描比全表扫描要好，因为它只需要开始于索引的某一点，而结束于另一点，不用扫描全部索引.

**Index**: Full Index Scan,index与ALL的区别为index类型只遍历索引树。因为索引文件通常比数据文件小。（也就是说虽然all和index都是读全表，但index是从从索引中读取的，而all是从硬盘中读的）

**ALL**: Full Table Scan：将遍历全表以找到匹配的行

备注：一般来说，得保证查询至少达到range级别，最好达到ref。

###### possible\_keys

显示可能应用在这张表中的索引，一个或多个；

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。

查询中若使用了覆盖索引,则该索引仅出现在key列表中；

###### 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。在不损失精确性的情况下，长度越短越好。

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。

查询的结果越精确，条件越多，消耗的字节数越多，代价就越大（key\_len越大）

###### Ref

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

###### Rows

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所需要读取的行数。

###### filtered

###### Extra

包含了不适合在其他列中显示但十分重要的额外信息

1. **Using filesor**t: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称为“文件排序”

1. **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临时表。常见于排序order by 和分组查询group by。
2. **Using index**:表示对应的select操作中使用了覆盖索引(Covering Index),避免访问了数据行，效率不错。如果同时出现using where，表明索引被引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（covering Index）：或说为索引覆盖；

就是select的数据列只用从索引中获取，不必读取数据行，MYSQL可以利益索引返回select列表中的字段，而且不必根据索引再次读取数据文件，换句话说查询列要被所建的索引覆盖

1. Using where：表明使用了where过滤
2. Using join buffer：使用了连接缓存；
3. Impossible where：where子句的值总是false，不能用来获取任何元组
4. Select table optimized away：在没有Grop by 子句IDE情况下，基于索引优化MIN|MAX操作或者对于MyISAM存储引擎优化（count（\*））操作，不必等到执行阶段再进行计算，查询执行计划生成阶段即完成优化。
5. distinct：优化distint操作，在找到第一匹配的元组后即停止找同样值的动作。

### 索引优化

#### 单表优化

按照BTree索引的工作原理，当条件是一个范围值，会导致相应索引失效。

#### 两表优化

左连接加右表，右连接加左表；

#### 三表优化

## 查询截取分析

### 查询优化

### 慢查询日志

### 批量数据脚本

### Show profile

### 全局查询日志

## MYSQL锁机制

## 主从复制