<https://blog.csdn.net/m0_37984616/article/details/81047676?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

A screenshot of a cell phone

Description automatically generated

连接池组件、管理服务和工具组件、SQL接口组件、查询分析器组件、优化器组件、缓冲组件、插件式存储引擎、物理文件；

1、连接层：主要完成一些类似于连接处理，授权认证及相关的方案；

2、服务层：主要完成大多数核心服务功能；

3、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；

注：储存引擎式基于表的，而不是数据库的；

## MySQL存储引擎

InnoDB 和 MyISAM区别：

1、InnoDB支持主外键、事务；

2、InnoDB是行锁，操作时候只锁一行数据，适合高并发；MyISAM是表索；

3、InnoDB不仅缓存索引，还缓存真实数据；MyISAM只缓存索引；

4、InnoDB需要表空间大；

5、InnoDB关注事务，MyISAM关注性能（查）；

## SQL优化

问：执行时间长，等待时间长

可能是SQL语句太次，索引失效，关联查询太多join，服务器调优及参数设置（缓冲池）

SQL执行顺序：

FROM <LEFT\_TABLE>

ON <JOIN\_CONDITION>

<JOIN\_TYPR> JOIN <RIGHT\_TABLE>

WHERE

GROUP BY

HAVING

SELECT

DISTINCT

ORDER BY

A picture containing screenshot

Description automatically generated

最后两个由于MySQL不支持FULL OUTER JOIN，所以用 UNION 连接，即1和2的语句进行UNION得到6，3和4的语句进行UNION得到7。

## 索引

索引：排好序的快速查找数据结构；

优点：提高数据检索的效率，降低数据排序的成本；

缺点：会降低更新表的速度；

单值索引：一个索引只包含单个列，一个表可以由多个单值索引；

唯一索引：索引列的值必须唯一，允许有空值；

复合索引：即一个索引包含多个列；

### 索引结构：

BTree索引，Hash索引，full-text全文索引，R-Tree索引；

B+树与B树的不同在于：

（1）所有关键字存储在叶子节点，非叶子节点不存储真正的data

（2）为所有叶子节点增加了一个链指针

下图是BTree索引数据结构

### 索引性能分析：

Explain（执行计划）：

使用：EXPLAIN + sql语句；

作用：

1、表的读取顺序；

2、数据读取操作的操作类型；

3、哪些索引可以使用；

4、那些索引被实际使用；

5、表之间的引用；

6、每张表有多少行被优化器查询；

如果查询的两个表大小相当，那么用in和exists差别不大。

如果两个表中一个较小，一个是大表，则子查询表大的用exists，子查询表小的用in：

所以无论那个表大，用not exists都比not in要快。not in 那么内外表都进行全表扫描，没有用到索引；而not extsts 的子查询依然能用到表上的索引。

### Join连接索引分析

单表分析

例：select id from A where c1 = 1 and c2 > 1 order by v1 desc limit 1;

建立联合索引（c1,c2,v1），但explain时候发现 type 是range，extra中使用using filesort，这需要优化；

产生原因：按照BTree索引工作原理，先排序c1，如果c1相同，排序c2，c2相同在排序v1，当c2字段在联合索引中处于中间位置，因为c2 > 1条件是一个范围值（range），MySQL无法利用索引在对后面的v1部分进行索引。所以建立（c1，v1）解决这个问题。

双表分析

左连接加在右表的索引，右连接加在左表的索引；

Left Join 条件用于确定如何从右表搜索行，左边数据一定有，所以右边数据一定要建索引。

三表分析

建索引和双表的原理相同；

总结

尽可能减少Join语句的NestedLoop的循环总次数，永远用小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；

## 索引失效

1、最好全值匹配；

2、最左前缀法则：如果索引了多列，查询从索引的最左前列开始，且不能跳过索引中的列；

3、不在索引列上做任何操作（计算，函数，类型转换），会导致索引时校而转向全表扫描；

4、存储引擎不能使用索引中范围条件右边的列，即范围之后全失效；

5、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selec \*；

6、MySQL在使用不等于的时候无法使用索引会导致全表扫描；

7、is null，is not null 也无法使用索引；

8、like 以通配符开头（’%aa‘）索引会失效，变成全表扫描；

9、字符串不加单引号，索引失效；

10、少用 or，用它来连接时候会索引失效

覆盖索引：建索引的列和要查询的列相同，例如索引列是c1，c2，select 查询的也是c1，c2；

定值是常量，范围之后必失效，最终看排序，一般order by 是给个范围；

group by 分组，分组之前必排序，顺序不对会有临时表产生；

### 永远小表驱动大表

in 和 exists 选择：

// 工作原理，先查B表数据，然后查A的 id

select \* from A where id in (select id from B)

// 工作原理，先查A表的id，然后查B表的id

select \* from A a where exists (select 1 from B b where a.id = b.id )

// 结论：当B表的数据小于A表时候用 in；当A表数据小时候用 exists

Order By 排序

MySQL支持两种排序，index和fileSort，index效率高，它指MySQL扫描索引本身完成排序。

Order By满足两种情况使用index：

1、Order By 语句使用索引最左前列

2、使用where子句与Order By子句条件组合满足索引最左前列

如果不在索引列上，fileSort有两种算法，4.1版本之前双路排序，进行两次IO；之后单路排序，进行一次IO；

Order By时不要select \*，只查询所需要的字段；当两种算法的数据超出sort\_buffer的容量会创建tmp文件进行合并运算，导致多次IO，所以需要尝试提高sort\_buffer\_size 和max\_length\_for\_sort\_ size。

为排序使用索引，MySQL能为排序与查询使用相同的索引

## 查询截取分析

1、观察，查看慢SQL情况；

2、开启查询日志，设置阀值；

3、explain 分析；

4、show profile 查看执行细节和生命周期情况

5、dba 进行参数调优

### 慢查询日志

响应时间超过long\_query\_tine的SQL，被记录到慢查询日志中。

## MySQL锁机制

分类可以分为读锁（共享锁）和写锁；表锁和行锁；

表锁：偏向MyISAM存储引擎，开销小，加锁快，并发度低；

加锁：lock table tablename read，tablename2 write；解锁：unlock tables;

查看哪些表被锁了 show open tables;

分析表锁定

show status likes ‘table%’;

Table\_locks\_immediate：产生表级锁定的次数；

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；

MyISAM的读写锁调度是写优先，所以不适合做写为主表的引擎。

行锁：偏向InnoDB存储引擎，开销大，会出现死锁，锁的粒度最小，发生锁冲突的概率最低，并发度高。

InnoDB与MyISAM最大不同是InnoDB支持事务，并且采用行级锁；