# .Mysql Query Optimizer

1. Mysql中专门负责优化select语句的优化器模块，主要功能：通过计算分析系统中收集的统计信息，为客户端请求的query提供他认为最优的执行计划（他认为最优，不见得是DBA认为最优的，这部分最耗时间）。
2. 当客户端向MySQL请求一条query，命令解析器模块完成请求分类，区别出是select并转发给MySQL Query Optimizer时，MySQL Query Optimizer首先会对整条query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，如去掉一些无用或显而易见的条件、结构调整等。然后然后分析query中的Hint信息（如果），看现实的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时候

## IO

磁盘I/O瓶颈发生在装入数据远大于内存容量的时候

## 服务器硬件的性能瓶颈

top，free，iostat和vmstat来查看系统的性能状态

# .explain（执行计划）

## 介绍

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是如何处理你的SQL语句的。分析你的查询语句或是表结构的性能瓶颈。

## 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. 哪些索引被实际使用
5. 表之间的引用
6. 每张表有多少行被优化器查询

## 用法

Explain+SQL语句

出现含有以下字段的表格

Id select\_type table type possible\_keys key key\_len ref rows extra

## 字段解释

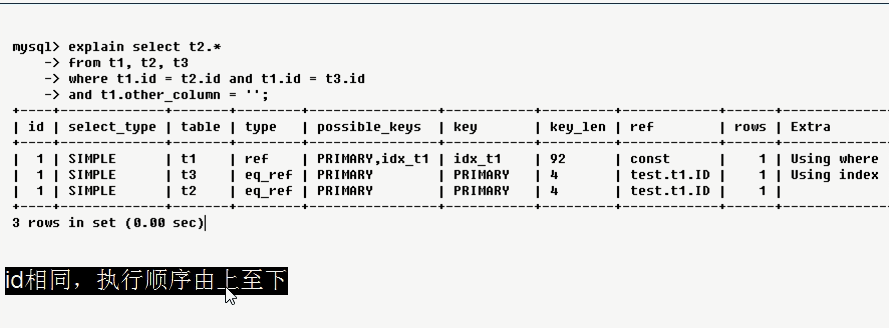
### Id（重要）

#### 解释

**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。**

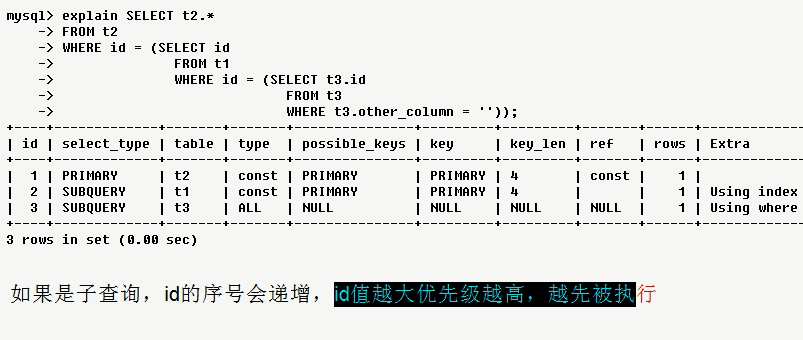
#### 三种情况

1. Id相同，执行顺序由上至下



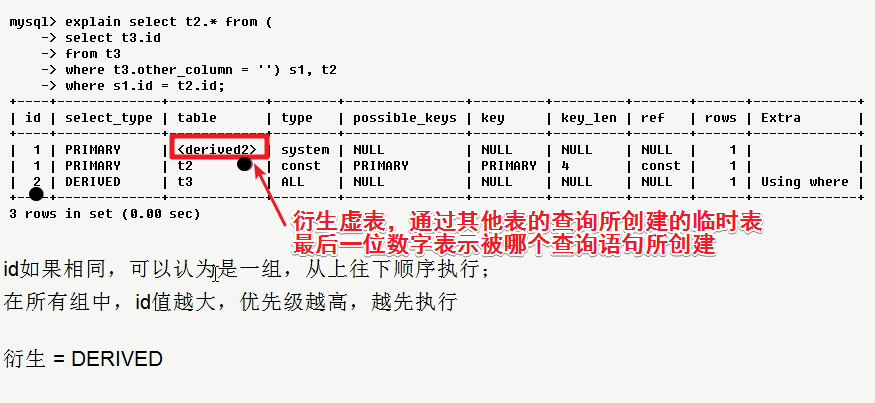
这里SQL写的时t1，t2，t3，但是实际执行顺序是t1，t3，t2。

1. 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。



这里的执行顺序是t3，t1，t2

1. Id相同不同，同时存在



这里的执行顺序为t3，derived2，t2

#### 总结

**相同顺序走，不同，大的先执行**

### select\_type

#### 解释

查询类型，主要是区别普通查询、联合查询，子查询等复杂查询

#### 取值

1. SIMPLE

简单的select查询，查询中不包含子查询或者UNION

1. PRIMARY

查询中若包含任何复杂的子部分，最外层查询会被标记

1. SUBQUERY

在select或where列表中包含的子查询

1. 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，把结果放在临时表里。

1. UNION

若第二个select出现在UNION之后，则被标记为UNION；

若UNION包含在FROM子句的子查询中，外层select将被标记为DERIVED

1. UNION RESULT

从UNION表获取结果的SELECT

### Table

这一行数据是关于那张表的

### Type（重要）

访问类型

#### 八种取值

|ALL|，|index|，|range|，|ref|，|eq\_ref|，|const,system|，|null|

**从最好到最次的顺序**

System>const>eq\_ref>ref>range>index>ALL

一般，最少得保证到range，最好到ref级别

#### System

表中只有一行记录（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就能将该查询转换为一个常量，开发中少用。

例如，通过id查询员工，查询条件为常量。查询条件需是一个索引，主键都是索引。

#### Eq\_ref

唯一性索引扫描，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的扫描。恰好只查到一条数据。

例如一个部门只有一个员工，查询这个部门的员工，只有一个，查询条件为常量。查询条件需是一个索引。

#### Ref

非唯一性索引扫描，返回匹配某个单独之的所有行，本质上也是一种索引访问，它访问所有匹配某个单独值的行，然而，它可能会找到多个符合条件的行，多以他应该属于查找和扫描体的混合体。

例如：查询一个部门的所有员工，where一个常量，返回多条记录。查询条件需是一个索引。

#### Range

只检索给定范围的行，使用一个索引来选择行。Key列显示使用了哪个索引，一般就是在你的where语句中出现了between、<、>、in等的查询。这种范围扫描比全表扫描要好，因为它只需要开始于索引的某一点，而结束与另一点，不用扫描全部索引。

例如，id in(1,2,3)或者id between 3 and 7

#### Index

全索引扫描，只遍历索引树，通常比all块，因为索引文件通常比数据文件小（一般索引都会被加载到内存中，index直接从索引中读取，而all是从硬盘中读取的）

例如，查询表中所有id

#### All

将遍历全表已找到匹配的行

例如select \* from t1

### Possible\_keys

显示可能用用在这张表中的索引，一个或多个。查询所涉及的字段上若存在索引，该索引将被列出，但不一定被查询实际使用。MySQL自己分析判断后，会实际使用他认为合适的索引。

### Key（重要）

MySQL实际用到的索引，如果为NULL，则没有使用索引，可能没建索引，也有可能建了索引却没有用上。

查询中如果出现率覆盖索引，则该索引仅出现在key列表中。

### 补充一点

Possible\_keys（涉及查询字段的所有索引，理论上使用）和key（实际用到的索引）可以判断是否用到索引，或者索引是否失效，在多个索引竞争的过程中mysq最终用到了哪个索引。

一般而言，理论上使用的索引会和实际使用的索引多或一致，但是有一种特殊情况。

假设一张表建了一个索引，这个索引有两个字段，现在查找这两个字段，进行全表扫描，此时的type为index，扫描全部索引，这个时候理论上一个索引也不到，但实际上用到了这个索引。

### 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。在不损失精度性的情况下，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，不是通过表内检索得出。

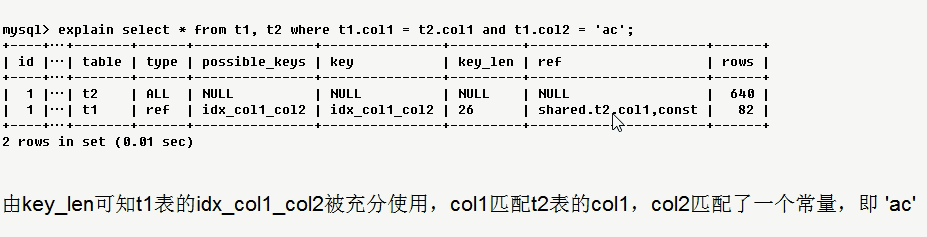
例如，现有索引（姓名，地址），查姓名，用到该索引。查姓名和地址两个条件，也用到该索引。后者比前者精度更高，key\_len也更大。

### Ref

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

如，where id = 1 这里id等于一个常数，会显示const

如，t1连接t2，t1.id=t2.t1id 这里会显示ti.id引用哪个数据库哪个表的哪个字段，例如test.t2.t1id。



### Rows（重要）

根据表统计信息及索引选用情况，大致估算出找到所需记录所需要读取的行数。

每张表有多少行被优化器查询。

### Extra（重要）

此列包含了，不适合在其他列中显示，但是十分重要的额外信息。

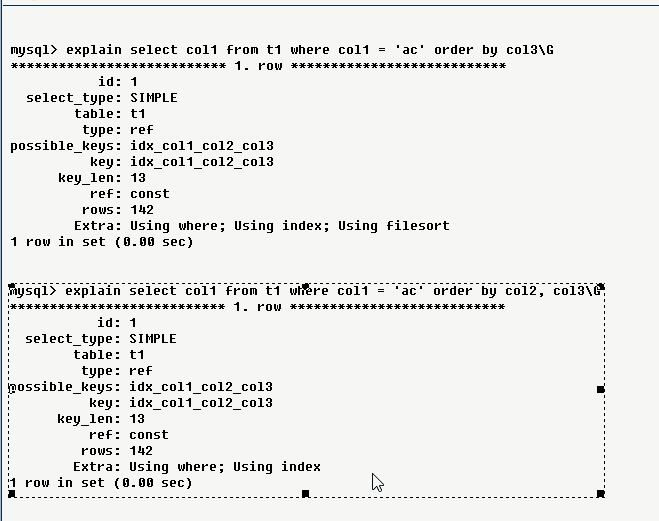
#### \*Using filesort

**出现这个，很不好**。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排序操作成为“文件内排序”。

例如，当前创建索引t1(col1,col2,col3)，当前查询按col1查询，排序按col3排序，col2没有用上，则本次排序会使用文件内排序，索引在排序时不会被使用上。

如果把3个字段全部使用上，比如按照col2和col3两个字段进行排序，就会在排序的时候用上索引。



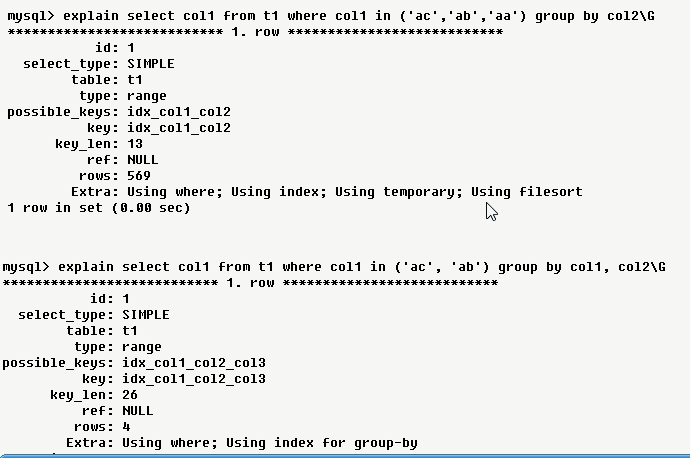
#### \*Using temporary

**出现这个，更不好。**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order by和分组group by。

例如，当前创建索引t1(col1,col2)，查询，col1 in(1,2,3)，然后用col2分组查询，则效率会大大下降，分组时用不上，还会创建临时表。

如果在分组时，把整个索引都用上，就会大大改善。例如分组时按col1,col2分组，这个时候既不会创建临时表，也不会出现文件内部排序。

在查询时最好不要出现临时表，当数据量特别大的时候，创建临时表会特别占用系统内部资源，释放资源时又要大动干戈，会拖慢整个系统。



#### \*Using index

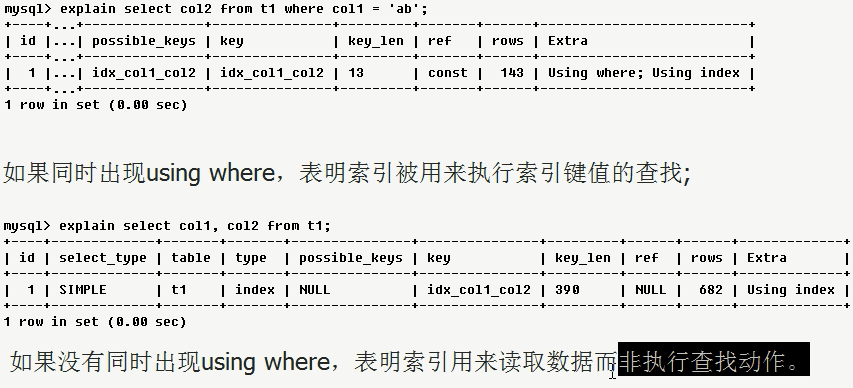
**出现这个比较好。**表示相应的select操作中使用了覆盖索引，避免访问了表的数据行，效率不错！

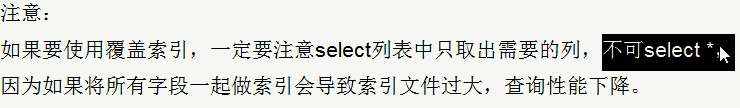
如果同时出现using where，表明索引被用来执行索引键值的查找（索引中某一个字段）。

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（如查找的列刚好和一个索引完全一样）。

##### 覆盖索引

就是select的数据列只用从索引中就能够获取，不必读取数据行，MySQL可以利用索引返回select列表中的字段，而不必根据索引再次读取数据文件，换句话说查询列要被所创建的索引覆盖（查询的字段刚好就是创建的索引，或者部分满足）。





#### Using where

表明使用了where过滤

#### Using join buffer

表明使用了连接缓存，如果连接特别多，可以在配置文件中的缓冲区的join buffer可以调大一点。

#### Impossible where

Where子句的值总是false，不能用来获取任何元组。

例如where 1=2

#### Select tables optimized away

在没有groupby子句的情况下，进行的优化。**不常见。**

#### Distinct

找到第一匹配的元组后就停止，**不常见。**