Explain命令在解决数据库性能上

是第一推荐使用命令

大部分的性能问题可以通过此命令来简单的解决

Explain可以用来查看SQL语句的执行效果，

可以帮助选择更好的索引和优化查询语句，

写出更好的优化语句。

使用方法:**explain  sql 语句;**

示例:



在执行以上语句后

我们会发现,会有一个执行结果.

显示不现的字段名称如下:

**id**

**select\_type**

**table**

**partitions**

**type**

**possible\_keys**

**key**

**key\_len**

**ref**

**rows**

**filtered**

**Extra**

接下来, 我们就对每一个字段进行详细讲解

执行结果字段共有如下些:

**1、id: select查询的序列号**

包含一组数字,

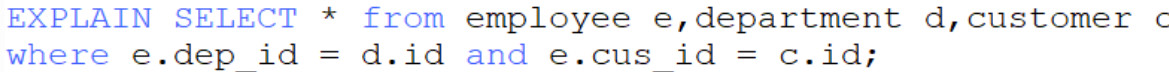
表示查询中执行select子句或操作表的顺序

值的三种情况:

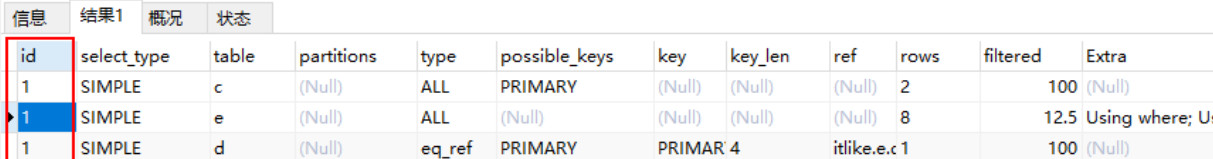
**id相同:**

执行顺序由上到下

示例: 多表连查



执行结果:



分析:

结果id值相等,表执行顺序为c,e,d

**id不同:**

如果是子查询,

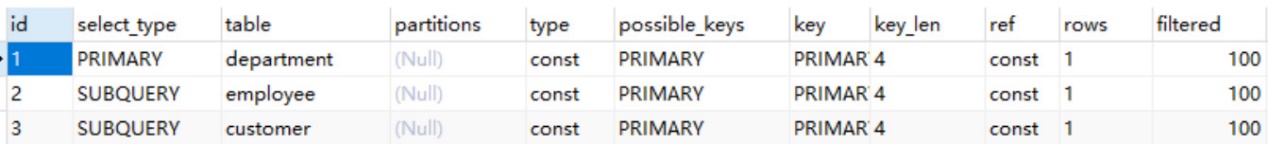
id的序号会递增,

id值越大优先级越高,优先被执行.

示例:

E:\WeChat_Files\Files\WeChat Files\wyp9334\FileStorage\Temp\36e35f07ce3c03eb3af3ff7fd6f5c000

执行结果:



分析:

结果id的序号递增,

表执行顺序为:customer,employee,department

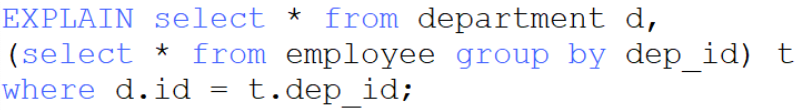
**id相同不同,同时存在:**

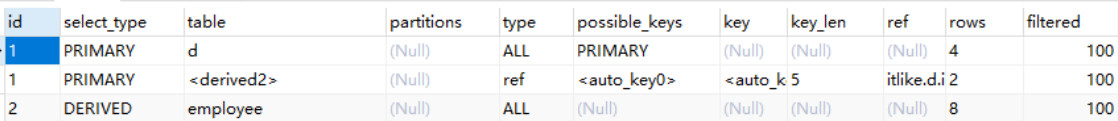
可以认为是一组,从上往下顺序执行

在所有组中,id值越大,优先级越高,越先执行

deriverd 衍生出来的虚表

示例:

执行结果:



分析:

id的执行顺序为1,1,2

根据优先级越高,越先执行原则

表执行顺序为:

employee, deriverd 衍生表 d

**2、select\_type:查询类型**

主要用于区别普通查询,联合查询,子查询等复杂查询

结果值有如下:

**SIMPLE:**

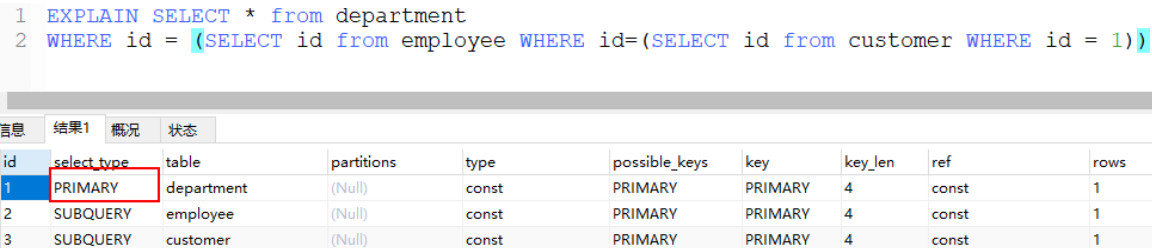
简单select查询,查询中不包含子查询或者UNION

**PRIMARY:**

查询中若包含任何复杂的子查询,

最外层查询则被标记为primary

示例:



执行语句是一个子查询

department在最外层,被标记为primary

**SUBQUERY:**

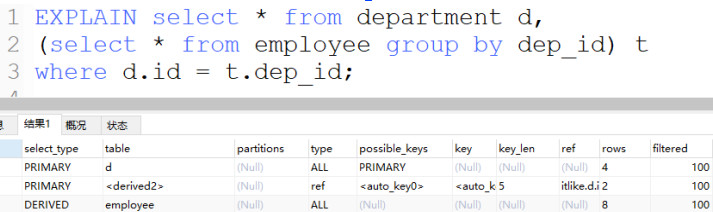
在select或where中包含了子查询

**DERIVED:**

在from列表中包含的子查询被标记为derived(衍生)

把结果放在临时表当中

示例:



**UNION:**

若第二个select出现的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deriver

**UNION RESULT:**

从union表获取结果select

两个UNION合并的结果集在最后

示例:



**3、table:显示这一行的数据是关于哪张表的**

**4、partitions:显示查询访问的分区**

**5、type:访问类型排列**

结果值最好到最差:

**system:**

表中有一行记录(系统表)  这是const类型的特例,

平时不会出现

**const:**

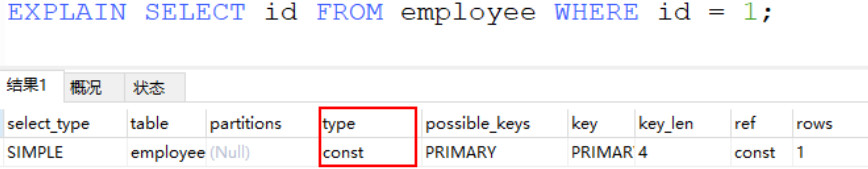
表示通过索引一次就找到了

const用于比较primary 或者 unique索引.

直接查询主键或者唯一索引

因为只匹配一行数据,所以很快

示例:



**eq\_ref:**

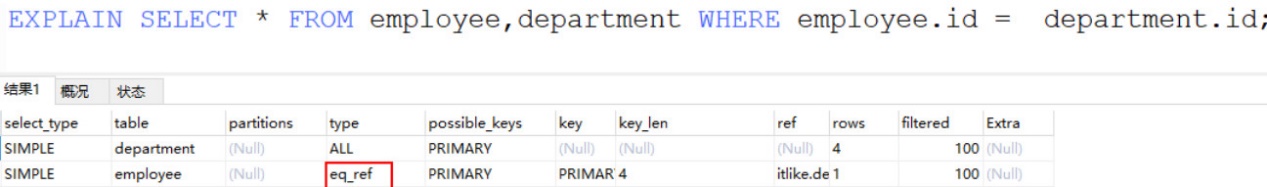
唯一性索引扫描

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扫描

表数据:





**ref:**

非唯一性索引扫描,返回匹配某个单独值的所有行

本质上也是一种索引访问

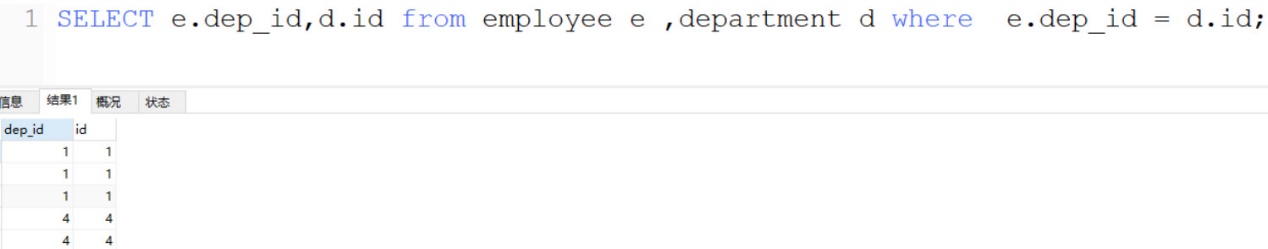
它返回所有匹配某个单独值的行

可能会找到多个符合条件的行,

所以它应该属于查找和扫描的混合体

示例:





**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

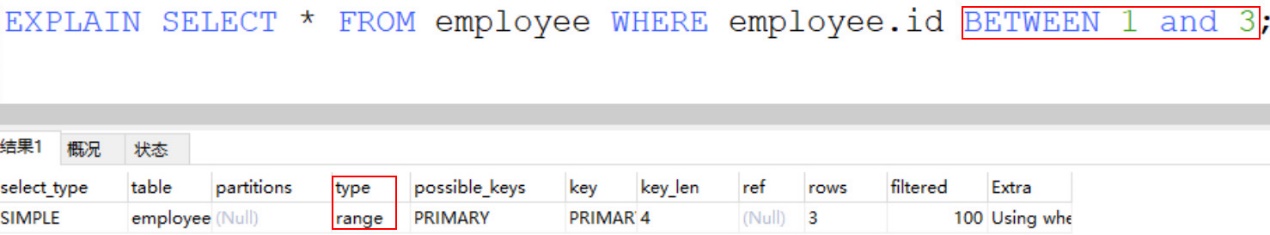
一般就是在你的where语句中出现between\<\>\ in等查询

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的某一点.而结束语另一点

不用扫描全部索引

示例:

**index：**

Full Index Scan

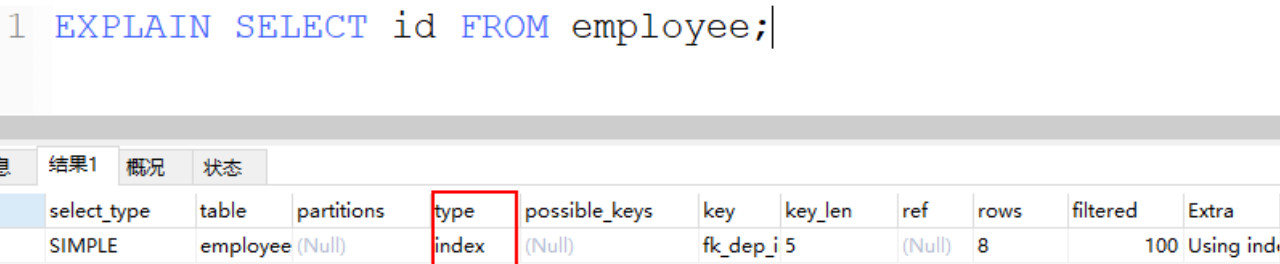
index与All区别为index类型只遍历索引树,

通常比All要快,因为索引文件通常比数据文件要小

all和index都是读全表,

但index是从索引中读取,all是从硬盘当中读取

示例:



**ALL：**

将全表进行扫描,从硬盘当中读取数据

如果出现了All 切数据量非常大, 一定要去做优化

示例:

要求

一般来说,保证查询至少达到range级别

最好能达到ref

**6、possible\_keys:**

key与keys主要作用,

是查看是否使用了建立的索引,

也即判断索引失效

在建立多个索引的情况下,

mysql最终用到了哪一个索引

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

可能自己创建了4个索引,

在执行的时候,

可能根据内部的自动判断,

只使用了3个

**7、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

possible\_keys与key关系

理论应该用到哪些索引

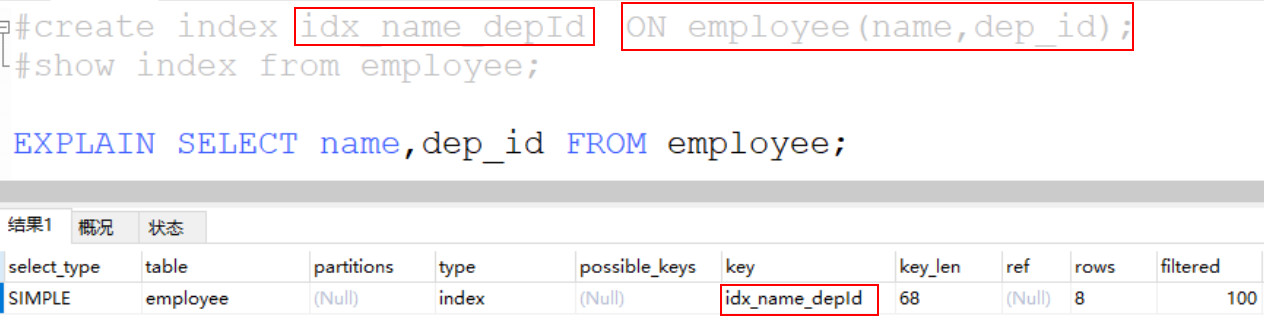
实际用到了哪些索引

覆盖索引 查询的字段

和建立的字段刚好吻合,

这种我们称为覆盖索引

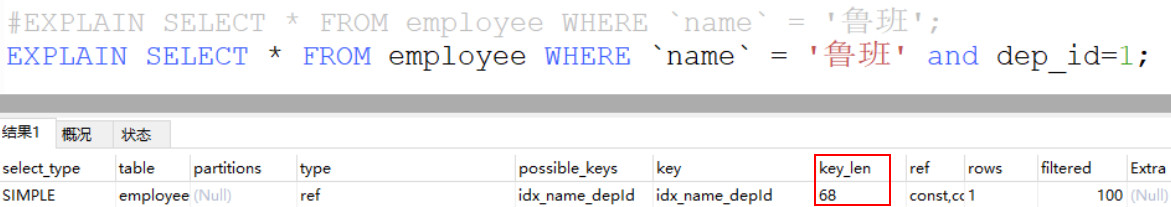
示例:



**8、key\_len:表示索引中使用的字节数**

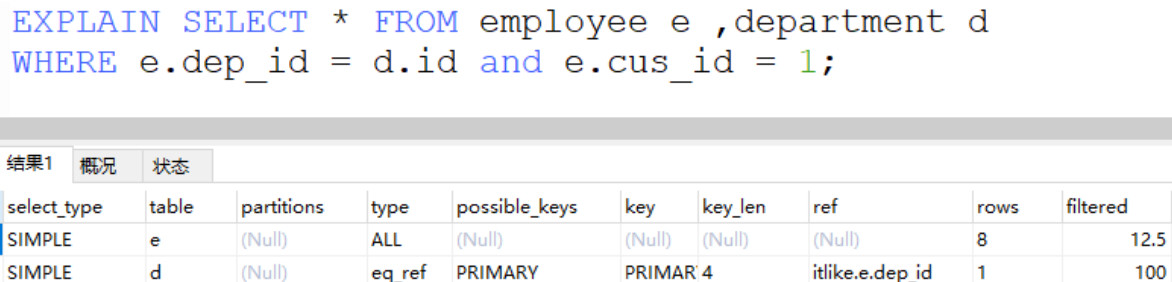
可通过该列计算查询中使用的索引长度 .

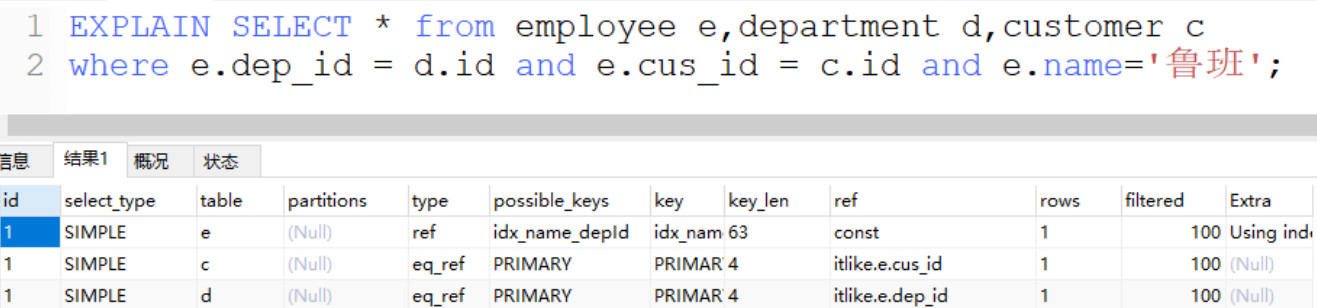
示例:

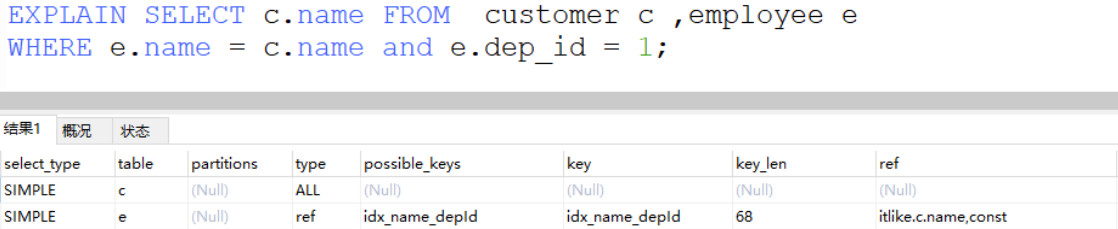


**(9).索引是否被引入到, 到底引用到了哪几个索引**

示例:





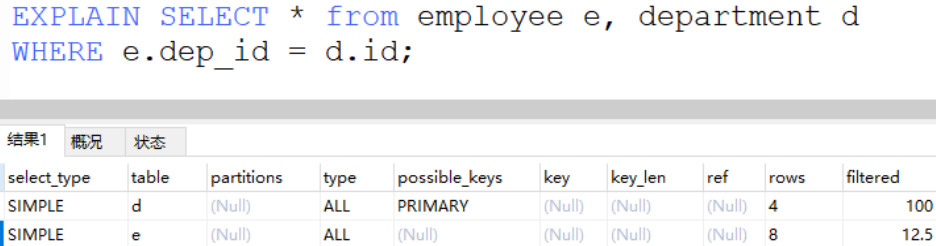


**10.rows:根据表统计信息及索引选用情况**

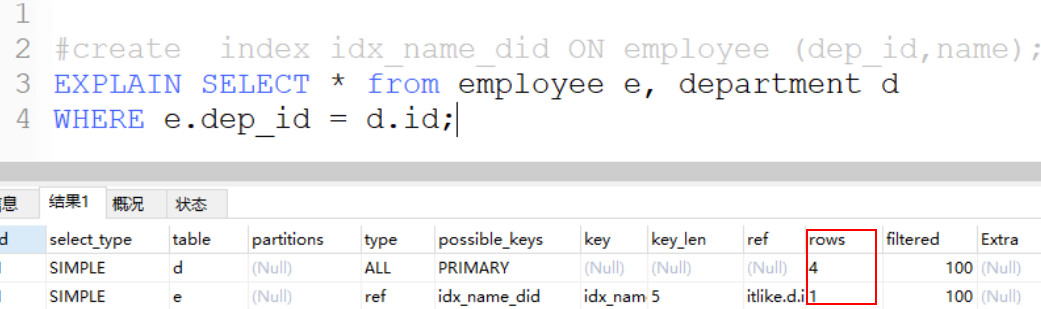
大致估算出找到所需的记录所需要读取的行数

每长表有多少行被优化器查询过

没有建立索引:



建立了索引后:



**11.filtered:**

满足查询的记录数量的比例，

注意是百分比，不是具体记录数

值越大越好，filtered列的值依赖统计信息，

并不十分准确

**12.Extra:额外信息**

**Using filesort:**

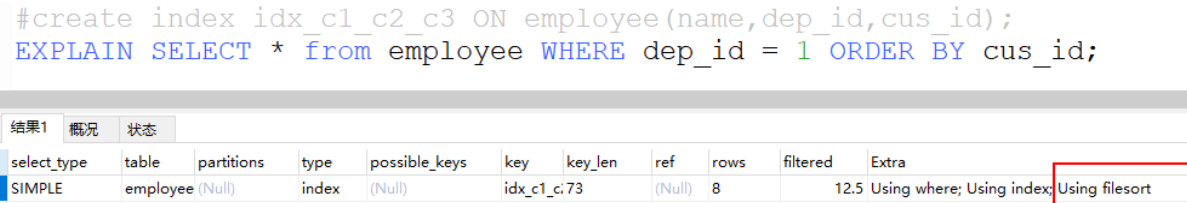
说明mysql会对数据使用一个外部的索引排序,

而不是按照表内的索引顺序进行

Mysql中无法利用索引完成排序操作

称为"文件排序"

示例:



**Using temporary:**

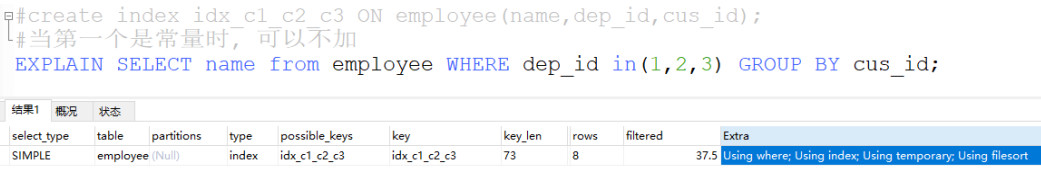
使用了临时表保存中间结果,

Mysql在对查询结果排序时,

使用了临时表,

常见于排序orderby 和分组查询group by

示例:



**use index:**

表示相应的select中使用了覆盖索引,

避免访问了表的数据行, 效率很好

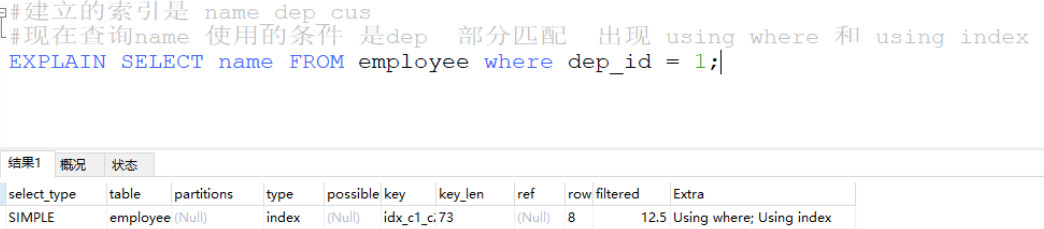
如果同时出现using where

表明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

示例:



**using where:**

表明使用了wher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