# MySQL优化概述

设计： 存储引擎，字段类型，范式

功能： 索引，缓存，分区。

架构： 主从复制，读写分离，负载均衡。

合理SQL： 测试，经验。

# 存储引擎

Create table tableName () engine=myisam|innodb;

一种用来存储MySQL中对象（记录和索引）的一种特定的结构（文件结构）

存储引擎，处于MySQL服务器的最底层，直接存储数据。导致上层的操作，依赖于存储引擎的选择。

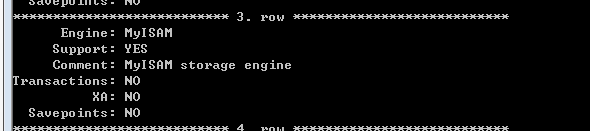


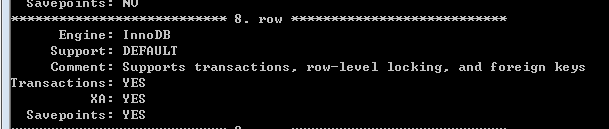
Tip：存储引擎就是特定的数据存储格式（方案）

## Show engines

查看当前MySQL支持的存储引擎列表







## Innodb

>=5.5 默认的存储引擎，MySQL推荐使用的存储引擎。

提供事务，行级锁定，外键约束的存储引擎。

事务安全型存储引擎。更加注重数据的完整性和安全性。

存储格式

数据，索引集中存储，存储于同一个表空间文件中。

数据（记录行）

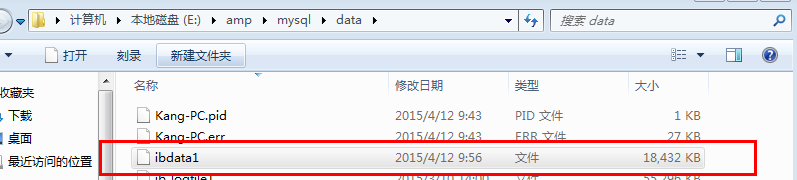
索引（一种检索机制，也需要一定的空间）

创建innodb表后，存在文件如下：

.frm 表结构文件。



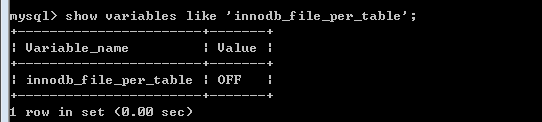
Innodb表空间文件：innodb的数据和索引。



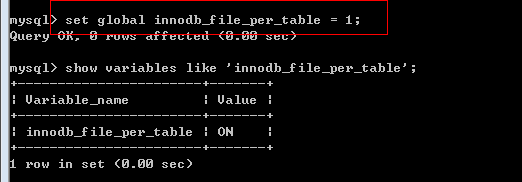
该位置，可以被配置的。

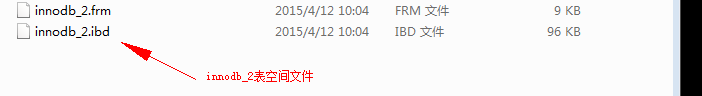
默认，所有的innodb表的表空间文件，都在同一个空间中。

通过配置，达到每张innodb表，一个表空间文件的目的：

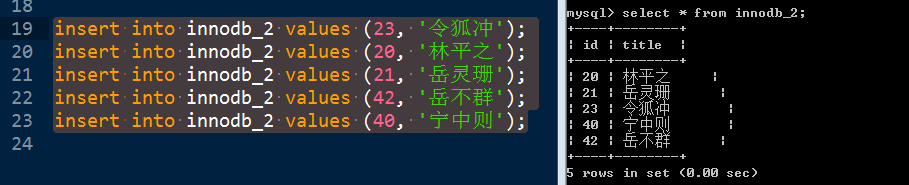


开启该配置：





数据按照主键顺序存储



插入时做排序工作，效率低。

特定功能

事务：

外键约束：

维护数据完整性。

并发性处理：

擅长处理并发的。

行级锁定： row-level locking，实现了行级锁定，在一定情况下，可以选择行级锁来提升并发性。也支持表级锁定，innodb根据操作选择。

多版本并发控制, MVCC，效果达到无阻塞读操作。

## MyISAM

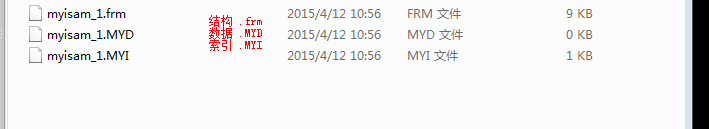
<= 5.5 MySQL默认的存储引擎。

ISAM：Indexed Sequential Access Method(索引顺序[存取方法](http://baike.baidu.com/view/3401818.htm" \t "_blank))的缩写，是一种文件系统。

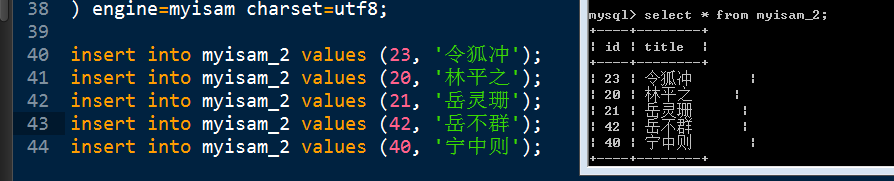
擅长与处理 高速读与写。

存储方式

数据索引分别存储于不同的文件中。



数据的存储顺序为插入顺序



插入速度快，空间占用量小。

功能

全文索引支持。（>=5.6 innodb 支持）

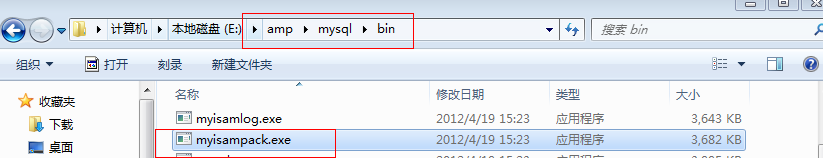
数据的压缩存储。.MYD文件的压缩存储。

压缩前：



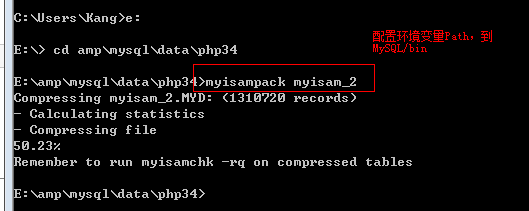


**压缩：**工具 myisamPack完成 压缩功能：

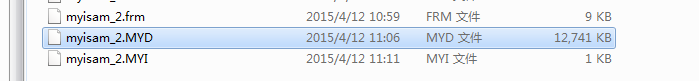


进入到 需要压缩表的数据目录：

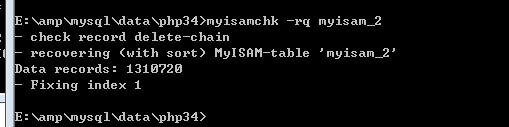
执行压缩指令 myisampack 表名

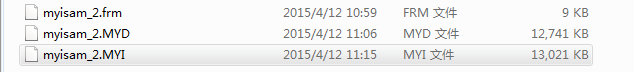


结果：



注意，压缩后，需要重新修复索引：





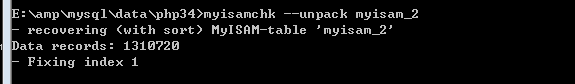
Tip：压缩优势：节省磁盘空间，减少磁盘IO开销。

特点：压缩表为只读表。

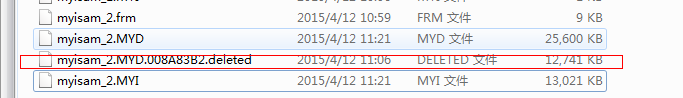


如果需要更新，则需要解压后更新，再压缩（重新索引）：

利用工具：myisamchk –unpack 表名



结果



Flush table myisam\_2



并发性：

仅仅支持表级锁定。

支持 并发插入。写操作中的插入操作，不会阻塞读操作（其他操作）

Innodb PK myisam：

数据完整性，并发性处理，擅长更新，删除。

高速查询及插入。擅长 插入，查询。

## Archive

存档型

仅提供 插入和查询操作。非常高效 无阻塞的插入和查询。

## Memory

内存型

数据存储于内存中，存储引擎。缓存型存储引擎。

## 插件式存储引擎

# 锁

当客户端操作表（记录）时，为了保证操作的隔离性（多个客户端操作不能互相影响），通过加锁来处理。

操作方面：

读锁：读操作时增加的锁，也叫共享锁，S-lock。特征是 阻塞其他客户端的写操作，不阻塞读操作。

写锁：写操作时增加的锁，也叫独占锁或排他锁，X-lock。特征，阻塞其他客户端的读，写操作。

锁定粒度（范围）：

行级：提升并发性，锁本身开销大

表级：不利于并发性，锁本身开销小。

# 类型选择

满足需求。

原则：

## 尽可能小（占用存储空间少）

Tinyint, smallint, mediumint,int, bigint

Varchar(N) varchar(M)

Datetime, timestamp

## 尽可能定长（占用存储空间固定）

Char,varchar

Decimal（变长）, double(float)(定长)

## 尽可能使用整数

IPV4， int unsigned， varchar(15)

Enum

Set

多用位运算。

# 范式，逆范式

Goods

Goods\_id, goods\_name, cat\_id

Category

Cat\_id, cat\_name,

分类列表查询：

分类ID 分类名称 商品数量

3 计算机 567

Select c.\*, count(g.goods\_id) as goods\_count from category as c left join goods as g c.cat\_id=g.cat\_id group by c.cat\_id;

此时商品数量较大。

重新设计category表：增加存当前分类下商品数量的字段。

Category

Cat\_id, cat\_name, goods\_count

每当商品改动时，修改对应分类的数量信息。

再查询分类列表时：

Select \* from category;

此时额外的消耗，出现在维护该字段的正确性上，保证商品的任何更新都正确的处理该数量才可以。

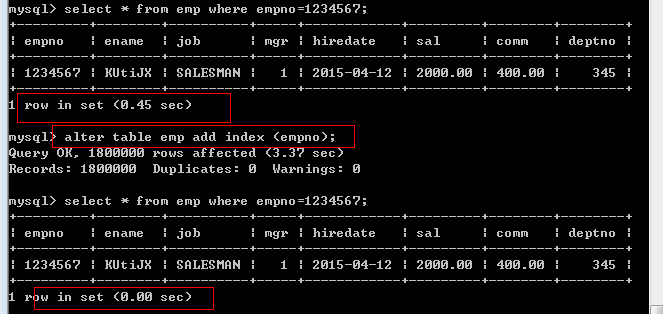
# 索引的使用

利用关键字，就是记录的部分数据（某个字段，某些字段，某个字段的一部分），建立与记录位置的对应关系，就是索引。

索引的关键字一定是排序的。



测试查询，添加索引前后比对执行时间：



## 索引的类型

4种类型：

主索引，唯一索引，普通索引，全文索引。

无论任何类型，都是通过建立关键字与位置的对应关系来实现的。

以上类型的差异：对索引关键字的要求不同。

关键字：记录的部分数据（某个字段，某些字段，某个字段的一部分）。

普通索引,index： 对关键字没有要求。

唯一索引,unique index： 要求关键字不能重复。同时增加唯一约束。

主键索引,primary key： 要求关键字不能重复，也不能为NULL。同时增加主键约束。

全文索引,fulltext key： 关键字的来源不是所有字段的数据，而是从字段中提取的特别关键词。

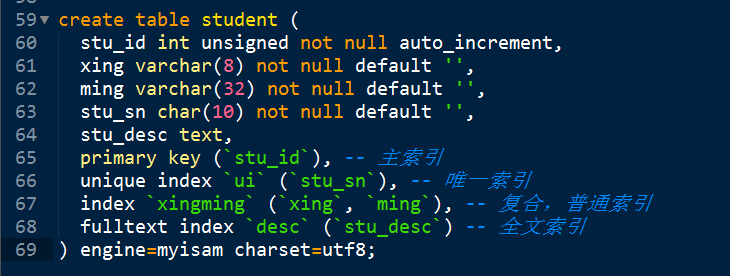
关键字的来源：可以是某个字段，也可以是某些字段。如果一个索引通过在多个字段上提取的关键字，称之为 复合索引。

alter table emp add index (field1, field2);

## 管理索引的语法

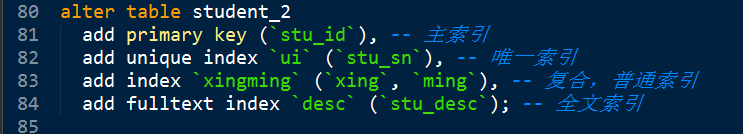
### 创建

建表时



TiP；索引可以起名字，但是主索引不能起名字，因为一个表仅仅可以有一个主索引，其他索引可以出现多个。名字可以省略，mysql会默认生成，通常使用字段名来充当。

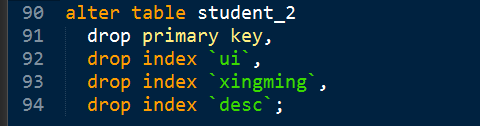
更新表结构



Tip：

1. 如果表中存在数据，数据符合唯一或主键的约束才可能创建成功。
2. Auto\_increment属性，依赖于一个KEY。

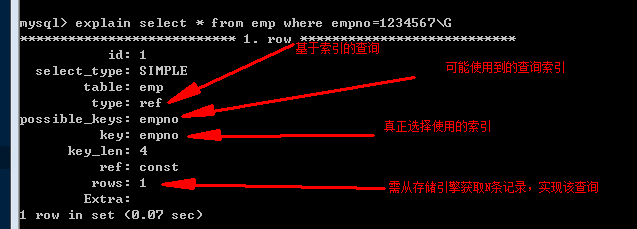
### 删除



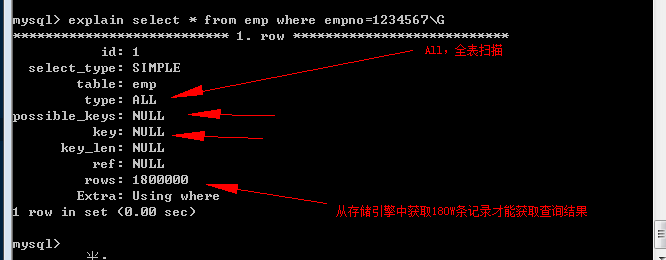
Tip： 别忘了 auto\_increment依赖于KEY

# Explain 执行计划

可以通过在select语句前使用 explain，来获取该查询语句的执行计划，而不是真正执行该语句。



删除索引时，再看执行计划：



Tip：select语句才能获取到执行计划。（新版本会扩展其他语句的执行计划的获取）

# 索引的使用

## 使用场景

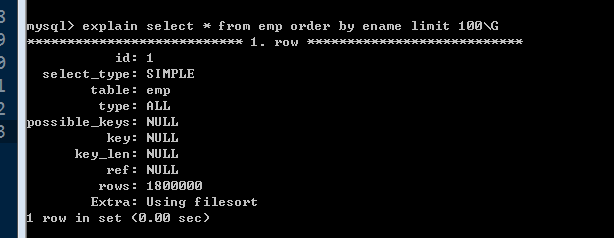
### 索引检索

条件过滤

### 索引排序

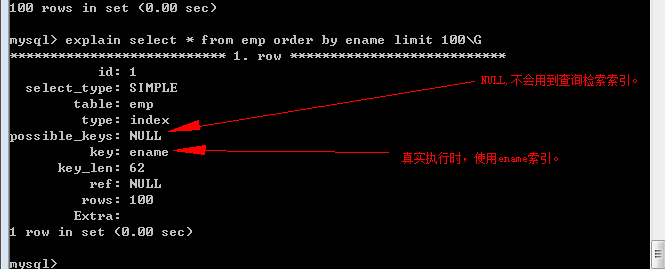
如果order by 排序需要的字段，上存在索引，可能使用到索引。

例如，按照ename字段排序查询：



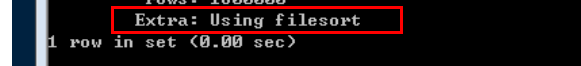
此时，没有任何索引。

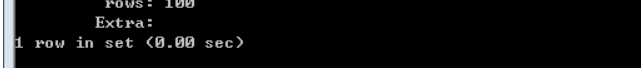
在ename字段上建立索引：



Tip：对比以上两个执行计划：

extra位置：





其中：extra额外信息。

Using filesort，表示使用文件排序（外部排序，内存外部）。

### 索引覆盖

索引拥有的关键字内容，覆盖了查询所需要的全部数据，此时，就不需要在数据区获取数据，仅仅在索引区即可。

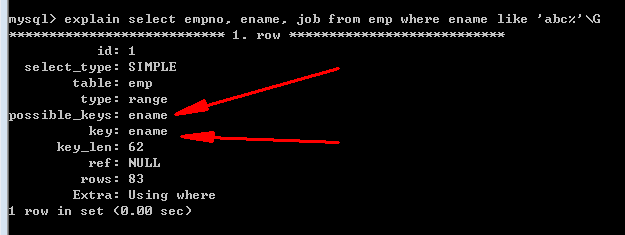
例如，利用名字检索：



可以在ename字段建立索引：



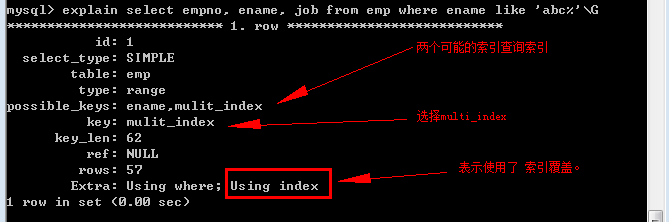
分析执行：



再增加一个索引：

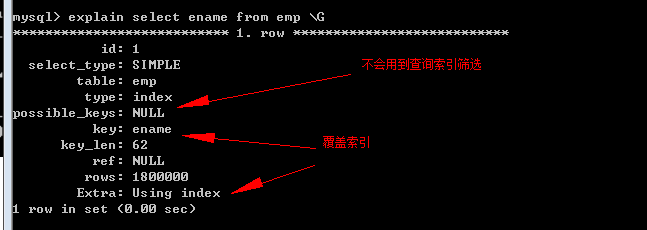


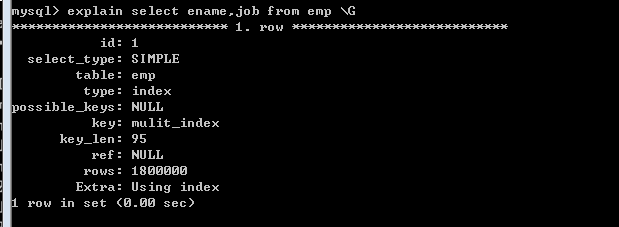
完成相同的查询：



再例如：

说明，不是非要查询用到，才可以索引覆盖，只要满足要求都可以覆盖！





直到索引使用场景时：

建立索引索引时，不要仅仅考虑where检索了吧，同时考虑其他的使用场景。

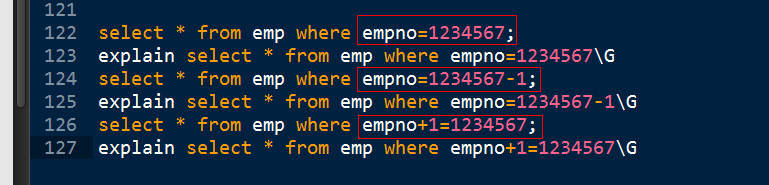
（在所有的where字段上增加索引，就是不合理的）

## 使用原则

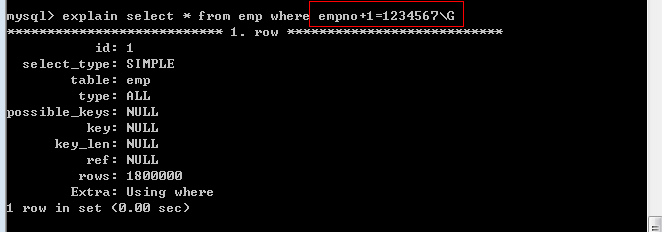
索引存在，没有满足使用原则，导致索引无效：

### 列独立

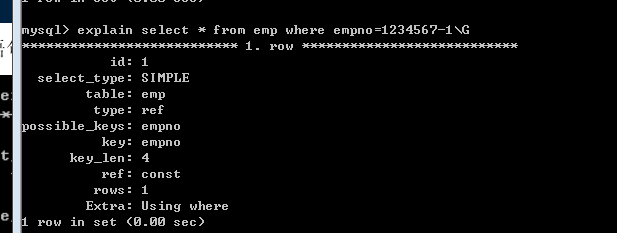
如果需要某个字段上使用索引，则需要在字段参与的表达中，保证字段独立在一侧。



第三个语句 empno-1就不是列独立：就不能用索引。类似函数内等。（write\_time < unix\_timestamp()-$gc\_maxlifetime）

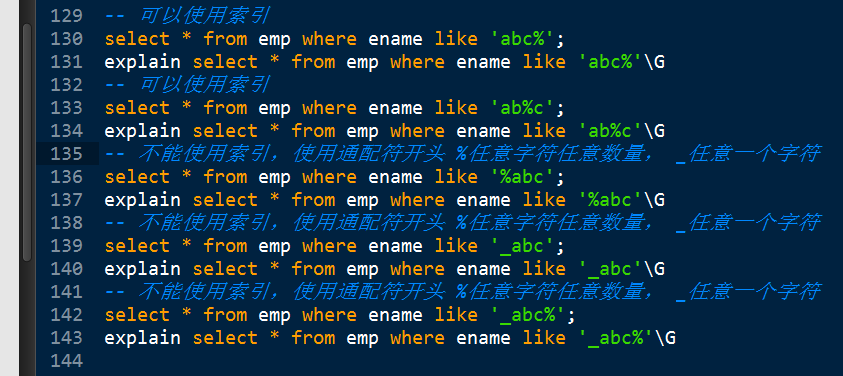


其他两个列独立可以使用：



### 左原则

Like：匹配模式必须要左边确定不能以通配符开头。

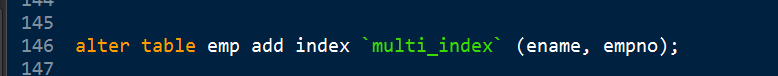


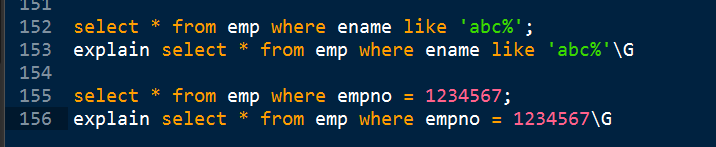
业务逻辑上出现: field like ‘%keywork%’;类似查询，需要使用全文索引。

复合索引：

一个索引关联多个字段。

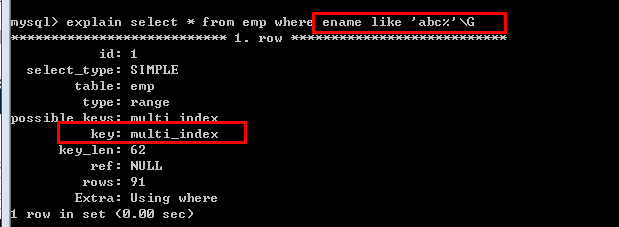
仅仅针对左边字段有效果。



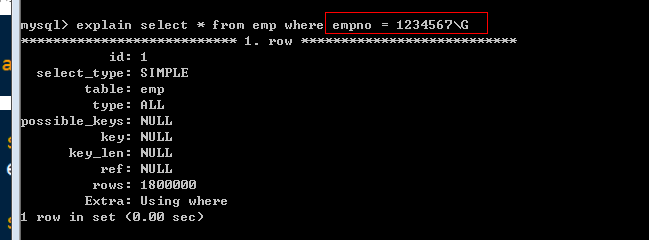


结果：

Ename的查询，使用了索引：

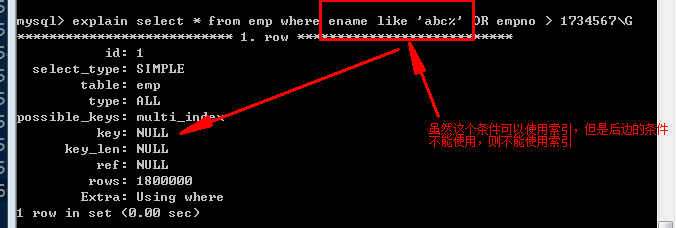


Empno的查询没有使用索引：

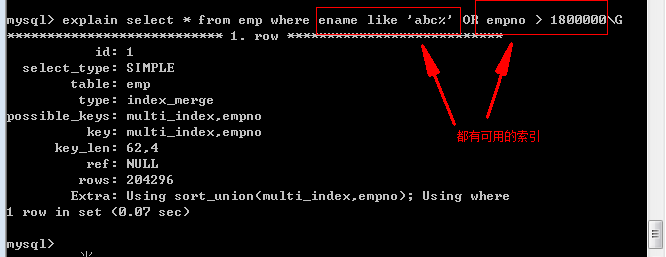


### OR的使用

必须要保证 OR 两端的条件都存在可以用的索引，该查询才可以使用索引。

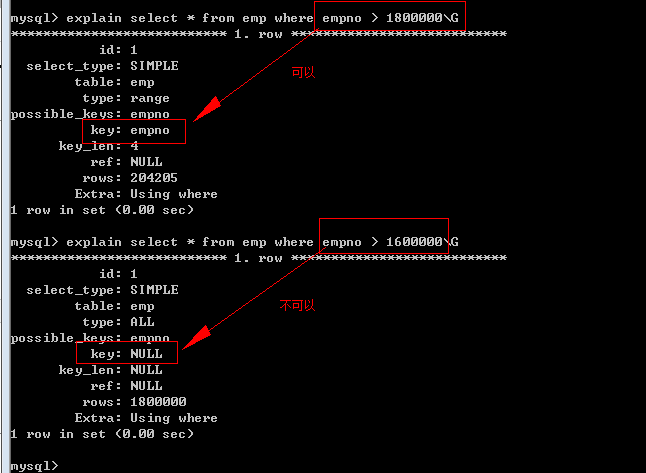


为后面的条件增加可以使用的索引：



### MySQL智能选择

即使满足了上面说原则，MySQL也能弃用索引：



弃用索引的主要原因：

查询即使使用索引，会导致出现大量的 随机IO，相对于从数据记录的第一条遍历到最后一条的顺序IO开销，还要大。

目的：建立索引时，建立满足使用原则的字段上。