Spring 的Transactional注解，默认使用数据源的隔离级别。  
Mysql是 可重复读（Read Repeatable，避免了脏读和不可重复读，可能出现幻读）  
Oracle和SqlServer是 读已提交（Read Committed，避免了脏读，可能出现不可重复读）

1. 查看全局隔离级别

|  |
| --- |
| show global variables like '%isolation%' |

1. 设置全局隔离级别

|  |
| --- |
| set global transaction\_isolation ='read-committed'; |

1. 查看会话隔离级别

|  |
| --- |
| show session variables like '%isolation%' |

1. 设置会话隔离级别

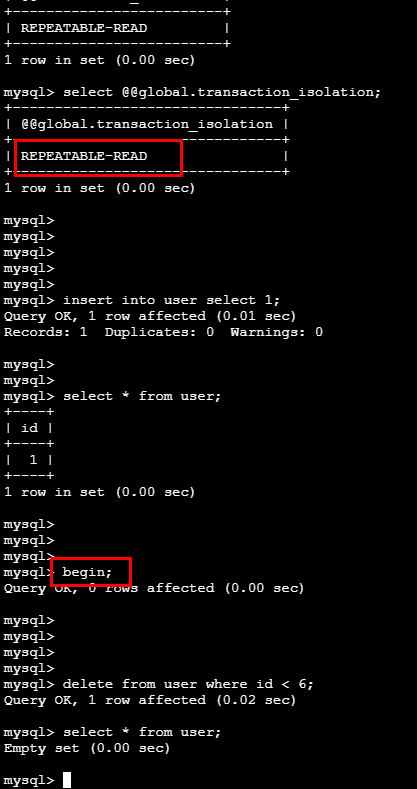
|  |
| --- |
| set session transaction\_isolation ='read-committed'; |

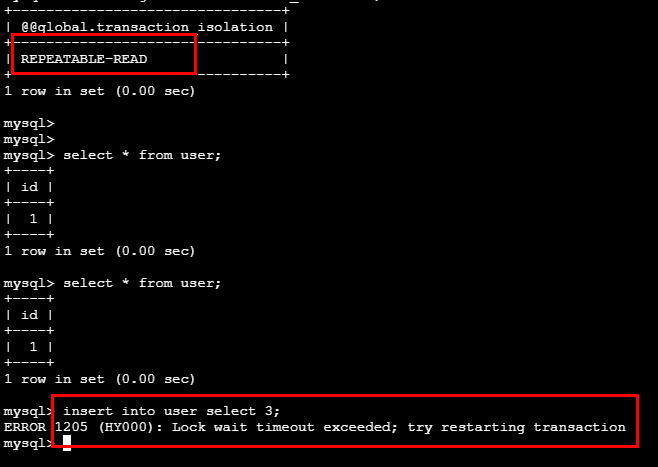
1. 直接通过变量名查看

|  |
| --- |
| select @@global.transaction\_isolation,session.transaction\_isolation; |

正常情况下，可重复读可能会出现了幻读问题，但是Mysql的可重复读通过**间隙锁**解决了幻读，

开启了可重复读，一个线程的事务没有完成时，别的连接的数据库insert或者delete会阻塞。





Mysql使用读已提交隔离级别和binlog使用statement时会出现主从数据不一致问题

binlog的记录顺序是按照事务commit顺序为序的

简单理解：

1、采用**RR隔离级别**时，A事务未提交时，B事务执行insert会阻塞

2、A在事务中使用for update，B未开启事务，这时B执行insert会阻塞

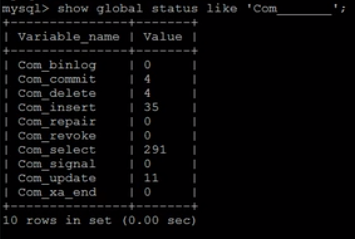
# 使用技巧

1、使用delete|update order by xxx limit 1 来删除|更新最旧的那个值

# 优化SQL

## 1、查看执行次数

*show global status like ‘Com\_\_\_\_\_\_\_’;* 查看各种操作的次数



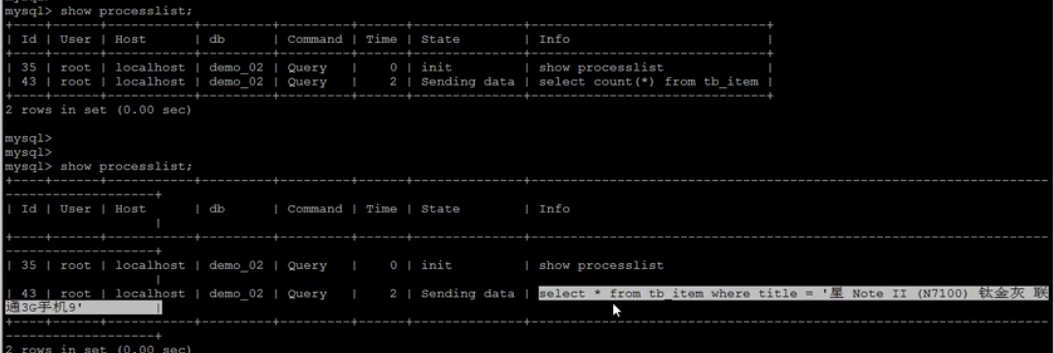
*show global status like ‘innodb\_rows\_%’;* 查看innodb表操作的行数



## 2、定位低效sql

1、慢查询日志

2、*show processlist 查看sql执行的时间*

**

每条数据是一个连接，time表示操作的时间（秒），state表示这条语句执行的状态（可能会有 copying to tmp table，sorting result，sending data）等过程。

## EXPLAIN命令分析SQL执行计划

### 1、select\_type

**SIMPLE**：简单的select查询，查询中不包括子查询或者UNION查询

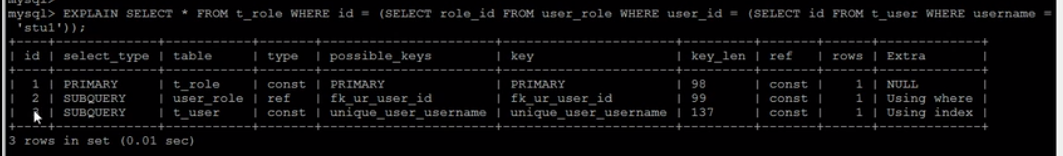
**PRIMARY**：查询中若包含任何复杂的子查询，最外层查询标记为该标记

**SUBQUERY**：在SELECT、WHERE条件中包含的SELECT子查询

**DERIVED**：在FROM中包含的SELECT子查询

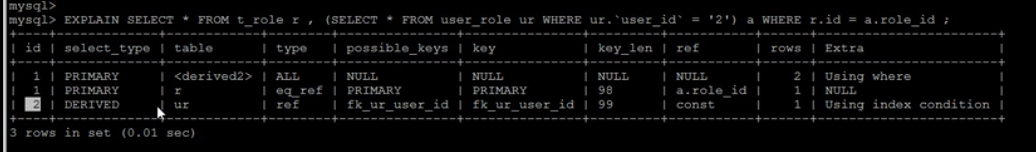
**UNION**：在UNION之后包含的SELECT子查询；若UNION包含在FROM子句的子查询中，外层SELECT将被标记为DERIVED

**UNION RESULT**：从UNION表中获取结果的SELECT



这里的ref都是const，因为这里面的id=，user\_id=，username=，等号右边的都是确定的值

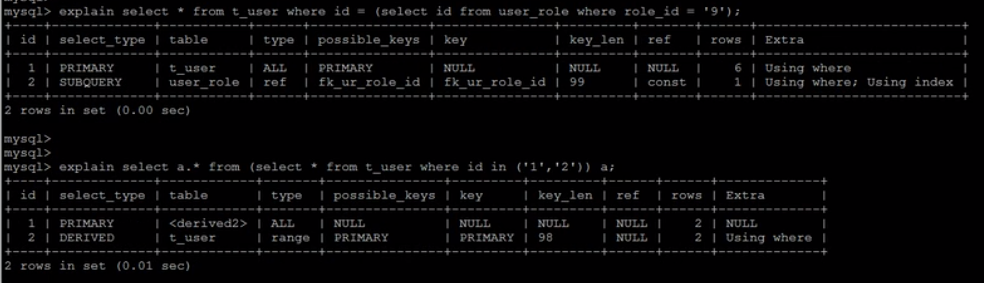
并且由于子查询的select是在where后面，所以select\_type是SUBQUERY（如果子查询在select后面的话也是SUBQUERY）

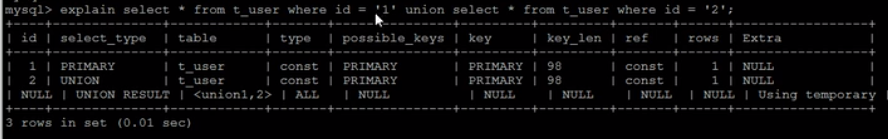


第三行：id为2的子查询【select \* from user\_role ur where ur.user\_id = 2】在from之后（select\_type=DERIVED），user\_id命中非聚簇索引（type=ref，因为非聚簇索引命中结果为多行），且user\_id=2，2为常量（ref=const）

第一行：id为1的临时表查询【select \* from a】，id为2的语句是它的子查询（select\_type=PRIMARY，table=<derived2>），且是全表查询（type=ALL）。**[Using where不理解，t\_role 和 a表的查询顺序是否和行数有关？]**

第二行：id为1的查询【select \* from t\_role r where r.id = a.role\_id】，r.id是主键索引（type=eq\_ref，因为主键索引命中结果为一行），并且where条件中r.id=a.role\_id（ref=a.role\_id）.



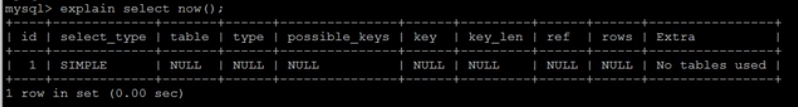


### table

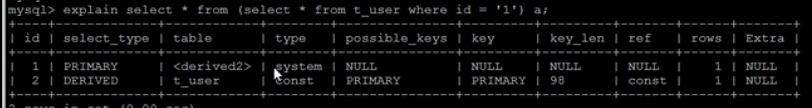
table表示当前查询来此哪张表，当是<derived2>时，表示这次查询的表示是一个from语句后的子查询的衍生表，derived后面的数据是几表示衍生自ID为几的子查询

### type

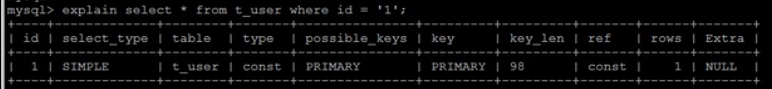
**null**：不访问任何表、索引，直接返回结果。EXPLAIN SELECT new();



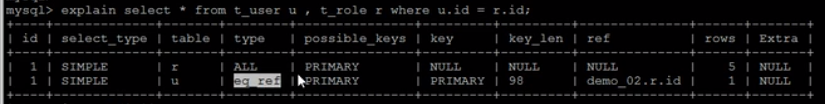
**system**：表中这有一条数据（等于系统表），是const类型的特例，一般不会出现



**const**：针对主键索引、唯一索引，且where条件中**a.id=3**，=右边是一个常量，不是其他表的列

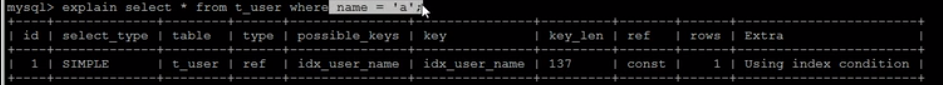


**eq\_ref**：针对主键索引、唯一索引，使用关联查询**a.id=b.user\_id**，查出的结果为一条（因为是主键、唯一索引）。这时会选择全表扫描行数少的表（这里是t\_role）。

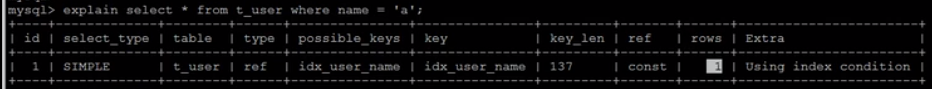


**ref：**针对非主键、非唯一索引，使用**a.name=’a’，**查出的结果为多条。

*？？对非唯一索引关联查询时a.name=b.id时会怎么样？？*

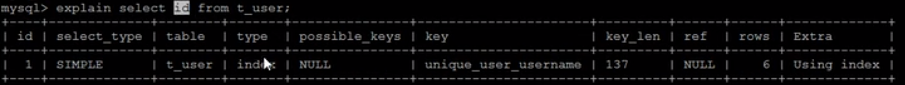


？？name是唯一索引，name=’a’为什么type=ref，应该是const啊？？

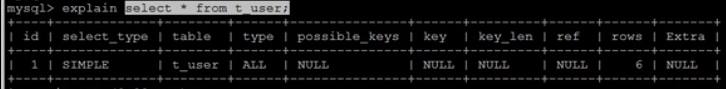


**range：**where之后对索引列进行between、<、>、in操作，如果选择的范围包含了所有的index值，就退化成了**index**

**index：**扫描所有的索引，遍历了整个索引树。



**all：**全表扫描



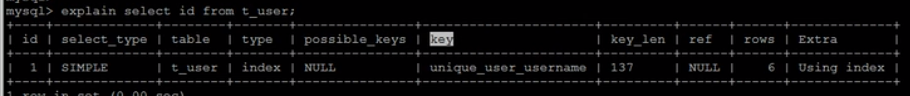
### key

possible\_key：可能使用到的索引

key：实际使用的索引，null表示没有使用索引

key\_len：索引字段的最大可能长度，并非实际使用的长度，在不损失精度的情况下，长度越小越好

可能出现的情况：possible\_key为null，key不为null。因为这里的t\_user表只有两列，且name列有索引，这是直接查name列的索引即可返回id列（产生了覆盖索引）



### rows

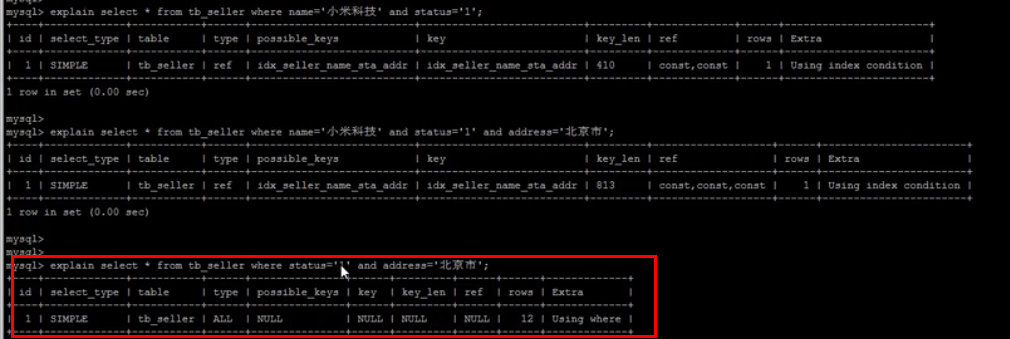
扫描行的数量

### extra

额外的执行计划信息

**using where**：先查索引树，获得id，然后查聚簇索引树（一般就是主键ID的索引树）

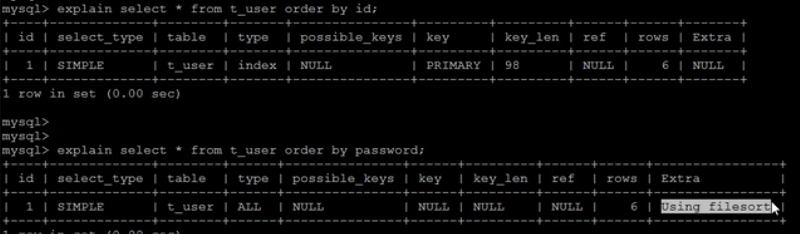
？？这里where查询不满足最左匹配的索引，为什么extra是using where？？



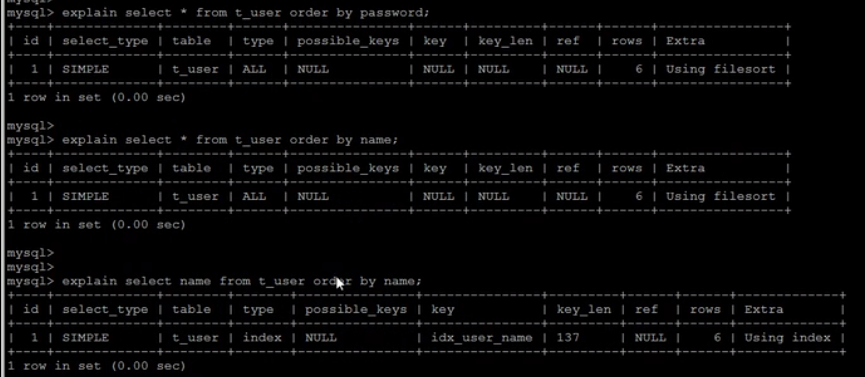
**using index**：查询到覆盖索引

**using filesort**：使用了外部的索引排序，而不是按照表内的索引顺序进行读取，称为“文件排序”

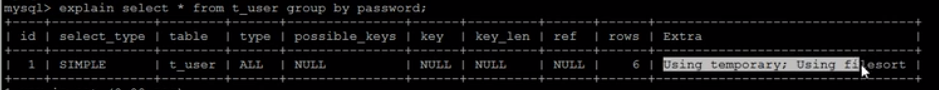
按照id排序没有出现using filesort，因为id上有索引。但是使用password排序就出现了using filesort，因为password上面没有索引。



？？name上面有索引，但是为什么select \* from t\_user order by name是using filesort，但是select name from t\_user order by name就不是了？？



**using temporary**：使用了临时表，常见于order by、group by



## show profile命令分析SQL执行计划

1. 当前版本查看是否支持

**select @@have\_profiling**

1. 查看是否开启

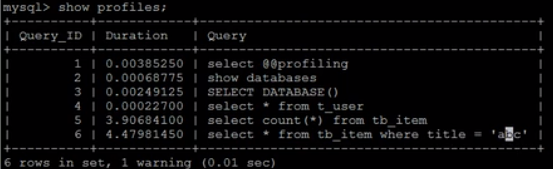
**select @@profiling**

1. 开启

**set profiling = 1;**

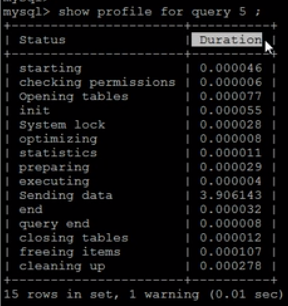
1. 查询所有语句的耗时

**show profile;**



1. 查看某条sql执行过程中每个线程的状态和耗时

show profile [all cpu block io context switch page faults] for query id



Starting：开始

Checking permissions：鉴权

Opening table：打开表

Init：初始化

System\_lock：系统锁

Optimizing：优化器

Statistics：统计

Preparing：准备

Executing：执行

**Sending\_data：MySQL线程开始访问数据行（大量磁盘IO操作）并把结果返回给客户端的时间，而不仅仅是返回给客户端的时间**

Query\_end：查询结束

End：结束

Closing table：关闭表

Freeing items：

Clean up：

## trace命令分析SQL执行计划

通过trace指令了解为什么**优化器**选择A计划而不是B计划

1. 打开trace，设置格式为json，

set optimizer\_trace=”enable=on”,end\_markers\_in\_json=on;

1. 设置trace最大能使用的内存大小，避免解析过程中因为默认内存大小而不能完整展示

set optimizer\_trace\_max\_mem\_sizw=1000000

1. 执行一条查询语句
2. 查看trace结果

select \* from information\_schema.optimizer\_trace\G;

## 索引的使用

### 1、技巧

若a,b,c复合索引

1. 复合索引全值查询
2. 复合索引最左前缀

A 和顺序没有关系，where a=’’ and b=’’ 和 where b=’’ and a=’’都是满足最左前缀的

B where a=’’ and c=’’ 这时只会查a那部分索引，c不会走索引。EXPLAIN中的key\_len显示a的长度

1. 复合索引范围查询的右边用不到索引

where a=’’ and b>’’ and c=’’ 这时只会走**a,b索引**，c不会走索引

1. 索引列使用函数，该列索引失效
2. 字符串不加单引号，该列索引失效（因为监测到类型是字符串后，会隐式转换，使索引失效）
3. 尽量使用覆盖索引，避免select \*
4. 用or分开的条件，如果or左边列有索引，右边列没有索引，那么左边的索引不会生效
5. like模糊匹配时，%不能在左边。？？*如果要使用 %aaa% 来查询，那么可以使用覆盖索引，select a,b,c from tb where c likne ‘%aaa%’？？*
6. 有时候索引比全表扫描慢，如果某个列的值都是重复的，那么索引扫描这个列不如全表扫描*？？和选择度类似？？*
7. **is null，is not null有时会索引失效**

如果某个索引列的值绝大部分都是null，那么用where a is null查询则不走索引直接全表查询。is not null同理。*？？和选择度类似？？*

1. in走索引，not in索引失效
2. 尽量使用复合索引

A a，b，c分别建索引，where a=’’ and b=’’ and c=’’是只会命中一个最优的索引（选择度最高的，或者理解为重复数据少的）

### 查看索引使用情况

show [global] status like ‘Handler\_read%’