## 一. 插入大量数据（1亿条）

Mybatis插入：

单次insert插入10000条数据：

总时间：321072毫秒

总时间：321秒

批量（1000）插入10000：

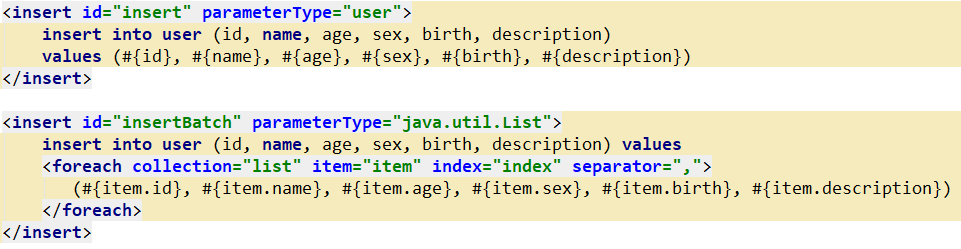
总时间：1257毫秒

总时间：1秒

批量（20000）插入10000000：

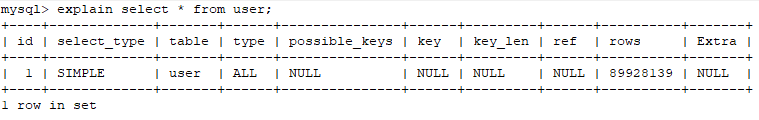
总时间：334662毫秒

总时间：334秒

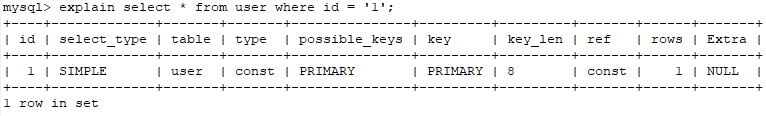


## 二. 对插入的数据进行查询—之索引优化

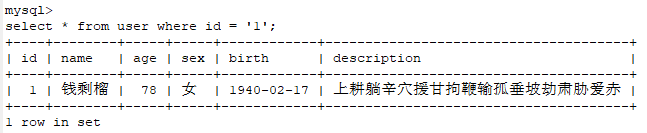
1. 首先进行全表查询发现全表扫描



2. 对主键进行查询发现是const（常量）

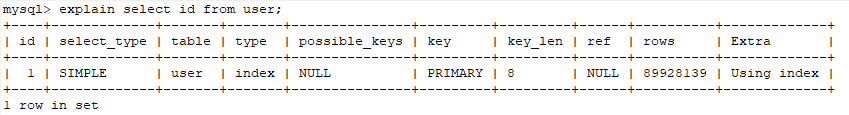


而且使用主键查询一瞬间就可以查出数据



全表查询直接卡死（如果不对数据库优化后果很严重，例如需要分区分表分库等）

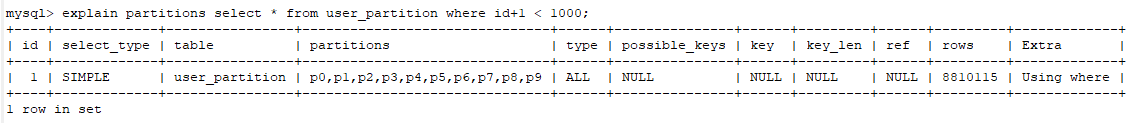
3. 单独查询主键列



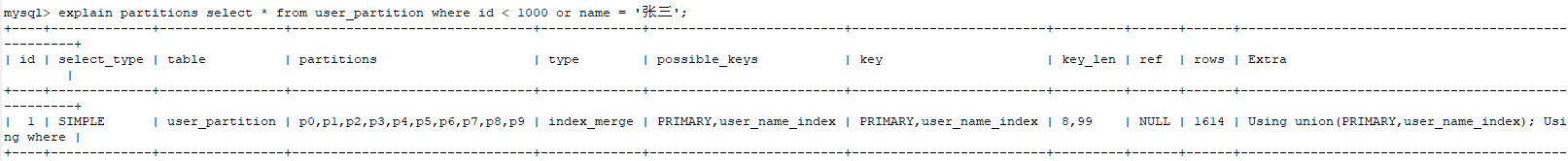
发现是使用了index（索引）

4. 然后对表进行任何的修改都会卡死，而且数据库服务过高，电脑始终很卡，所以这里直接删除该表，然后新建一个表并进行分区，同时关联一个部门表，新加薪资字段comm，与salary，同时插入五千万条数据，每个分区一千万条数据。

5. 首先，始终将索引单独放在比较符号的一侧，不然就会进行全表扫描



6. 其次，MySQL5.1开始使用索引合并技术，如下：



两个索引列进行or操作不再是全表扫描，而是index\_merge，使用两个索引。

但是，有时候频繁的索引合并一定程度上说明了这个表设计的问题，应该有合理的组合索引。

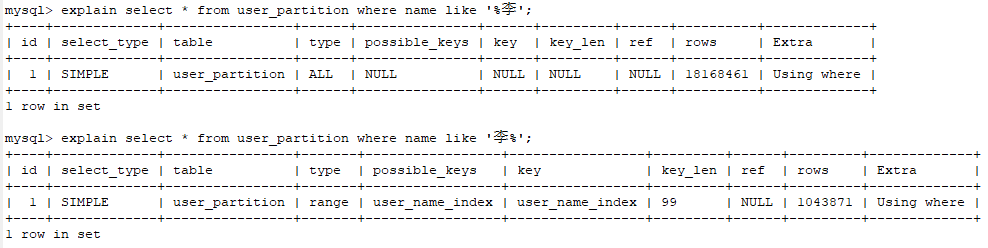
而且，服务器对索引进行合并会耗费大量CPU和内存资源。

另外，这里MySQL之前当对某个索引进行范围条件查询的时候，后面的列不会用到索引，但是这里用到了索引。

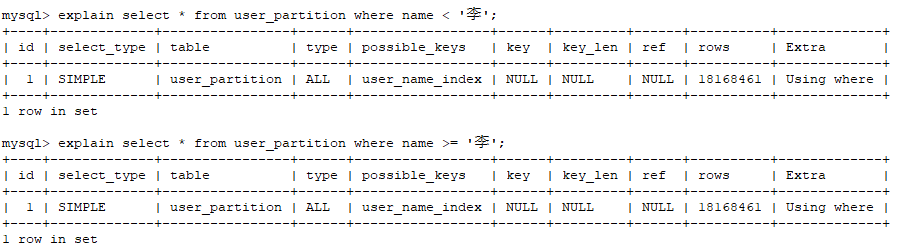


7. 普通索引

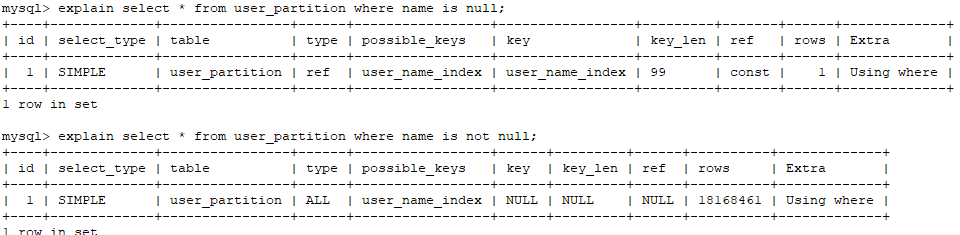
· like 中%开头会使得索引失效



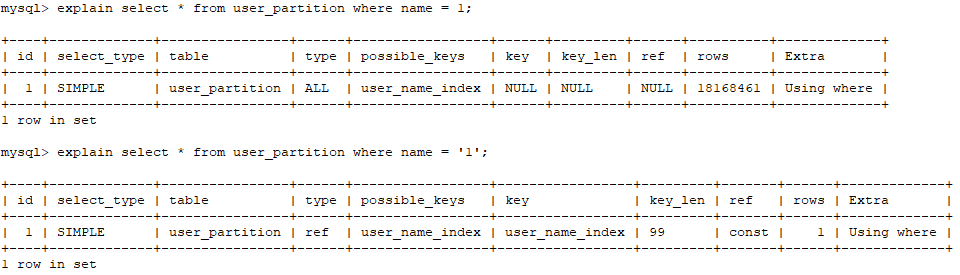
* 字符串的范围查询会使得索引失效



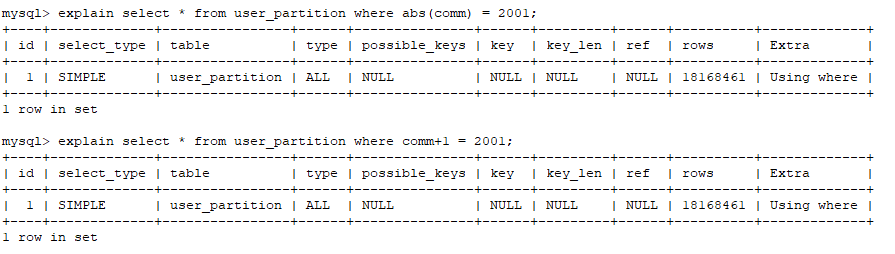
· 使用is not null会使得索引失效



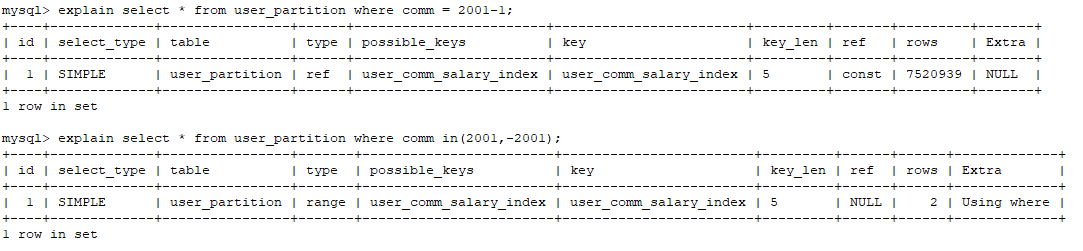
· 把字符串类型的列用’’括起来，否则不会使用索引



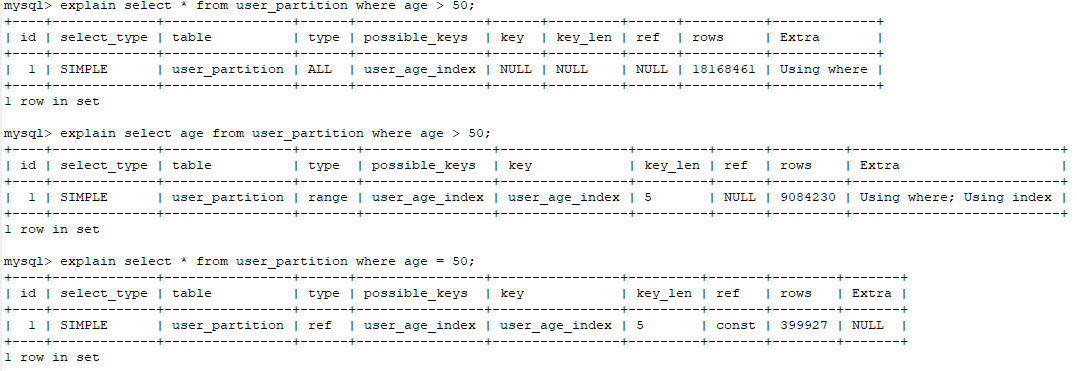
· where表达式中必须把索引列单独放在一侧，否则会使索引失效。例如，以下便不会使用索引。



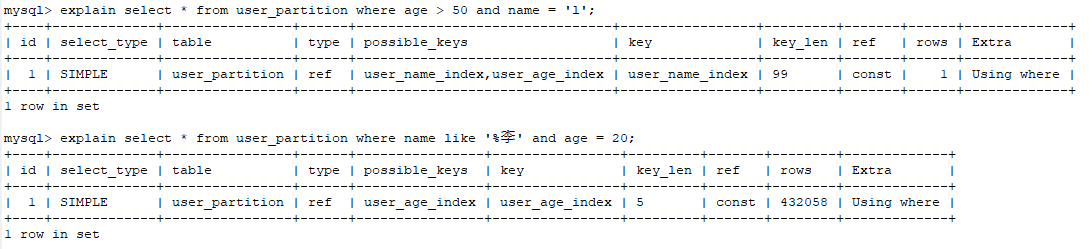
改写如下：



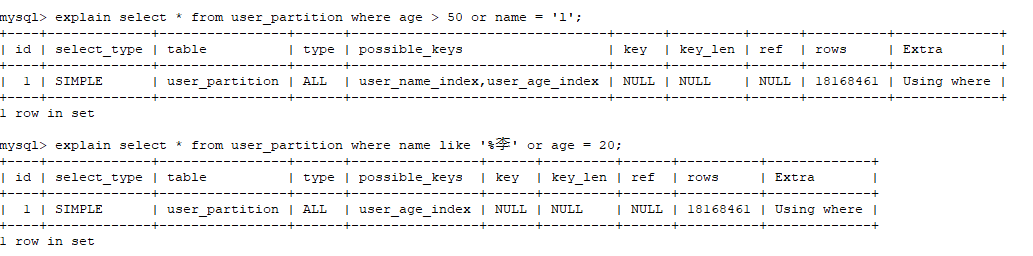
· 某些情况下，select \* 与范围查询会使索引失效



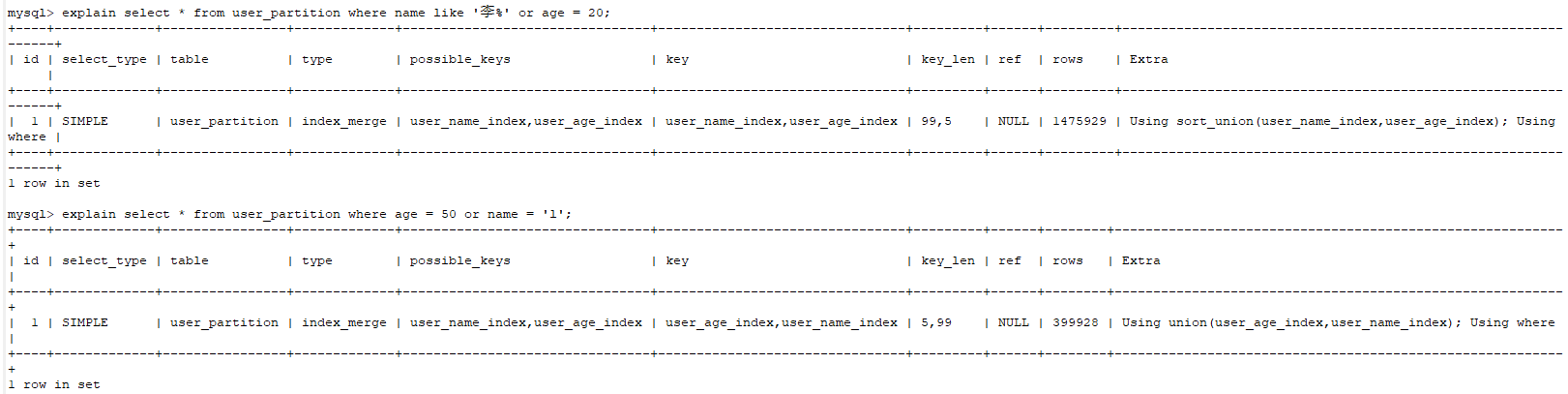
· 不知为何，以下两个范围查询与and都会使用之后的索引



· or中以下范围查询的情况会使得索引失效，字符串的模糊查询以%开头



但是以下情况下会使用索引合并优化



8. 组合索引

这里有一个组合索引（comm，salary），下面分析四种情况下的使用：

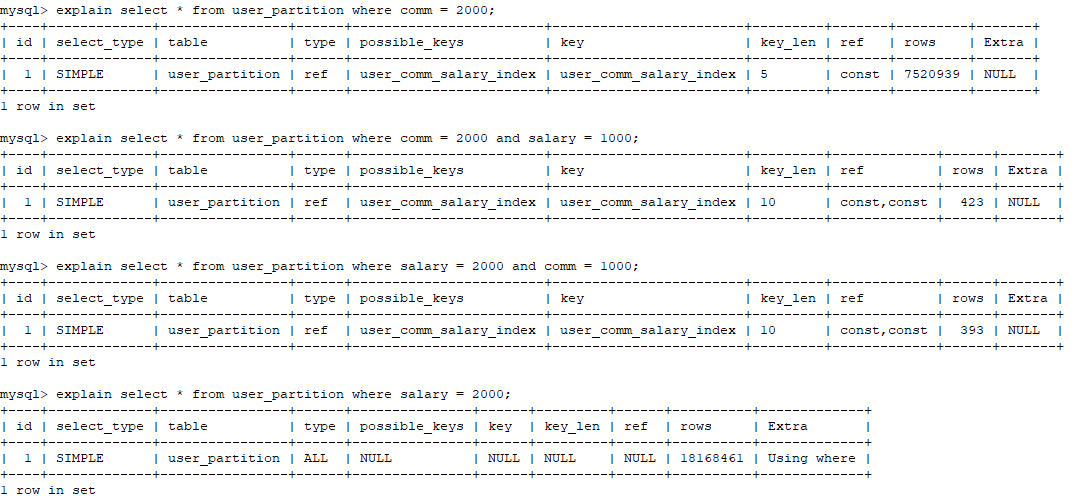
（1）comm

（2）comm，salary

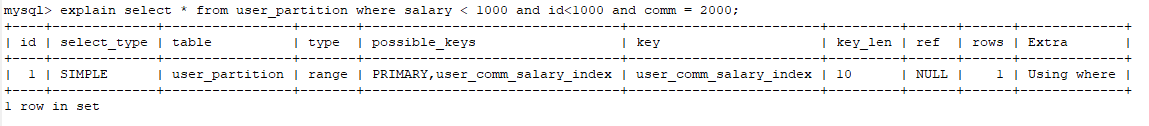
（3）salary，comm

（4）salary

可以看到只有第四种情况没有使用组合索引。

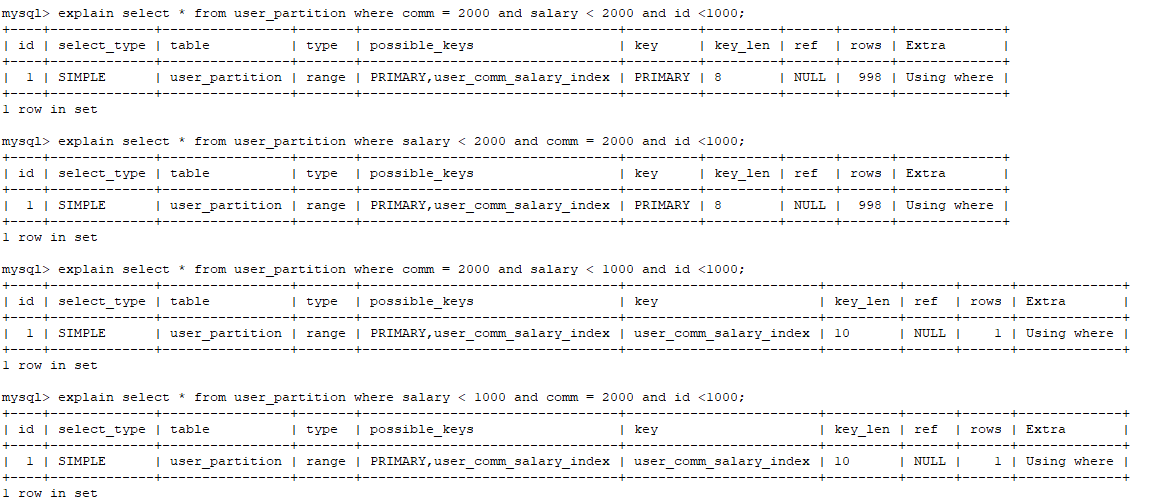


即便这样写依然会用到组合索引

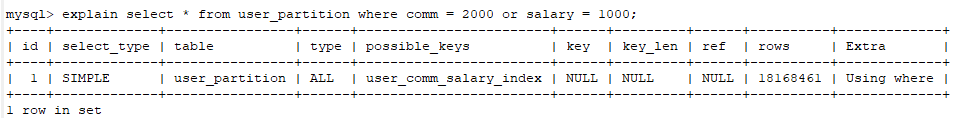


另外使用组合索引应该把列数少的放到前面（定义索引的时候有效）：

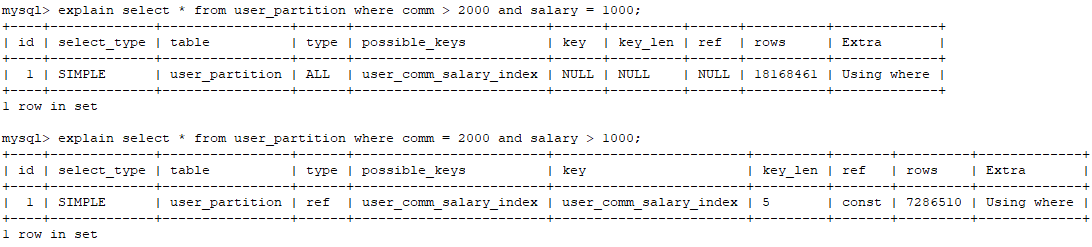
（猜想）以下情况说明mysql会自动优化组合索引的顺序：（comm，salary）等同于（salary，comm）



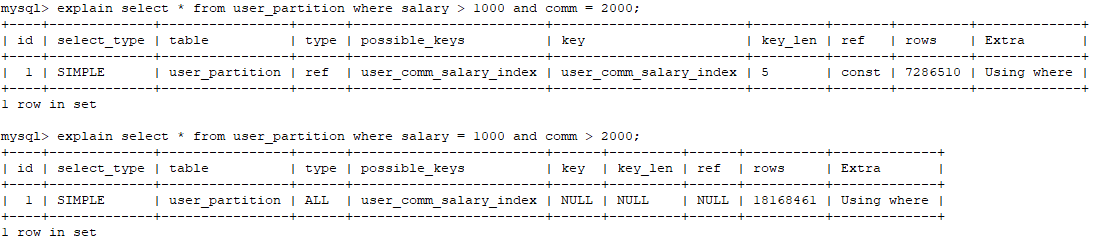
· 组合索引中使用or会使得索引失效



· 组合索引中范围比较在前面不会用到索引

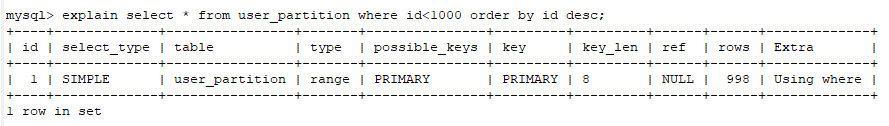


如下例子可以验证，其实where条件中会自动优化组合索引的顺序

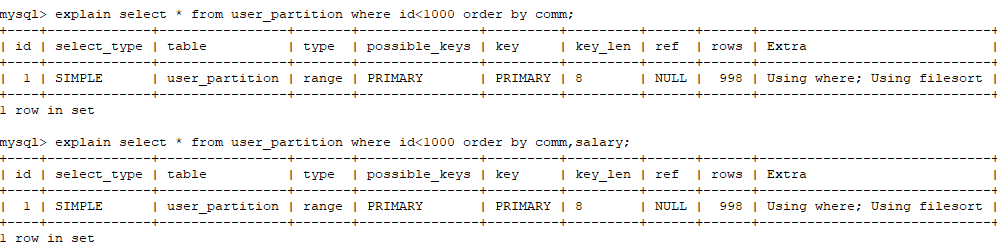


9. 在排序中使用索引

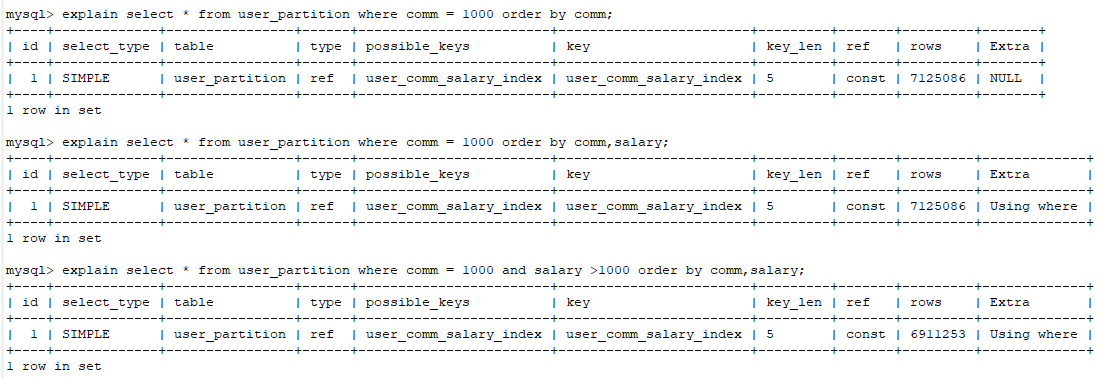
在where语句中使用到的字段在排序时会用到索引



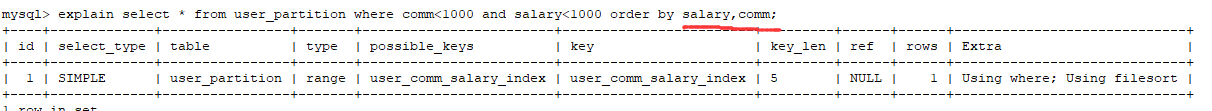
以下情况均不会用到组合索引，可以看到使用了文件排序



以下三种情况可以看到使用了组合索引user\_comm\_salary\_index



但是如果排序的字段顺序不是组合索引的顺序则排序不会使用组合索引，如下：



其它失效情况同组合索引的失效。

注：索引的使用与否最好根据实际情况自行explain判断并且优化。

## 二. 对插入的数据进行查询—之慢查询优化

1. 目的

优化查询其实就是减少和消除：

（1）执行时间很长的操作；

（2）额外多次执行的操作。

2. 性能模式分析（摘自MySQL官方文档）

以下示例演示如何使用性能模式语句事件和阶段事件来检索与由[SHOW PROFILES](https://dev.mysql.com/doc/refman/8.0/en/show-profiles.html" \o "13.7.6.31 SHOW PROFILES语法)和[SHOW PROFILE](https://dev.mysql.com/doc/refman/8.0/en/show-profile.html" \o "13.7.6.30 SHOW PROFILE语法)语句提供的分析信息相当的数据。

该[setup\_actors](https://dev.mysql.com/doc/refman/8.0/en/setup-actors-table.html)表可用于限制主机，用户或帐户对历史事件的收集，以减少运行时开销和历史表中收集的数据量。该示例的第一步显示如何将历史事件的收集限制为特定用户。

性能模式以皮秒（万亿分之一秒）显示事件定时器信息，以将定时数据标准化为标准单元。在以下示例中， TIMER\_WAIT值除以1000000000000以显示以秒为单位的数据。值也被截断为6位小数，以[SHOW PROFILES](https://dev.mysql.com/doc/refman/8.0/en/show-profiles.html" \o "13.7.6.31 SHOW PROFILES语法)与 [SHOW PROFILE](https://dev.mysql.com/doc/refman/8.0/en/show-profile.html)语句相同的格式显示数据。

1. 将历史事件的集合限制为将运行查询的用户。默认情况下， [setup\_actors](https://dev.mysql.com/doc/refman/8.0/en/setup-actors-table.html" \o "26.12.2.1 setup_actors表)配置为允许所有前台线程的监视和历史事件收集：

mysql> SELECT \* FROM performance\_schema.setup\_actors;

+------+------+------+---------+---------+

| HOST | USER | ROLE | ENABLED | HISTORY |

+------+------+------+---------+---------+

| % | % | % | YES | YES |

+------+------+------+---------+---------+

更新[setup\_actors](https://dev.mysql.com/doc/refman/8.0/en/setup-actors-table.html" \o "26.12.2.1 setup_actors表)表中的默认行 以禁用所有前台线程的历史事件收集和监视，并插入一个新行，以便为将运行查询的用户启用监视和历史事件收集：

mysql> UPDATE performance\_schema.setup\_actors

SET ENABLED = 'NO', HISTORY = 'NO'

WHERE HOST = '%' AND USER = '%';

mysql> INSERT INTO performance\_schema.setup\_actors

(HOST,USER,ROLE,ENABLED,HISTORY)

VALUES('localhost','test\_user','%','YES','YES');

[setup\_actors](https://dev.mysql.com/doc/refman/8.0/en/setup-actors-table.html)现在 ，表格中的数据应与以下内容类似：

mysql> SELECT \* FROM performance\_schema.setup\_actors;

+-----------+-----------+------+---------+---------+

| HOST | USER | ROLE | ENABLED | HISTORY |

+-----------+-----------+------+---------+---------+

| % | % | % | NO | NO |

| localhost | test\_user | % | YES | YES |

+-----------+-----------+------+---------+---------+

1. 通过更新[setup\_instruments](https://dev.mysql.com/doc/refman/8.0/en/setup-instruments-table.html" \o "26.12.2.3 setup_instruments表)表确保启用语句和阶段检测 。某些仪器可能已默认启用。

mysql> UPDATE performance\_schema.setup\_instruments

SET ENABLED = 'YES', TIMED = 'YES'

WHERE NAME LIKE '%statement/%';

mysql> UPDATE performance\_schema.setup\_instruments

SET ENABLED = 'YES', TIMED = 'YES'

WHERE NAME LIKE '%stage/%';

1. 确保已启用events\_statements\_\*和 events\_stages\_\*使用者。某些消费者可能已默认启用。

mysql> UPDATE performance\_schema.setup\_consumers

SET ENABLED = 'YES'

WHERE NAME LIKE '%events\_statements\_%';

mysql> UPDATE performance\_schema.setup\_consumers

SET ENABLED = 'YES'

WHERE NAME LIKE '%events\_stages\_%';

1. 在您正在监视的用户帐户下，运行要分析的语句。例如：

mysql> SELECT \* FROM employees.employees WHERE emp\_no = 10001;

+--------+------------+------------+-----------+--------+------------+

| emp\_no | birth\_date | first\_name | last\_name | gender | hire\_date |

+--------+------------+------------+-----------+--------+------------+

| 10001 | 1953-09-02 | Georgi | Facello | M | 1986-06-26 |

+--------+------------+------------+-----------+--------+------------+

1. EVENT\_ID通过查询[events\_statements\_history\_long](https://dev.mysql.com/doc/refman/8.0/en/events-statements-history-long-table.html" \o "26.12.6.3 events_statements_history_long表) 表来 识别语句 。此步骤类似于运行 [SHOW PROFILES](https://dev.mysql.com/doc/refman/8.0/en/show-profiles.html)以识别 Query\_ID。以下查询生成类似于的输出[SHOW PROFILES](https://dev.mysql.com/doc/refman/8.0/en/show-profiles.html" \o "13.7.6.31 SHOW PROFILES语法)：

mysql> SELECT EVENT\_ID, TRUNCATE(TIMER\_WAIT/1000000000000,6) as Duration, SQL\_TEXT

FROM performance\_schema.events\_statements\_history\_long WHERE SQL\_TEXT like '%10001%';

+----------+----------+--------------------------------------------------------+

| event\_id | duration | sql\_text |

+----------+----------+--------------------------------------------------------+

| 31 | 0.028310 | SELECT \* FROM employees.employees WHERE emp\_no = 10001 |

+----------+----------+-------------------------------------------------------+

1. 查询 [events\_stages\_history\_long](https://dev.mysql.com/doc/refman/8.0/en/events-stages-history-long-table.html" \o "26.12.5.3 events_stages_history_long表) 表以检索语句的阶段事件。阶段与使用事件嵌套的语句相关联。每个阶段事件记录都有一个NESTING\_EVENT\_ID包含EVENT\_ID父语句的列。

mysql> SELECT event\_name AS Stage, TRUNCATE(TIMER\_WAIT/1000000000000,6) AS Duration

FROM performance\_schema.events\_stages\_history\_long WHERE NESTING\_EVENT\_ID=31;

+--------------------------------+----------+

| Stage | Duration |

+--------------------------------+----------+

| stage/sql/starting | 0.000080 |

| stage/sql/checking permissions | 0.000005 |

| stage/sql/Opening tables | 0.027759 |

| stage/sql/init | 0.000052 |

| stage/sql/System lock | 0.000009 |

| stage/sql/optimizing | 0.000006 |

| stage/sql/statistics | 0.000082 |

| stage/sql/preparing | 0.000008 |

| stage/sql/executing | 0.000000 |

| stage/sql/Sending data | 0.000017 |

| stage/sql/end | 0.000001 |

| stage/sql/query end | 0.000004 |

| stage/sql/closing tables | 0.000006 |

| stage/sql/freeing items | 0.000272 |

| stage/sql/cleaning up | 0.000001 |

+--------------------------------+----------+

3. 不要向数据库请求不需要的数据

（1）查询不需要的记录，例如返回全部行

（2）查询不需要的列，例如select \*甚至多表关联select \*

（3）重复查询相同的数据

4. 查询的三个指标

· 响应时间

· 扫描的行数

· 返回的行数

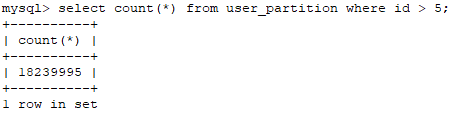
5. 切分查询

6. 分解关联查询

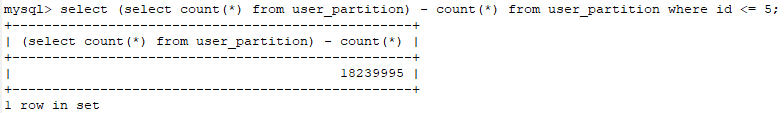
7. 优化count()查询

对于count(\*)，如果一个count(某列)其中某列是代表所有结果集，也就是没有null的数据，那么MySQL会把count(某列)优化为count(\*)。如果想要返回结果集行数，那么最好用count(\*)；

简单的优化：



这样基本扫描了所有的行，所以改写成如下：



对比速度：



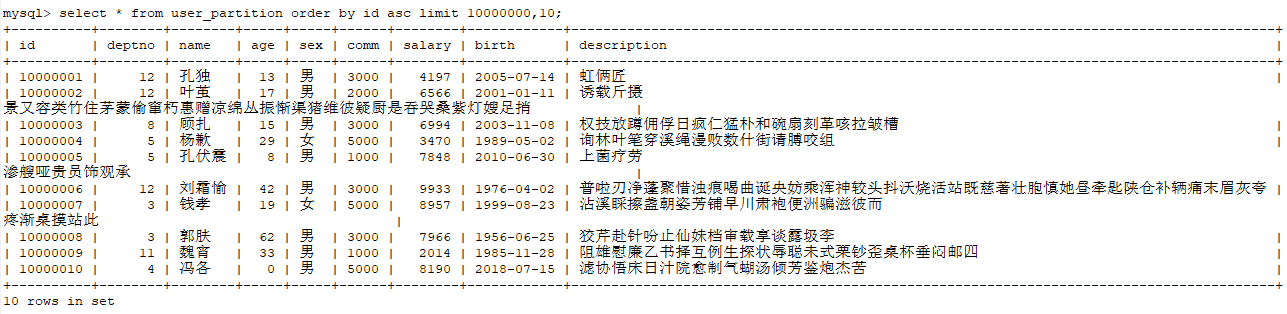
如果是MyISAM引擎估计差距会更大，因为MyISAM引擎会缓存一个数值记录结果集数量；

另一种情况下我们可以使用近似值：

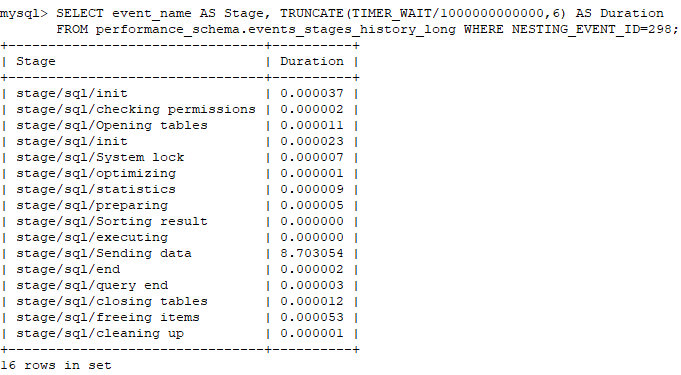
当对精度要求不高的时候可以返回explain的近似值；

8. 优化limit分页查询

正常分页

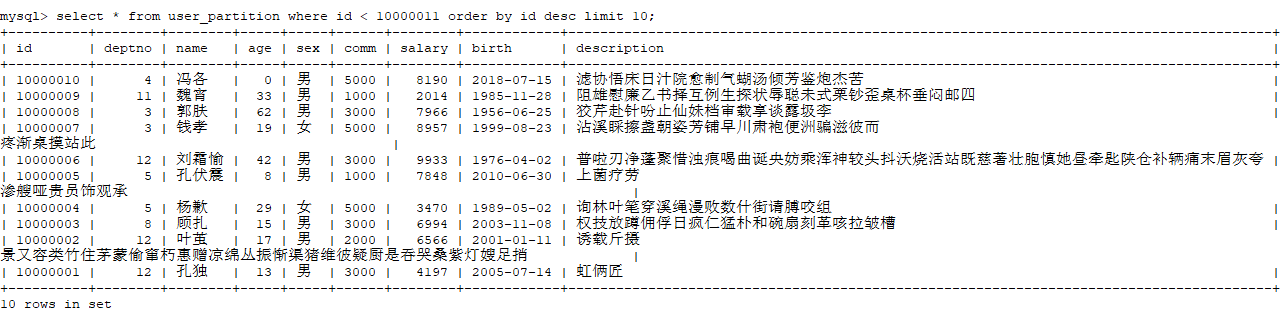


运行的时间：

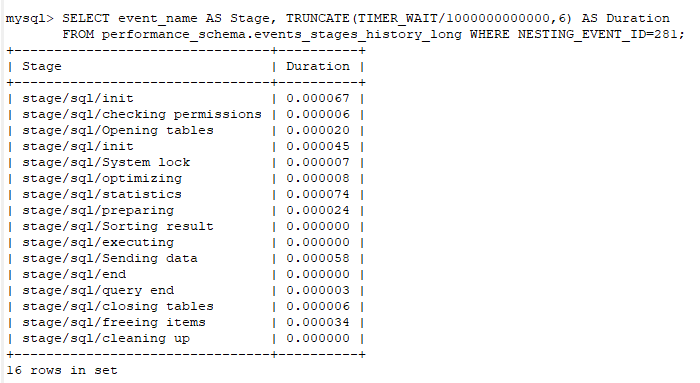


该分页会先查询10000010条记录然后丢弃10000000条记录，所以查询了过多不需要的数据，因此导致查询速度变慢；

优化：



运行的时间：



对比：



缺点：必须有一条连续的offset该例子中的主键id是连续单调递增的；

可以用一个缓存记录上次访问的位置的id，然后下次翻页直接id<缓存+20；就可以了（该方法可行是因为主键是单调递增的）

9. 优化关联查询

10. 优化子查询

尽量用关联查询代替子查询。

11. 优化group by和distinct

MySQL使用同样的方法优化它们，而且在内部会相互转化这两类查询。

尽量用索引优化它们。

12. 优化union查询

13. 使用用户自定义变量