# 10 **MySQL为什么有时候会选错索引？**

先建一个表，表里有a,b两个字段，并分别建上索引。

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`a` int(11) DEFAULT NULL,

`b` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `a` (`a`),

KEY `b` (`b`)

) ENGINE=InnoDB；

然后，往表t插入10万行记录，取值按整数递增，即（1,1,1），（2,2,2）（3,3,3）直到（100000，100000,100000）用存储过程。

Delimiter ;;

Create procedure idata()

Begin

Declare i int;

Set i = 1;

While(i<=100000)

Insert into t values（i,i,i）;

Set i= i +1;

End while;

End;;

Delimiter;

Call idata();

接下来分析一条语句，

Select \* from t where a between 10000 and 20000;

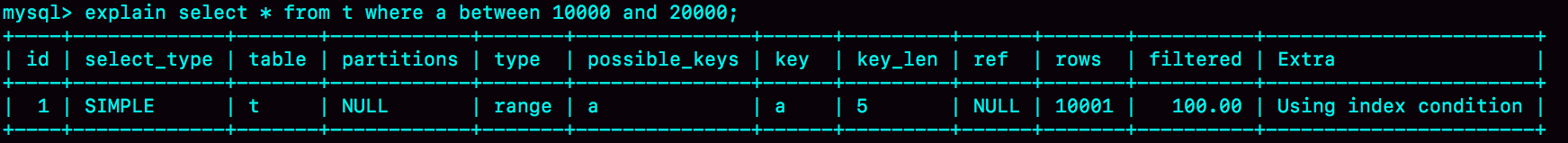


图1，使用explain命令查看语句执行情况。

从图1上看，这条查询语句的执行符合预期，key 这个字段是 a ,表示优化器选择了索引a

不过看如下的例子准备好10万行的数据，在做如下

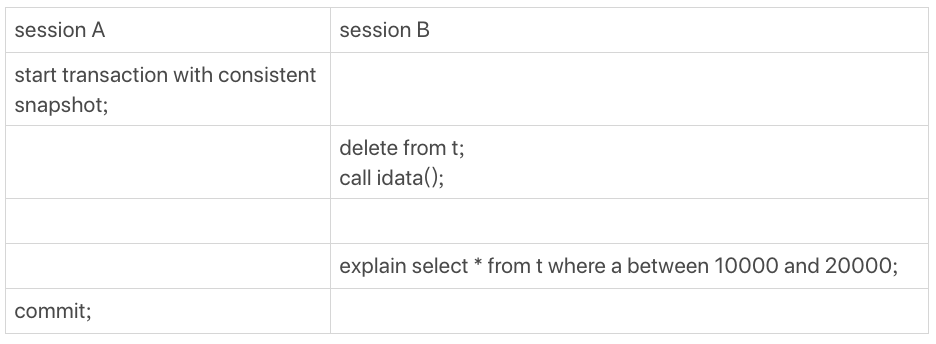


图2 sessionA和sessionB的执行流程

SessionA开启了一个事务，SessionB把数据都删除，有调用idata这个存储过程，插入10万行数据，

sessionB的查询语句select \* from t where a between 10000 and 20000 就不会再选择索引a了，我们可以通过慢查询日志（slow log）来看下具体的情况

为了说明优化器选择的结果是否正确，使用force index(a) 来让优化器强制使用索引a,

set long\_query\_time=0;

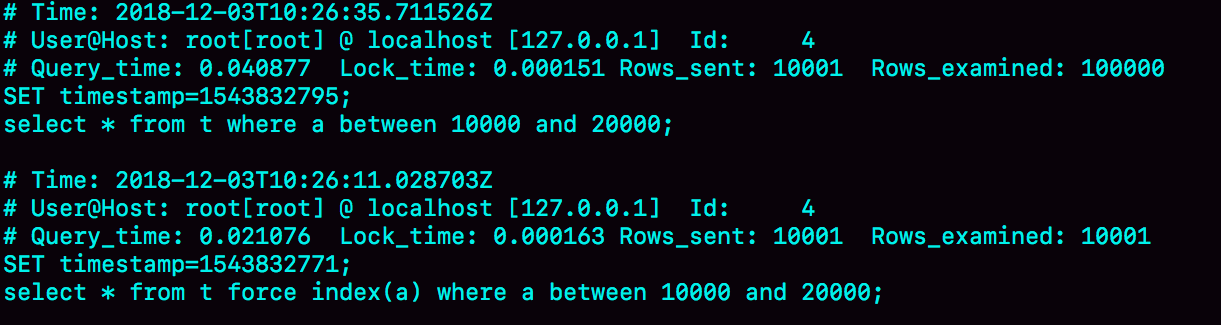
select \* from t where a between 10000 and 20000; /\*Q1\*/

select \* from t force index(a) where a between 10000 and 20000;/\*Q2\*/

第一句，将慢查询日志的阈值设置为0，表示这个线程接下来的语句都会被记录到慢查询日 志中

第二句，Q1是sessionB原来的语句

第三句，Q2 是加了force index(a)和sessionB原来的查询语句执行情况对比

 图3 slow log结果

Q1,扫描10万行，虽然走全表扫描，执行40毫秒，Q2扫描10001行，执行了21毫秒，也就是我们在没有使用force index的时候，MySQL用错了索引，导致了更长的执行时间。 **优化器逻辑**

选择索引是优化器的工作，而优化器的目的是找到一个最优的执行方案，并用最小代价去执行语句，在数据库里面，扫描行数是影响执行代价的原因之一，扫描的行数越少，访问磁盘数据的次数越少，消耗CPU资源越少。

扫描行数并不是唯一的判断标准，优化器还会结合是否使用临时表，是否排序等因素进行综合判断。

**扫描行数是怎么判断的？**

MySQL在真正进行执行语句前，并不能精确的知道满足这个条件的记录有多少条，而只是根据统计信息来估算记录。

这个统计信息就是索引的“区分度”，一个索引上不同的值越多，这个索引的区分度就越好，而一个索引上不同的数，我们称之为“基数”，这个基数越大，索引区分度越好。

可以使用show index 看到一个索引的基数

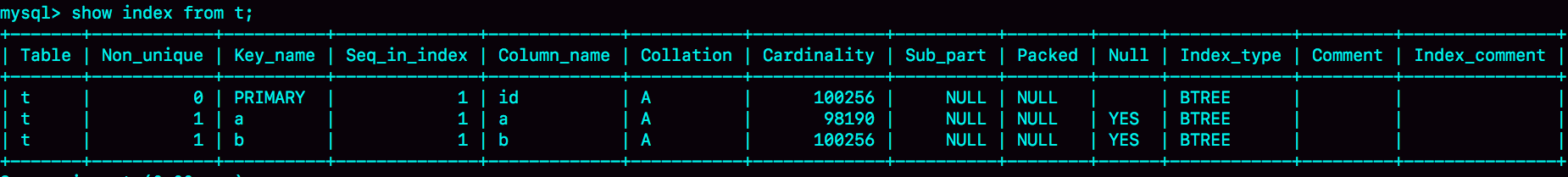


图4 表t的show index 结果

虽然这个表的每一行的字段都是一样的，但是统计信息，这个三个索引的基数并不同，而且都不准确，

**MySQL是怎样得到索引的基数？--** 采样统计

为什么要采样统计？因为把一张表取出来一行行统计，虽然可以得到精确的结果，但是代价太大，只能采样统计。

采样统计，InnoDB默认会选择N个数据页，统计这些页面上的不同的值，得到一个平均值，然后乘以这个索引的页面数，就得到了这个索引基数。而数据表的持续更新，索引统计信息也不会固定不变，所以当变更的数据行超过1/M的时候，会自动触发重新做一次索引统计。

MySQL中，有两种索引统计的方式，可以通过设置参数innodb\_stats\_persistent的值来选择

设置为on，统计信息会持久存储，此时，默认N是20，M是10

设置为off,统计信息只存储在内存中，此时，默认N是8，M是16

由于采样统计，不管N是20还是8，基数很容易不准确，

可以从图4中看到，这次索引的统计值（cardinality列）虽不够精确，但大体上还是差不多，选错索引还有其他的原因

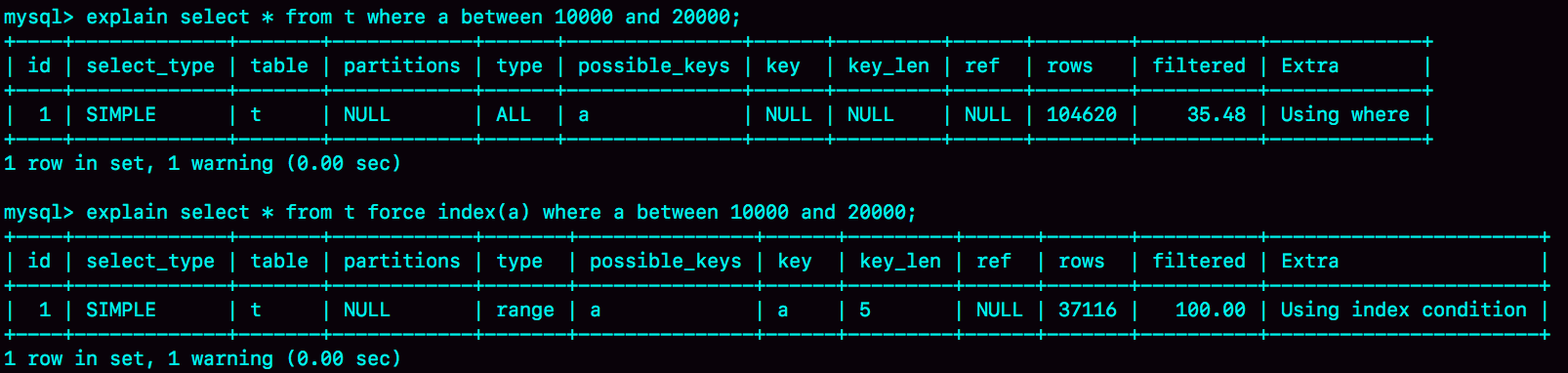


图 5 意外的explain 结果

Rows这个字段表示预估扫描行数，Q1的结果还是符合预期，rows是104620，但是Q2是37116行，偏差就大了，而图1用explain命令看到rows只有10001行，是这个偏差误导了优化的判断

此时，一个疑问不是为什么不准，而是优化器为什么放着37000行的执行计划不用，却选择了扫描100000的执行计划呢？

这是因为如果使用算因a,每次从索引a上拿到一个值，都要回到主键索引上查出整行数据，这个大家也是要算进去的。

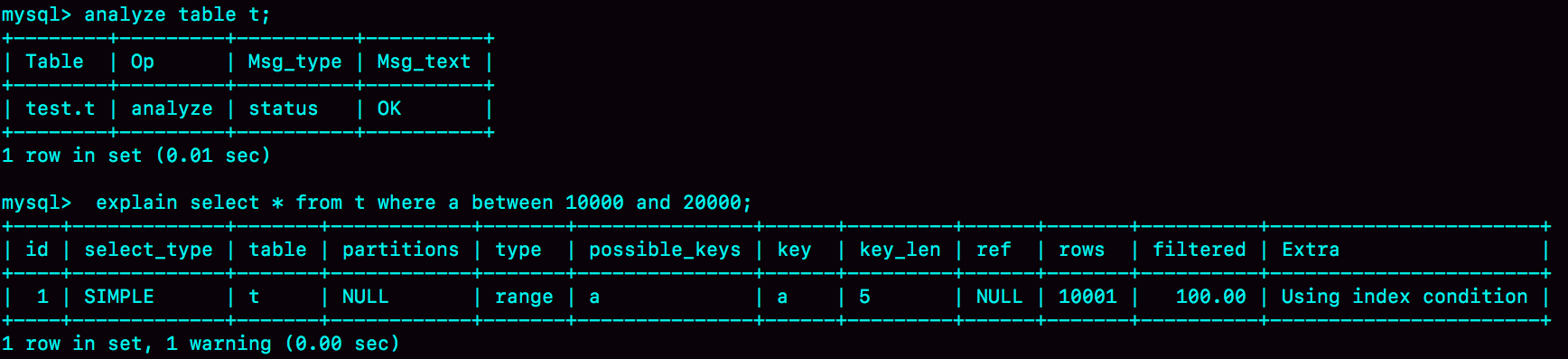
而如果选择扫描10万行，是直接在主键索引上扫描的，没有额外的代价。

优化器估算这两个选择的行，从结果看来，优化器认为直接扫描主键索引更快，当然，从执行时间来看，这个选择并不是最优的。

使用普通索引需要把回表的代价算进去，图1执行explain的时候，也考虑了这个策略的代价，但图1的选择是对的。

所以mysql选错索引，这事还得归咎于没能准确的判断出扫描行数，至于为什么会得到错误的扫描行数，**这个原因作为课后思考题，**

既然统计信息不对，那就修正，analyze table t 命令，可以用来重新统计索引信息。我们来看一下执行效果。



这回就对了

所以在实践中，如果发现explain的结果预估rows值跟实际情况差距比较大，可以采用这个方法来处理。

其实如果统计不准确，通过analyze命令可以解决很多问题，优化器可不止是看扫描行数。

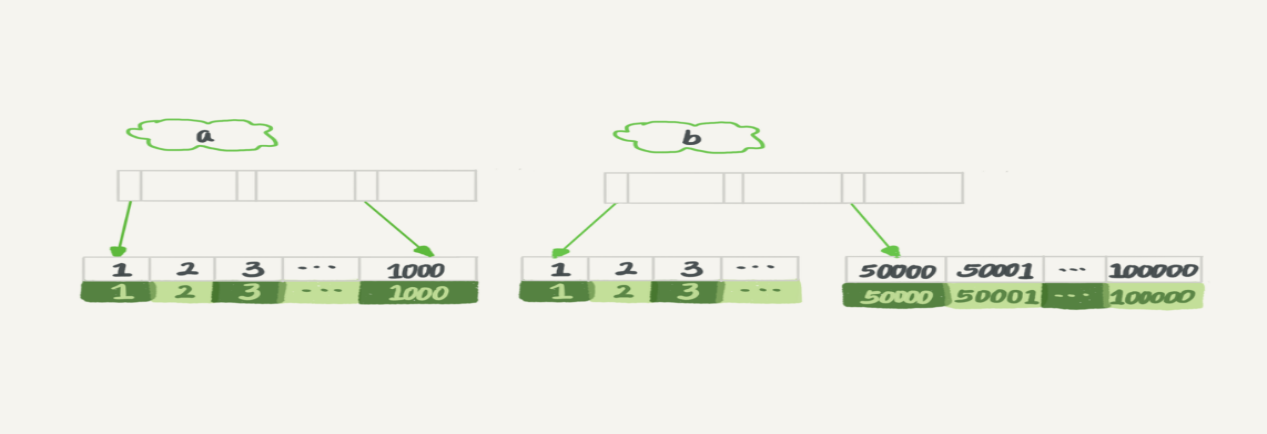
基于这个表t

mysql> select \* from t where (a between 1 and 1000) and (b between 50000 and 100000) order by b limit 1;

从条件上看，这个查询没有符合条件的记录，会因此返回空集合

在开始执行这条语句之前，如果人工选择索引，会选择那个？

为了便于分析，看下索引结构图

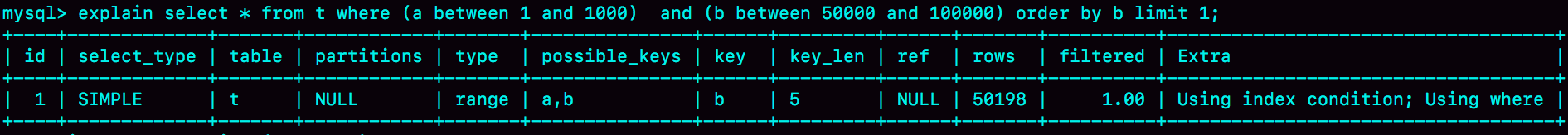


如果是用索引a进行查询，就是扫描a的前1000个值，然后去b的对应的id，再到主键索引去查出每一行，然后根据字段B来过滤，显然这样需要扫描1000行。

如果使用索引b进行查询，即使扫描b的最后的50001个值，与上面的执行过程相同，也需要回表在判断，所以需要扫描50001行。

如果使用索引a的话，执行速度会快很多，看下执行结果

mysql> explain select \* from t where (a between 1 and 1000) and (b between 50000 and 100000) order by b limit 1;



可以看到返回结果中key字段显示，这次选择索引b,而rows字段显示需要扫描的行数是50198

从结果可以得出结论

1. 扫描行数的估计的值不准确
2. 这个例子mysql选错索引

**索引选择异常和处理**

1. **使用force index强行选择一个索引。**MySQL会根据词法解析的结果分析出来可以使用的索引作为候选项，然后在候选列表中依次判断每隔索引需要扫描多少行。如果force index指定索引在候选列表中，就直接选择这个索引，不再评估其他索引的代价
2. **考虑修改语句，引导MySQL使用期望索引，**比如，在这个例子中，显然把

“order by b limit 1 ”改成“order by b,a limit 1”，语义逻辑相同

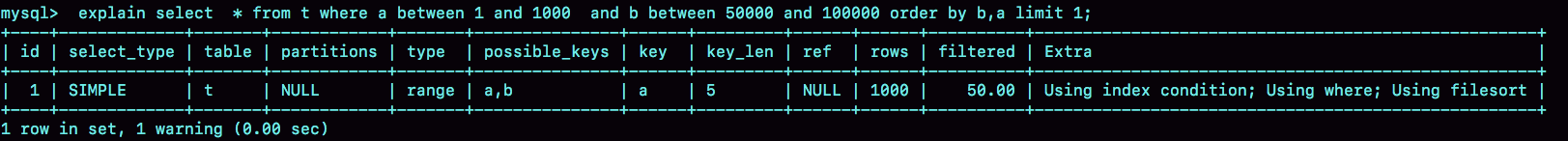


图10 order by b,a limit执行结果

之前优化器使用索引b,是因为它认为使用索引b可以避免排序（b本身就是索引，已经是有序的了，如果选择 索引b的话，不需要在排序，只需要遍历），所以扫描行数多，也判定代价更小。

现在order by b,a这种写法，要求b,a排序，就意味着两个索引都需要排序，因此扫描行数影响决策的主要条件，于是此时优化器选择了只需要扫描1000行的索引a

当然，这种修改并不是通用的手段，只是刚好在这个语句里有limit 1 因此如果满足条件的记录，order by b limit1 和order by b,a limit 1都会返回b是最小的哪一行，逻辑上一致，才可以这么做

还有一种改法

mysql> select \* from (select \* from t where (a between 1 and 1000) and (b between 50000 and 100000) order by b limit 100)alias limit 1;

让优化器意识到使用索引b代价就是很高的，不具备通用性

第三种，**在有些场景下，可以新建一个更合适的索引，来提供优化器做选择，或者删掉误用的索引。**

思考题：

通过sessionA配合，让sessionB删除数据后有重新插入了一遍数据，然后就让发现explain结果中，rows字段从10001变成37000多，

而如果没有sessionA的配合，只是单独执行delete from t,call idata(),expain 这三句话，会看到rows字段其实还是10000左右，这是什么原因呢？