

## mysql怎么调优

### 第一：选择最合适的字段属性。

mysql在创建数据库的时候肯定是数据库中的表越小越好，这样才能提高查询的速度。但是现实往往不如人意，每天可能产生的数据量是以数万或者数十万为单位的，这时就要充分地考虑在建表时的字段长度了。比如说在存储电话号的时候，如果将其写成CHAR(255),这显然会给数据库带来很多不必要的空间浪费，明明CHAR(11)就可以解决的问题。

### 第二：使用连接查询（join）代替子查询。

子查询的优点是可以使用简单的SELECT语句就可以完成逻辑较为复杂的查询动作，而且还能避免死锁的情况产生。但是有时也可以考虑使用连接查询来完成，**毕竟连接查询不需要像子查询一样在内存中创建临时表，再从临时表中过滤数据，从而加快查询速度。**

### 第三：使用union联合查询。

union本来就有联合的含义，它可以将多个SELECT语句的查询结果联合到一个查询中，**从而替代了手动创建临时表的过程。而且union还有一个好处就是使用完了以后自动就删除了，一点也不占用内存空间。**这真是“事了拂衣去，片叶不沾身”呀。

### 第四：通过事务来管理。

事务是数据库调优时的一个老生常谈的概念，由于其4大特性，事务是不可避免的调优方式之一，它可以保证数据的完整性、一致性。可以用一句话来总结“去不了终点，那就回到原点。”

### 第五：锁定表。

尽管事务能够保证数据库的完整性和一致性，但是它本身其实也存在一些弊端。因为它本身具有一定的独占性，当事务没有执行完毕以前，用户发出的其他操作就只能等待，一直等到所有的事务都结束了才能继续执行。如果用户访问量特别大的时候，这会造成系统的严重延迟问题。

所以可以通过锁定表的方法，保证在没有执行到UNLOCKTABLES指令之前所有被LOCKTABLE修饰地的查询语句不会被插入、删除和修改。

### 第六：合理使用外键。

外键本身存在的作用就是保证表与表之间的参照完整性，使用外键可以有效地增加数据之间的关联性。

### 第七：正确使用索引。

索引是提高数据库性能的最常见的方法，能够正确地使用索引，能够大大地提高查询效率。

但是也不能够为所有的列都创建索引，因为它本身会占用内存，维护起来也很麻烦，它就是一把双刃剑，所以索引在使用的时候需要根据实际情况正确使用才行。

### 第八：优化查询语句。

一个好的查询语句能够很明显地提高查询效率，反之就会慢到怀疑是不是程序写错了。

那么什么样的查询语句算是好的呢？可以参考以下几点要求

- 1、在比较的时候尽量在相同的字段之间进行比较。例如不能将一个带有索引的INT字段和BIGINT字段进行比较。
- 2、已将创建了索引的字段上不要使用函数，那样会导致索引失效。

3、查询时不鼓励使用like语句查询，因为那会消耗掉一部分系统的性能

## 一.创建索引

- 1.要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引
- 2.(1)在经常需要进行检索的字段上创建索引，比如要按照表字段username进行检索，那么就应该在姓名字段上创建索引，如果经常要按照员工部门和员工岗位级别进行检索，那么就应该在员工部门和员工岗位级别这两个字段上创建索引。  
(2)创建索引给检索带来的性能提升往往是巨大的，因此在发现检索速度过慢的时候应该首先想到的就是创建索引。  
(3)一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。

## 二.避免在索引上使用计算

在where字句中，如果索引列是计算或者函数的一部分，DBMS的优化器将不会使用索引而使用全表查询,函数

属于计算的一种,同时在in和exists中通常情况下使用EXISTS，因为in不走索引  
效率低：

```
1  select * from user where salary*22>11000(salary是索引列)
2
```

效率高

```
1  select * from user where salary>11000/22(salary是索引列)
2
```

## 三.使用预编译查询

程序中通常是根据用户的输入来动态执行SQL，这时应该尽量使用参数化SQL,这样不仅可以避免SQL注入漏洞

攻击，最重要数据库会对这些参数化SQL进行预编译，这样第一次执行的时候DBMS会为此SQL语句进行查询优化

并且执行预编译，这样以后再执行这个SQL的时候就直接使用预编译的结果，这样可以大大提高执行的速度。

## 四.调整Where字句中的连接顺序

DBMS一般采用自下而上的顺序解析where字句，根据这个原理表连接最好写在其他where条件之前，那些可以过滤掉最大数量记录。

## 五.尽量将多条SQL语句压缩到一句SQL中

每次执行SQL的时候都要建立网络连接、进行权限校验、进行SQL语句的查询优化、发送执行结果，这个过程是非常耗时的，因此应该尽量避免过多的执行SQL语句，能够压缩到一句SQL执行的语句就不要用多条来执行。

## 六.用where字句替换HAVING字句

避免使用HAVING字句，因为HAVING只会在检索出所有记录之后才对结果集进行过滤，而where则是在聚合前刷选记录，如果能通过where字句限制记录的数目，那就能减少这方面的开销。HAVING中的条件一般用于聚合函数的过滤，除此之外，应该将条件写在where字句中。

## 七.使用表的别名

当在SQL语句中连接多个表时，请使用表的别名并把别名前缀于每个列名上。这样就可以减少解析的时间并减少哪些友列名歧义引起的语法错误。

## 八.用union all替换union

当SQL语句需要union两个查询结果集合时，即使检索结果中不会有重复的记录，如果使用union这两个结果集同样会尝试进行合并，然后在输出最终结果前进行排序，因此如果可以判断检索结果中不会有重复的记录时候，应该用union all，这样效率就会因此得到提高。

## 九.考虑使用“临时表”暂存中间结果

简化SQL语句的重要方法就是采用临时表暂存中间结果，但是，临时表的好处远远不止这些，将临时结果暂存在临时表，后面的查询就在tempdb中了，这可以避免程序中多次扫描主表，也大大减少了程序执行中“共享锁”阻塞“更新锁”，减少了阻塞，提高了并发性能。

但是也得避免频繁创建和删除临时表，以减少系统表资源的消耗。

## 十.只在必要的情况下才使用事务begin tran

SQL Server中一句SQL语句默认就是一个事务，在该语句执行完成后也是默认commit的。其实，这就是begin tran的一个最小化的形式，好比在每句语句开头隐含了一个begin tran，结束时隐含了一个commit。

有些情况下，我们需要显式声明begin tran，比如做“插、删、改”操作需要同时修改几个表，要求要么几个表都修改成功，要么都不成功。begin tran可以起到这样的作用，它可以把若干SQL语句套在一起执行，最后再一起commit。好处是保证了数据的一致性，但任何事情都不是完美无缺的。Begin tran付出的代价是在提交之前，所有SQL语句锁住的资源都不能释放，直到commit掉。

可见，如果Begin tran套住的SQL语句太多，那数据库的性能就糟糕了。在该大事务提交之前，必然会阻塞别的语句，造成block很多。

Begin tran使用的原则是，在保证数据一致性的前提下，begin tran 套住的SQL语句越少越好！有些情况下可以采用触发器同步数据，不一定要用begin tran。

## 十一.尽量避免使用游标

尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

## 十二.用varchar/nvarchar 代替 char/nchar

尽可能的使用 varchar/nvarchar 代替 char/nchar，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了，不管是否插入值（NULL也包含在内），都是占用 100个字符的空间的，如果是varchar这样的变长字段，null 不占用空间。

## 十三.查询select语句优化

- 1.任何地方都不要使用 select \* from t，用具体的字段列表代替“\*”，不要返回用不到的任何字段
- 2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，

```
1      select id from t where num is null
2
```

可以在num上设置默认值0，确保表中num列没有null值，  
然后这样查询：

```
1      select id from t where num=0
2      select id from t where num=10 or num=20
```

可以这样查询：

```
1      select id from t where num=10
2      union all
3      select id from t where num=20
4
```

#### 4.不能前置百分

```
1  select id from t where name like '%abc%'
2
```

若要提高效率，可以考虑全文检索。

```
1      select id from t where num in(1,2,3)
2
```

对于连续的数值，能用 between 就不要用 in 了：

```
1      select id from t where num between 1 and 3
2
```

6.如果查询的两个表大小相当，那么用in和exists差别不大。

in：

例如：表A（小表），表B（大表）

```
1  select * from A where cc in (select cc from B) 效率低，用到了A表上cc列的索引；
2  select * from A where exists(select cc from B where cc=A.cc) 效率高，用到了B表上cc列的索引。
3
```

相反的

```
1  select * from B where cc in (select cc from A) 效率高，用到了B表上cc列的索引；
2  select * from B where exists(select cc from A where cc=B.cc) 效率低，用到了A表上cc列的索引。
3
```

## 十四.更新Update语句优化

1.如果只更改1、2个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志

## 十五. 删除Delete语句优化语句

1.最高效的删除重复记录方法 ( 因为使用了ROWID)例子：

```
1  DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP_NO =  
    E.EMP_NO);
```

## 十六.插入Insert语句优化

1.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

-----

## sql优化的15个小技巧

避免使用select \*

用union all代替union

小表驱动大表

批量操作

多用limit

in中值太多

增量查询

高效的分页

用连接查询代替子查询

join的表不宜过多

join时要注意

控制索引的数量

选择合理的字段类型

提升group by的效率

索引优化

### 1. 避免使用select \*

很多时候，我们写sql语句时，为了方便，喜欢直接使用select \*，一次性查出表中所有列的数据。

反例：

```
select * from user where id=1;
```

1

在实际业务场景中，可能我们真正需要使用的只有其中一两列。查了很多数据，但是不用，白白浪费了数据库资源，比如：内存或者cpu。

此外，多查出来的数据，通过网络IO传输的过程中，也会增加数据传输的时间。

还有一个最重要的问题是：select不会走覆盖索引，会出现大量的回表操作，从而导致查询sql的性能很低。

那么，如何优化呢？

正例

```
select name,age from user where id=1;
```

1

sql语句查询时，只查需要用到的列，多余的列根本无需查出来。

## 2. 用union all代替union

我们都知道sql语句使用union关键字后，可以获取排重后的数据。而如果使用union all关键字，可以获取所有数据，包含重复的数据。

反例：

```
(select * from user where id=1)
```

union

```
(select * from user where id=2);
```

1

2

3

排重的过程需要遍历、排序和比较，它更耗时，更消耗cpu资源。所以如果能用union all的时候，尽量不用union。

正例：

```
(select * from user where id=1)
```

union all

```
(select * from user where id=2);
```

除非是有些特殊的场景，比如union all之后，结果集中出现了重复数据，而业务场景中是不允许产生重复数据的，这时可以使用union。

## 3. 小表驱动大表

小表驱动大表，也就是说用小表的数据集驱动大表的数据集。

假如有order和user两张表，其中order表有10000条数据，而user表有100条数据。时如果想查一下，所有有效的用户下过的订单列表。可以使用in关键字实现：

```
select * from order
```

```
where user_id in (select id from user where status=1)
```



也可以使用exists关键字实现：

```
select * from order
where exists (select 1 from user where order.user_id = user.id and status=1)
```

前面提到的这种业务场景，使用in关键字去实现业务需求，更加合适。

为什么呢？

因为如果sql语句中包含了in关键字，则它会优先执行in里面的子查询语句，然后再执行in外面的语句。

如果in里面的数据量很少，作为条件查询速度更快。

而如果sql语句中包含了exists关键字，它优先执行exists左边的语句（即主查询语句）。然后把它作为条件，去跟右边的语句匹配。如果匹配上，则可以查询出数据。如果匹配不上，数据就被过滤掉了。

这个需求中，order表有10000条数据，而user表有100条数据。order表是大表，user表是小表。如果order表在左边，则用in关键字性能更好。

总结一下：

in 适用于左边大表，右边小表。

exists 适用于左边小表，右边大表。

不管是用in，还是exists关键字，其核心思想都是用小表驱动大表。

#### 4. 批量操作

如果你有一批数据经过业务处理之后，需要插入数据，该怎么办？

反例：

```
for(Order order: list){
orderMapper.insert(order);
}
```

在循环中逐条插入数据。

```
insert into order(id,code,user_id)
values(123,'001',100);
```

1

2

该操作需要多次请求数据库，才能完成这批数据的插入。

但众所周知，我们在代码中，每次远程请求数据库，是会消耗一定性能的。而如果我们代码需要请求多次数据库，才能完成本次业务功能，势必会消耗更多的性能。

那么如何优化呢？

正例：

```
orderMapper.insertBatch(list):
```

1

提供一个批量插入数据的方法。

```
insert into order(id,code,user_id)
```

```
values(123,'001',100),(124,'002',100),(125,'003',101);
```

1

2

这样只需要远程请求一次数据库，sql性能会得到提升，数据量越多，提升越大。

但需要注意的是，不建议一次批量操作太多的数据，如果数据太多数据库响应也会很慢。批量操作需要把握一个度，建议每批数据尽量控制在500以内。如果数据多于500，则分多批次处理。

## 5. 多用limit

有时候，我们需要查询某些数据中的第一条，比如：查询某个用户下的第一个订单，想看看他第一次的首单时间。

反例：

```
select id, create_date
from order
where user_id=123
order by create_date asc;
```

根据用户id查询订单，按下单时间排序，先查出该用户所有的订单数据，得到一个订单集合。然后在代码中，获取第一个元素的数据，即首单的数据，就能获取首单时间。

```
List<Order> list = orderMapper.getOrderList();
Order order = list.get(0);
```

虽说这种做法在功能上没有问题，但它的效率非常不高，需要先查询出所有的数据，有点浪费资源。

那么，如何优化呢？

正例：

```
select id, create_date
from order
where user_id=123
order by create_date asc
limit 1;
```

使用limit 1，只返回该用户下单时间最小的那一条数据即可。

此外，在删除或者修改数据时，为了防止误操作，导致删除或修改了不相干的数据，也可以在sql语句最后加上limit。

例如：

```
update order set status=0,edit_time=now(3)
where id>=100 and id<200 limit 100;
```

这样即使误操作，比如把id搞错了，也不会对太多的数据造成影响。

## 6. in中值太多

对于批量查询接口，我们通常会使用in关键字过滤出数据。比如：想通过指定的一些id，批量查询出用户信息。

sql语句如下：

```
select id,name from category
where id in (1,2,3...1000000000);
```

1  
2

如果我们不做任何限制，该查询语句一次性可能会查询出非常多的数据，很容易导致接口超时。这时该怎么办呢？

```
select id,name from category
where id in (1,2,3...100)
limit 500;
```

1  
2  
3

可以在sql中对数据用limit做限制。

不过我们更多的是要在业务代码中加限制，伪代码如下：

```
public List<Category> getCategory(List<Long> ids) {
    if(CollectionUtils.isEmpty(ids)) {
        return null;
    }
    if(ids.size() > 500) {
        throw new BusinessException("一次最多允许查询500条记录")
    }
    return mapper.getCategoryList(ids);
}
```

## 7. 增量查询

有时候，我们需要通过远程接口查询数据，然后同步到另外一个数据库。

反例：

```
select * from user;
```

1

如果直接获取所有的数据，然后同步过去。这样虽说非常方便，但是带来了一个非常大的问题，就是如果数据很多的话，查询性能会非常差。

这时该怎么办呢？

正例：

```
select *  
from user  
where id>#{lastId} and create_time >= #{lastCreateTime}  
limit 100;
```

按id和时间升序，每次只同步一批数据，这一批数据只有100条记录。每次同步完成之后，保存这100条数据中最大的id和时间，给同步下一批数据的时候用。

通过这种增量查询的方式，能够提升单次查询的效率。

## 8. 高效的分页

有时候，列表页在查询数据时，为了避免一次性返回过多的数据影响接口性能，我们一般会对查询接口做分页处理。

在mysql中分页一般用的limit关键字：

```
select id,name,age  
from user limit 10,20;
```

1

2

如果表中数据量少，用limit关键字做分页，没啥问题。但如果表中数据量很多，用它就会出现性能问题。

比如现在分页参数变成了：

```
select id,name,age  
from user limit 1000000,20;
```

mysql会查到1000020条数据，然后丢弃前面的1000000条，只查后面的20条数据，这个是非常浪费资源的。

那么，这种海量数据该怎么分页呢？

优化sql：

```
select id,name,age  
from user where id > 1000000 limit 20;
```

1

2

先找到上次分页最大的id，然后利用id上的索引查询。不过该方案，要求id是连续的，并且有序的。

还能使用between优化分页。

```
select id,name,age  
from user where id between 1000000 and 1000020;
```

1

2

需要注意的是between要在唯一索引上分页，不然会出现每页大小不一致的问题。

## 9. 用连接查询代替子查询

mysql中如果需要从两张以上的表中查询出数据的话，一般有两种实现方式：子查询和 连接查询。

子查询的例子如下：

```
select * from order
where user_id in (select id from user where status=1)
```

1

2

子查询语句可以通过in关键字实现，一个查询语句的条件落在另一个select语句的查询结果中。程序先运行在嵌套在最内层的语句，再运行外层的语句。

子查询语句的优点是简单，结构化，如果涉及的表数量不多的话。

但缺点是mysql执行子查询时，需要创建临时表，查询完毕后，需要再删除这些临时表，有一些额外的性能消耗。

这时可以改成连接查询。 具体例子如下：

```
select o.* from order o
inner join user u on o.user_id = u.id
where u.status=1
```

1

2

3

## 10. join的表不宜过多

根据阿里巴巴开发者手册的规定，join表的数量不应该超过3个。

反例：

```
select a.name,b.name,c.name,d.name
from a
inner join b on a.id = b.a_id
inner join c on c.b_id = b.id
inner join d on d.c_id = c.id
inner join e on e.d_id = d.id
inner join f on f.e_id = e.id
inner join g on g.f_id = f.id
```

如果join太多，mysql在选择索引的时候会非常复杂，很容易选错索引。

并且如果没有命中中，nested loop join 就是分别从两个表读一行数据进行两两对比，复杂度是  $n^2$ 。

所以我们应该尽量控制join表的数量。

正例：

```
select a.name,b.name,c.name,a.d_name
from a
```

```
inner join b on a.id = b.a_id
```

```
inner join c on c.b_id = b.id
```

如果实现业务场景中需要查询出另外几张表中的数据，可以在a、b、c表中冗余专门的字段，比如：在表a中冗余d\_name字段，保存需要查询出的数据。

不过我之前也见过有些ERP系统，并发量不大，但业务比较复杂，需要join十几张表才能查询出数据。所以join表的数量要根据系统的实际情况决定，不能一概而论，尽量越少越好。

## 11. join时要注意

我们在涉及到多张表联合查询的时候，一般会使用join关键字。

而join使用最多的是left join和inner join。

left join：求两个表的交集外加左表剩下的数据。

inner join：求两个表交集的数据。

使用inner join的示例如下：

```
select o.id,o.code,u.name
from order o
inner join user u on o.user_id = u.id
where u.status=1;
```

如果两张表使用inner join关联，mysql会自动选择两张表中的小表，去驱动大表，所以性能上不会有太大的问题。

使用left join的示例如下：

```
select o.id,o.code,u.name
from order o
left join user u on o.user_id = u.id
where u.status=1;
```

1

2

3

4

如果两张表使用left join关联，mysql会默认用left join关键字左边的表，去驱动它右边的表。如果左边的表数据很多时，就会出现性能问题。

要特别注意的是在用left join关联查询时，左边要用小表，右边可以用大表。如果能用inner join的地方，尽量少用left join。

## 12. 控制索引的数量

众所周知，索引能够显著的提升查询sql的性能，但索引数量并非越多越好。

因为表中新增数据时，需要同时为它创建索引，而索引是需要额外的存储空间的，而且还会有一定的性能消耗。

阿里巴巴的开发者手册中规定，单表的索引数量应该尽量控制在5个以内，并且单个索引中的字段数不超过5个。

mysql使用的B+树的结构来保存索引的，在insert、update和delete操作时，需要更新B+树索引。如果索引过多，会消耗很多额外的性能。

那么，问题来了，如果表中的索引太多，超过了5个该怎么办？

这个问题要辩证的看，如果你的系统并发量不高，表中的数据量也不多，其实超过5个也可以，只要不要超过太多就行。

**但对于一些高并发的系统，请务必遵守单表索引数量不要超过5的限制。**

那么，高并发系统如何优化索引数量？

能够建联合索引，就别建单个索引，可以删除无用的单个索引。

将部分查询功能迁移到其他类型的数据库中，比如：Elastic Search、HBase等，在业务表中只需要建几个关键索引即可。

### 13. 选择合理的字段类型

char表示固定字符串类型，该类型的字段存储空间是固定的，会浪费存储空间。

```
alter table order
```

```
add column code char(20) NOT NULL;
```

1

2

**varchar表示变长字符串类型，该类型的字段存储空间会根据实际数据的长度调整，不会浪费存储空间。**

```
alter table order
```

```
add column code varchar(20) NOT NULL;
```

1

2

如果是长度固定的字段，比如用户手机号，一般都是11位的，可以定义成char类型，长度是11字节。

但如果是企业名称字段，假如定义成char类型，就有问题了。

如果长度定义得太长，比如定义成了200字节，而实际企业长度只有50字节，则会浪费150字节的存储空间。

如果长度定义得太短，比如定义成了50字节，但实际企业名称有100字节，就会存储不下，而抛出异常。

所以建议将企业名称改成varchar类型，变长字段存储空间小，可以节省存储空间，而且对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

我们在选择字段类型时，应该遵循这样的原则：

**能用数字类型，就不用字符串，因为字符的处理往往比数字要慢。**

**尽可能使用小的类型，比如：用bit存布尔值，用tinyint存枚举值等。**

**长度固定的字符串字段，用char类型。**

**长度可变的字符串字段，用varchar类型。**

金额字段用decimal，避免精度丢失问题。

还有很多原则，这里就不一一列举了。

## 14. 提升group by的效率

我们有很多业务场景需要使用group by关键字，它主要的功能是去重和分组。

通常它会跟having一起配合使用，表示分组后再根据一定的条件过滤数据。

反例：

```
select user_id,user_name from order
group by user_id
having user_id <= 200;
```

1

2

3

这种写法性能不好，它先把所有的订单根据用户id分组之后，再去过滤用户id大于等于200的用户。

分组是一个相对耗时的操作，为什么我们不先缩小数据的范围之后，再分组呢？

正例：

```
select user_id,user_name from order
where user_id <= 200
group by user_id
```

1

2

3

使用where条件在分组前，就把多余的数据过滤掉了，这样分组时效率就会更高一些。

其实这是一种思路，不仅限于group by的优化。我们的sql语句在做一些耗时的操作之前，应尽可能缩小数据范围，这样能提升sql整体的性能。

## 15. 索引优化

sql优化当中，有一个非常重要的内容就是：索引优化。

很多时候sql语句，走了索引，和没有走索引，执行效率差别很大。所以索引优化被作为sql优化的首选。

索引优化的第一步是：检查sql语句有没有走索引。

那么，如何查看sql走了索引没？

可以使用explain命令，查看mysql的执行计划。

例如：

```
explain select * from order where code='002';
```

1

结果：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(NULL)	const	un_code,un_code_name	un_code	82	const	1	100.00	(NULL)



通过这几列可以判断索引使用情况，执行计划包含列的含义如下图所示：



说实话，sql语句没有走索引，排除没有建索引之外，最大的可能性是索引失效了。

下面说说索引失效的常见原因：

如果不是上面的这些原因，则需要再进一步排查一下其他原因。

此外，你有没有遇到过这样一种情况：明明是同一条sql，只有入参不同而已。有的时候走的索引a，有的时候却走的索引b？

没错，有时候mysql会选错索引。

必要时可以使用force index来强制查询sql走某个索引。

