111 案例实战：千万级用户场景下的运营系统SQL调优（3）

今天是我们第一个千万级用户场景下的运营系统SQL调优案例的最后一讲，也是最关键的一讲，我们要根据SQL语句的执行计划找出他速度慢的原因所在，然后还得想办法去优化他的速度。

上一次我们已经对SQL语句的执行计划做了一个分析，知道了那个SQL语句的执行过程，今天就得对SQL执行计划做一个透彻的分析，看看到底为什么他会慢

先来回看一下那个执行计划的内容：

```
+----+-------------+-------+------------+-------+---------------+----------+---------+------+
| id | select_type | table | type  | key | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+
| 1 | SIMPLE |  | ALL | NULL | NULL | 100.00 | NULL  |
| 1 | SIMPLE | users | ALL | NULL | 49651 | 10.00 | Using where; Using join buffer(Block Nested Loop)  |
| 2 | MATERIALIZED | users_extent_info | range | idx_login_time | 4561 | 100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+
```

之前说过，他执行的过程就是先执行了子查询查出来4561条数据，物化成了一个临时表，接着他对users主表做了一个全表扫描，扫描的过程中把每一条数据都放到物化临时表里去做全表扫描，本质在做join的事情。

那么这里为什么会跑的这么慢呢？其实很明显了，大家可以想一下，首先他对子查询的结果做了一次物化临时表，落地磁盘了，接着他还全表扫描了users表的所有数据，每一条数据居然跑到一个没有索引的物化临时表里再做一次全表扫描找匹配数据。

在这个过程里，对users表的全表扫描耗时不耗时？对users表的每一条数据跑到物化临时表里做全表扫描，耗时不耗时？所以这个过程必然是非常慢的，几乎就没怎么用到索引。

那么接着我们就很奇怪了，为什么会出现上述的一个全表扫描users表，然后跟物化临时表做join，join的时候还要全表扫描物化临时表的过程？

这里交大家一个技巧，就是在执行完上述SQL的EXPLAIN命令，看到执行计划之后，可以执行一下**show warnings**命令。

这个show warnings命令此时显示出来的内容如下：

/* select#1 */ select count(`d2`.`users`.`user_id`) AS `COUNT(users.user_id)`

from `d2`.`users` `users` semi join xxxxxx，下面省略一大段内容，因为可读性实在不高，大家关注的应该是这里的**semi join**这个关键字

这里就显而易见了！MySQL在这里，生成执行计划的时候，自动就把一个普通的IN子句，“优化”成了基于semi join来进行IN+子查询的操作，这个semi join是什么意思呢？

简单来说，对users表不是全表扫描了么？对users表里每一条数据，去对物化临时表全表扫描做semi join，不需要把users表里的数据真的跟物化临时表里的数据join上。只要users表里的一条数据，在物化临时表里可以找到匹配的数据，那么users表里的数据就会返回，这就叫做semi join，他是用来筛选的。

所以慢，也就慢在这里了，那既然知道了是semi join和物化临时表导致的问题，应该如何优化呢？

先别急，做个小实验，执行SET optimizer_switch='semijoin=off'，也就是关闭掉半连接优化，此时执行EXPLAIN命令看一下此时的执行计划，发现此时会恢复为一个正常的状态。

就是有一个SUBQUERY的子查询，基于range方式去扫描索引搜索出4561条数据，接着有一个PRIMARY类型的主查询，直接是基于id这个PRIMARY主键聚簇索引去执行的搜索，然后再把这个SQL语句真实跑一下看看，发现性能一下子提升了几十倍，变成了100多毫秒！

因此到此为止，这个SQL的性能问题，真相大白，其实反而是他自动执行的semi join半连接优化，给咱们导致了问题，一旦禁止掉semi join自动优化，用正常的方式让他基于索引去执行，性能那是嗖嗖的。

当然，在生产环境是不能随意更改这些设置的，所以后来我们想了一个办法，多种办法尝试去修改SQL语句的写法，在不影响他语义的情况下，尽可能的去改变SQL语句的结构和格式，最终被我们尝试出了一个写法，如下所示：

SELECT COUNT(id)

FROM users

WHERE (id IN (SELECT user_id FROM users_extent_info WHERE latest_login_time < xxxxx) OR id IN (SELECT user_id FROM users_extent_info WHERE latest_login_time < -1))

在上述写法下，WHERE语句的OR后面的第二个条件，根本是不可能成立的，因为没有数据的latest_login_time是小于-1的，所以那是不会影响SQL语义的，但是我们发现改变了SQL的写法之后，执行计划也随之改变。

他并没有再进行semi join优化了，而是正常的用了子查询，主查询也是基于索引去执行的，这样我们在线上上线了这个SQL语句，性能从几十秒一下子就变成几百毫秒了。

希望大家能认真体会这个SQL调优案例里的方法，其实最核心的，还是看懂SQL的执行计划，然后去分析到底他为什么会那么慢，接着你就是要想办法避免他全表扫描之类的操作，一定要让他去用索引，用索引是王道，是最重要的！

End