113 案例实战：亿级数据量商品系统的SQL调优实战（2）

今天我们继续来分析这个案例，上次已经讲到，下面的这个商品系统让用户根据品类筛选商品的SQL语句

select * from products where category='xx' and sub_category='xx' order by id desc limit xx,xx

在一个亿级数据量的商品表里执行，需要耗时几十秒，结果导致了数据库的连接资源全部打满，商品系统无法运行，处于崩溃状态。

现在就得来分析一下，到底为什么会出现这样的一个情况，首先要给大家解释一下，这个表当时肯定是对经常用到的查询字段都建立好了索引的，那么针对这里简化后的SQL语句，你可以认为如下的一个索引，KEY index_category(catetory,sub_category)肯定是存在的，所以基本可以确认上面的SQL绝对是可以用上索引的。

因为如果你一旦用上了品类的那个索引，那么按品类和子类去在索引里筛选，其实第一，筛选很快速，第二，筛出来的数据是不多的，按说这个语句应该执行的速度是很快的，即使表有亿级数据，但是执行时间也最多不应该超过1s。

但是现在这个SQL语句跑了几十秒，那说明他肯定就没用我们建立的那个索引，所以才会这么慢，那么他到底是怎么执行的呢？我们来看一下他的执行计划：

explain select * from products where category='xx' and sub_category='xx' order by id desc limit xx,xx

此时执行计划具体内容就不写了，因为大家之前看了那么多执行计划，基本都很熟悉了，我就说这里最核心的信息，他的possible_keys里是有我们的index_category的，结果实际用的key不是这个索引，而是PRIMARY！！而且Extra里清晰写了Using where

到此为止，这个SQL语句为什么性能这么差，就真相大白了，他其实本质上就是在主键的聚簇索引上进行扫描，一边扫描，一边还用了where条件里的两个字段去进行筛选，所以这么扫描的话，那必然就是会耗费几十秒了！

因此此时为了快速解决这个问题，就需要强制性的改变MySQL自动选择这个不合适的聚簇索引进行扫描的行为

那么怎么改变呢？交给大家一个办法，就是使用force index语法，如下：

select * from products force index(index_category) where category='xx' and sub_category='xx' order by id desc limit xx,xx

使用上述语法过后，强制让SQL语句使用了你指定的索引，此时再次执行这个SQL语句，会发现他仅仅耗费100多毫秒而已！性能瞬间就提升上来了！

因此当时在紧急关头中，一下子就把这个问题给解决了，这里也是告诉大家这样的一个实战技巧，就是你如何去强制改变MySQL的执行计划，之前就有一个朋友来问我们说，面试官问我，如果MySQL使用了错误的执行计划，应该怎么办？

其实答案很简单，就是这个案例里的情况，方法就是force index语法就可以了。

但是这个案例还没完，这里还遗留了很多的问题，比如：

- 为什么在这个案例中MySQL默认会选择对主键的聚簇索引进行扫描？
- 为什么没使用index_category这个二级索引进行扫描？
- 即使用了聚簇索引，为什么这个SQL以前没有问题，现在突然就有问题了？

这都是一系列奇怪的问题，让我们对这个案例进行了深入的探究，下次，我们就来给大家分析这个案例背后的这些故事。

End