114 案例实战：亿级数据量商品系统的SQL调优实战（3）

今天我们来分析一下这个案例背后的一些事情，上次我们提到了一系列的问题，包括：

- 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？

关于这些问题，咱们得一步一步的来解决。首先，第一个问题，为什么针对：

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

这样一个SQL语句，MySQL要选择对聚簇索引进行扫描呢？

其实关于这个逻辑，说起来也并不是太复杂，因为大家都知道，这个表是一个亿级数据量的大表，那么对于他来说，index_category这个二级索引也是比较大的

所以此时对于MySQL来说，他有这么一个判断，他觉得如果要是从index_category二级索引里来查找到符合where条件的一波数据，接着还得回表，回到聚簇索引里去。

因为SQL语句是要select *的，所以这里必然涉及到一次回表操作，回到聚簇索引里去把所有字段的数据都查出来，但是在回表之前，他必然要做完order by id desc limit xx,xx这个操作

举个例子吧，比如他根据where category='xx' and sub_category='xx'，从index_category二级索引里查找出了一大波数据。

比如从二级索引里假设搂出来了几万条数据，接着因为二级索引里是包含主键id值的，所以此时他就得按照order by id desc这个排序语法，对这几万条数据基于临时磁盘文件进行filesort磁盘排序，排序完了之后，再按照limit xx,xx语法，把指定位置的几条数据拿出来，假设就是limit 0,10，那么就是把10条数据拿出来。

拿出来10条数据之后，再回到聚簇索引里去根据id查找，把这10条数据的完整字段都查出来，这就是MySQL认为如果你使用index_category的话，可能会发生的一个情况。

所以他担心的是，你根据where category='xx' and sub_category='xx'，从index_category二级索引里查出来的数据太多了，还得在临时磁盘里排序，可能性能会很差，因此MySQL就把这种方式判定为一种不太好的方式。

因此他才会选择换一种方式，也就是说，直接扫描主键的聚簇索引，因为聚簇索引都是按照id值有序的，所以扫描的时候，直接按order by id desc这个倒序顺序扫描过去就可以了，然后因为他知道你是limit 0,10的，也就知道你仅仅只要拿到10条数据就行了。

所以他在按顺序扫描聚簇索引的时候，就会对每一条数据都采用Using where的方式，跟where category='xx' and sub_category='xx'条件进行比对，符合条件的就直接放入结果集里去，最多就是放10条数据进去就可以返回了。

此时MySQL认为，按顺序扫描聚簇索引，拿到10条符合where条件的数据，应该速度是很快的，很可能比使用index_category二级索引那个方案更快，因此此时他就采用了扫描聚簇索引的这种方式！

那接下来我们又要考虑一个问题了，那就是这个SQL语句，实际上之前在线上系统运行一直没什么问题，也就是说，之前在线上系统而言，即使采用扫描聚簇索引的方案，其实这个SQL语句也确实一般都运行不慢，最起码是不会超过1s的。

那么为什么会在某一天晚上突然的就大量报慢查询，耗时几十秒了呢？

原因也很简单，其实就是因为之前的时候，where category='xx' and sub_category='xx'这个条件通常都是有返回值的，就是说根据条件里的取值，扫描聚簇索引的时候，通常都是很快就能找到符合条件的值以及返回的，所以之前其实性能也没什么问题。

但是后来可能是商品系统里的运营人员，在商品管理的时候加了几种商品分类和子类，但是这几种分类和子类的组合其实没有对应的商品

也就是说，那一天晚上，很多用户使用这种分类和子类去筛选商品，where category='新分类' and sub_category='新子类'这个条件实际上是查不到任何数据的！

所以说，底层在扫描聚簇索引的时候，扫来扫去都扫不到符合where条件的结果，一下子就把聚簇索引全部扫了一遍，等于是上亿数据全表扫描了一遍，都没找到符合where category='新分类' and sub_category='新子类'这个条件的数据。

也正是因为如此，才导致这个SQL语句频繁的出现几十秒的慢查询，进而导致MySQL连接资源打满，商品系统崩溃！

因此到此为止，这个案例就彻底分析清楚了，包括案例背后的故事也给大家讲明白了，其实SQL调优并没那么难，核心在于你一定要看懂SQL的执行计划，理解他为什么会慢，只要你理解了，就是想各种办法去解决，这个解决办法可能不是专栏可以讲完的，我们会提供几种经典的方案，但是往往需要你在发现问题的时候自己想办法，或者网上搜索。

比如我们上次讲到的第一个案例，就是通过禁用MySQL的半连接优化或者是改写SQL语句结构来避免自动半连接优化，第二个案例就得通过force index语法来强制某个SQL用我们指定的索引，这些都是属于比较经典的解决方案。

End