115 案例实战：数十亿数量级评论系统的SQL调优实战（1）

今天来给大家讲一个新的SQL调优案例，就是针对我们电商场景下非常普遍的商品评论系统的一个SQL优化，这个商品评论系统的数据量非常大，拥有多达十亿量级的评论数据，所以当时对这个评论数据库，我们是做了分库分表的，基本上分完库和表过后，单表的评论数据在百万级别。

每一个商品的所有评论都是放在一个库的一张表里的，这样可以确保你作为用户在分页查询一个商品的评论时，一般都是直接从一个库的一张表里执行分页查询语句就可以了

好，那么既然提到了商品评论分页查询的问题，我们就可以从这里开始讲我们的案例了。

大家都知道，在电商网站里，有一些热门的商品，可能销量多达上百万，商品的评论可能多达几十万条。然后呢，有一些用户，可能就喜欢看商品评论，他就喜欢不停的对某个热门商品的评论不断的进行分页，一页一页翻，有时候还会用上分页跳转功能，就是直接输入自己要跳到第几页去。

所以这个时候，就会涉及到一个问题，**针对一个商品几十万评论的深分页问题。**

先来看看一个经过我们简化后的对评论表进行分页查询的SQL语句：

SELECT * FROM comments WHERE product_id ='xx' and is_good_comment='1' ORDER BY id desc LIMIT 100000,20

这个SQL语句想必大家都知道是怎么回事。

其实他的意思就是，比如用户选择了查看某个商品的评论，因此必须限定Product_id，同时还选了只看好评，所以is_good_commit也要限定一下

接着他要看第5001页评论，那么此时limit的offset就会是(5001 - 1) * 20，其中20就是每一页的数量，此时起始offset就是100000，所以limit后100000,20

对这个评论表呢，最核心的索引就是一个，那就是index_product_id，所以对上述SQL语句，正常情况下，肯定是会走这个索引的，也就是说，会通过index_product_id索引，根据product_id ='xx'这个条件从表里先删选出来这个表里指定商品的评论数据。

那么接下来第二步呢？当然是得按照 is_good_comment='1' 条件，筛选出这个商品评论数据里的所有好评了！但是问题来了，这个index_product_id的索引数据里，并没有is_good_commet字段的值，所以此时只能很尴尬的进行回表了。

也就是说，对这个商品的每一条评论，都要进行一次回表操作，回到聚簇索引里，根据id找到那条数据，取出来is_good_comment字段的值，接着对is_good_comment='1'条件做一个比对，筛选符合条件的数据。

那么假设这个商品的评论有几十万条，岂不是要做几十万次回表操作？虽然每次回表都是根据id在聚簇索引里快速查找的，但还是架不住你每条数据都回表啊！！！

接着对于筛选完毕的所有符合WHERE product_id ='xx' and is_good_comment='1'条件的数据，假设有十多万条吧，接着就是按照id做一个倒序排序，此时还得基于临时磁盘文件进行倒序排序，又得耗时很久。

排序完毕了，就得基于limit 100000,20获取第5001页的20条数据，最后返回。

这个过程，因为有几十万次回表查询，还有十多万条数据的磁盘文件排序，所以当时发现，这条SQL语句基本要跑个1秒~2秒。

那么如何对他进行优化呢？其实这个思路，反而就跟我们讲的第二个案例反过来了，第二个案例中基于商品品类去查商品表，是尽量避免对聚簇索引进行扫描，因为有可能找不到你指定的品类下的商品，出现聚簇索引全表扫描的问题。

所以当时第二个案例里，反而就是选择强制使用一个联合索引，快速定位到数据，这个过程中因为不需要进行回表，所以效率还是比较高的

大家如果有印象的话，应该还记得第二个案例里，就是根据category和sub_category组成的联合索引进行查找，所以不需要回表，这就节省下了大量回表操作的耗时，所以当时我们选择了这个方案。

然后第二个案例中，接着直接根据id临时磁盘文件排序后找到20条分页数据，再回表查询20次，找到20条商品的完整数据。因此当时对第二个案例而言，因为不涉及到大量回表的问题，所以这么做基本是合适的，性能通常在1s以内。

但是我们这个案例里，就不是这么回事了，因为WHERE product_id ='xx' and is_good_comment='1'这两个条件，不是一个联合索引，所以必须会出现大量的回表操作，这个耗时是极高的。

因此对于这个案例，我们通常会采取如下方式改造分页查询语句：SELECT * from comments a, (SELECT id FROM comments WHERE product_id ='xx' and is_good_comment='1' ORDER BY id desc LIMIT 100000,20) b WHERE a.id=b.id

上面那个SQL语句的执行计划就会彻底改变他的执行方式，他通常会先执行括号里的子查询，子查询反而会使用PRIMARY聚簇索引，按照聚簇索引的id值的倒序方向进行扫描，扫描过程中就把符合WHERE product_id ='xx' and is_good_comment='1'条件的数据给筛选出来。

比如这里就筛选出了十万多条的数据，并不需要把符合条件的数据都找到，因为limit后跟的是100000,20，理论上，只要有100000+20条符合条件的数据，而且是按照id有序的，此时就可以执行根据limit 100000,20提取到5001页的这20条数据了。

接着你会看到执行计划里会针对这个子查询的结果集，一个临时表，进行全表扫描，拿到20条数据，接着对20条数据遍历，每一条数据都按照id去聚簇索引里查找一下完整数据，就可以了。

所以针对我们的这个场景，反而是优化成这种方式来执行分页，他会更加合适一些，他只有一个扫描聚簇索引筛选符合你分页所有数据的成本，你的分页深度越深，扫描数据越多，分页深度越浅，那扫描数据就越少，然后再做一页20条数据的20次回表查询就可以了。

当时我们做了这个分页优化之后，发现这个分页语句一下子执行时间降低到了几百毫秒了，此时就达到了我们优化的目的。

但是这里还是要给大家提醒一点，大家会发现，SQL调优实际上是没有银弹的，比如对于第二个案例来说，按顺序扫描聚簇索引方案可能会因为找不到数据导致亿级数据量的全表扫描，所以对第二个案例而言，必须得根据二级索引去查找。

但是对于我们这第三个案例而言，因为前提是做了分库分表，评论表单表数据一般在一百万左右，所以首先，他即使一个商品没有评论，有全表扫描，也绝对不会像扫描上亿数据表那么慢

其次，如果你根据product_id的二级索引查找，反而可能出现几十万次回表查询，所以二级索引查找方式反而不适合，而按照聚簇索引顺序扫描的方式更加适合。

简而言之，针对不同的案例，要具体情况具体分析，他慢，慢的原因在哪儿，为什么慢，然后再用针对性的方式去优化他。

End