当我们在SQL里进行分组的时候，如何才能使用索引？

今天我们接着上次的内容来谈谈在SQL语句里假设你要是用到了group by分组语句的话是否可以用上索引，因为大家都知道，有时候我们会想要做一个group by把数据分组接着用count sum之类的聚合函数做一个聚合统计。

那假设你要是走一个类似select count(*) from table group by xx的SQL语句，似乎看起来必须把你所有的数据放到一个临时磁盘文件里还有加上部分内存，去搞一个分组，按照指定字段的值分成一组一组的，接着对每一组都执行一个聚合函数，这个性能也是极差的，因为毕竟涉及大量的磁盘交互。

因为在我们的索引树里默认都是按照指定的一些字段都排序好的，其实字段值相同的数据都是在一起的，假设要是走索引去执行分组后再聚合，那性能一定是比临时磁盘文件去执行好多了。

所以通常而言，对于group by后的字段，最好也是按照联合索引里的最左侧的字段开始，按顺序排列开来，这样的话，其实就可以完美的运用上索引来直接提取一组一组的数据，然后针对每一组的数据执行聚合函数就可以了。

其实大家会发现，这个group by和order by用上索引的原理和条件都是差不多的，本质都是在group by和order by之后的字段顺序和联合索引中的从最左侧开始的字段顺序一致，然后就可以充分利用索引树里已经完成排序的特性，快速的根据排序好的数据执行后续操作了。

这样就不再需要针对杂乱无章的数据利用临时磁盘文件加上部分内存数据结构进行耗时耗力的现场排序和分组，那真是速度极慢，性能极差的。

所以学到这里，实际上大家应该已经理解了一点，那就是我们平时设计表里的索引的时候，必须充分考虑到后续你的SQL语句要怎么写，大概会根据哪些字段来进行where语句里的筛选和过滤？大概会根据哪些字段来进行排序和分组？

然后在考虑好之后，就可以为表设计两三个常用的索引，覆盖常见的where筛选、order by排序和group by分组的需求，保证常见的SQL语句都可以用上索引，这样你真正系统跑起来，起码是不会有太大的查询性能问题了。

毕竟只要你所有的查询语句都可以利用索引来执行，那么速度和性能通常都不会太慢。如果查询还是有问题，那就要深度理解查询的执行计划和执行原理了，然后基于执行计划来进行深度SQL调优。

然后对于更新语句而言，其实最核心的就是三大问题，一个是你索引别太多，索引太多了，更新的时候维护很多索引树肯定是不行的；一个是可能会涉及到一些锁等待和死锁的问题；一个就是可能会涉及到MySQL连接池、写redo log文件之类的问题。

所以接下来，我们会陆续讲解这些实战场景中最主要遇到的一些问题，先从查询这块的一些普通场景慢慢讲起，我们会下一讲说一下回表问题以及覆盖索引，接着就会基于电商的实际场景讲解一些案例，告诉大家如何设计索引保证查询性能别太差。

然后再讲解查询语句的执行计划以及深度SQL调优的原理以及一些实战案例，再接着讲解更新时候遇到的一些问题，包括索引、锁问题、写磁盘等等这些问题以及对应的实战案例，等大家把这些都学好之后，其实数据库日常的索引设计，查询和更新的优化，都能搞定了！

那么接着就可以进入下一步的数据库高阶场景的讲解了，包括数据库的备份和恢复，主从架构和读写分离，高可用架构，分库分表架构。

End