案例实战：陌生人社交APP的MySQL索引设计实战（4）

今天是咱们的这个索引设计案例的最后一篇文章，之前通过三篇文章的分析，相信大家都已经理解了为什么我们要把索引设计成**（province, city, sex, hobby, character, age）**这样的一个形式。

这么做其实关键是要让最频繁查询的一些条件都放到索引里去，然后在查询的时候如果有些字段是不使用的，可以用in (所有枚举值)的方式去写，这样可以让所有查询条件都用上你的索引，同时对范围查询的age字段必须放在最后一个，这样保证范围查询也能用上索引。

那么今天我们来研究下一个问题，**假设在查询的时候还有一个条件**，是要根据用户最近登录时间在7天之内来进行筛选，筛选最近7天登录过APP的用户，那么实际上可能你的用户表里有这么一个字段，latest_login_time

你要是在where条件里加入这么一个latest_login_time <= 7天内语句，肯定这个是没法用上索引了。因为你这里必然会用一些计算或者是函数，才能进行一些时间的比对。

而且假设你的查询里还有age进行范围查询，那么我们之前说过，范围查询的时候，也就只有第一个范围查询是可以用上索引的，第一个范围查询之后的其他范围查询是用不上索引的。

也就是说，即使你索引设计成这样：（province, city, sex, hobby, character, age, latest_login_time），然后你的where语句写成这样：where xx xxx and age>=xx and age<=xxx and latest_login_time>=xx，虽然age和latest_login_time都在联合索引里，但是按照规则，只有age范围查询可以用到索引，latest_login_time始终是用不到索引的。

所以此时有一个技巧可以教给大家，你在设计表的时候，就必须考虑到这个问题，此时你完全可以设计一个字段为：does_login_in_latest_7_days，也就是说，这个人是否在最近7天内登录过APP。

假设在7天内登录了这个APP，那么这个字段就是1，否则超过7天没登录，这个字段就是0！这样就把一个时间字段转换为了一个枚举值的字段。

接下来的解决方案就简单化了，可以设计一个联合索引为：**（province, city, sex, hobby, character, does_login_in_latest_7_days, age）**，然后搜索的时候，一定会在where条件里带上一个does_login_in_latest_7_days=1，最后再跟上age范围查询，这样就可以让你的where条件里的字段都用索引来筛选。

实际上一般来说，假设你要是where语句里通过上述联合索引就可以过滤掉大部分的数据，就保留小部分数据下来基于磁盘文件进行order by语句的排序，最后基于limit进行分页，那么一般性能还是比较高的。

但有时候又怕一个问题，就是说万一你要是就仅仅使用联合索引里一些基数特别小的字段来筛选呢？

比如就基于性别来筛选，比如一下子筛选出所有的女性，可能有上百万用户数据，接着还要磁盘文件进行排序再分页？那这个性能可能就会极为的差劲了！

所以针对上述问题，可以针对那种基数很低的字段再加上排序字段单独额外设计一个辅助索引，专门用于解决where条件里都是基数低的字段，然后还要排序后分页的问题，比如说就可以设计一个联合索引为：（sex, score）。

此时万一你要是写出如下SQL：select xx from user_info where sex='female' order by score limit xx,xx，此时假设用之前设计的那个联合索引，那绝对是完蛋了，因为根本没法用索引

但是用我们设计的那个辅助的（sex, score）索引呢？

此时因为where条件里的字段是等值匹配，而且还是等于某个常量值，所以虽然order by后跟的score字段是（sex, score）索引里的第二个字段，order by没有从索引最左侧字段开始排列，但是他也可以使用到索引来排序。

因为具体到使用索引的层面，他会先对where条件里的sex='female'在索引树里筛选到这部分数据，接着在sex='female'的数据里，这些数据实际上都是排列在一起的，因为在索引里，会按照sex和score两个字段去进行排序，所以sex='female'的数据都是在一块儿的。

然后找到这部分数据之后，接着就可以确定，这部分数据肯定是按照score字段进行排序的，此时就可以按照score字段值的顺序，去读取你的limit语句指定的数据分页出来就可以了

所以此时你这种针对sex低基数的字段的筛选和基于评分排序的语句，整体运行的效率是非常高的，完全可以基于辅助索引来实现。

以此类推，完全可以通过对查询场景的分析，用**（province, city, sex, hobby, character, does_login_in_latest_7_days, age）**这样的联合索引去抗下复杂的where条件筛选的查询，此时走索引筛选速度很快，筛选出的数据量较少，接着进行排序和limit分页。

同时针对一些**低基数字段筛选+评分排序**的查询场景，可以设计类似（sex, score）的**辅助索引**来应对，让他快速定位到一大片低基数字段对应的数据，然后按照索引顺序去走limit语句获取指定分页的数据，速度同样会很快。

通过最近这个案例的分析，想必大家能够感悟到一些针对具体的查询场景来设计你的联合索引和辅助索引的技巧

核心重点就是，尽量利用一两个复杂的多字段联合索引，抗下你80%以上的 查询，然后用一两个辅助索引抗下剩余20%的非典型查询，保证你99%以上的查询都能充分利用索引，就能保证你的查询速度和性能！

End