mysql的索引

题目难度：★★★

知识点标签：mysql

学习时长：15分钟

题目描述

MySQL B+树索引和哈希索引的区别

介绍

1. 数据索引的存储是有序的
2. 在有序的情况下，通过索引查询一个数据是无需遍历索引记录的
3. 极端情况下，数据索引的查询效率为二分法查询效率，趋近于 log2(N)

在MySQL里常用的索引数据结构有B+树索引和哈希索引两种，我们来看下这两种索引数据结构及其不同的应用建议。

B-Tree 索引

B-Tree索引是用B-Tree数据结构来存储数据的。大多数mysql引擎都支持这种索引，也是用的比较多的一种它每一个叶子节点都包含指向下一个叶子节点的指针，从而方便叶子节点的范围遍历。

为什么B-Tree索引能加快访问的速度?

因为存储引擎不再需要进行全表扫描来获取需要的数据，取而代之的是从索引的根节点开始进行搜索。根节点的槽中存放指向子节点的指针，存储引擎根据这些指针向下层查找。通过比较节点页的值和要查找的值，可以找到合适的指针进入下层子节点。

什么时候用B-Tree索引

B-Tree索引适用于全键值、键值范围或者键前缀查找。其中键前缀查找只适用于最左前缀的查找。因为B-Tree是顺序组织存储的，所以很适合查找范围数据。它对以下类型有效：

1. 全值匹配

> 全值匹配指的是和索引中所有列进行匹配，比如name 和 age 都有索引，查询name=xxx and age=xx的

2. 匹配最左前缀的查询

> 只能匹配索引的第一列，如联合索引，只有左边的有效。

3. 匹配列前缀

> 只匹配某一列的值得开头部分，例如查找所有以A开头的名字的人。

4. 匹配范围值

> 查询 价格 1-1000之间的值。

5. 精确匹配左前列并范围匹配另外一列

> 就是查找名称为A的人，并且名字开头是B的。即第一列全匹配，第二列范围匹配。

因为索引树的节点是有序的，所以除了按值查找之外，索引还可以用于查询中的ORDER BY操作(按顺序查找)，如果ORDER BY子句满足前面列出的几种查询类型，则这个索引也可以满足对应的排序需求。

btree索引使用限制

1、只能按照索引的最左列开始查找。
2、如果查询中有某个列的范围查询，则其右边所有列都无法使用索引。就是 like 后面的列全部无法使用索引
3、不能跳过索引中的列。也就是说不能查询名称为A 并且年龄为B的数据。

这时候实际上C(服务端)才是房东。

B(代理)是中介把这个房子租给了A(客户端)。这个过程中A(客户端)并不知道这个房子到底谁才是房东

他都有可能认为这个房子就是B(代理)的由上的例子和图我们可以知道正向代理和反向代理的区别在于代理的对象不一样,正向代理的代理对象是客户端,反向代理的代理对象是服务端。

哈希索引

哈希索引基于哈希表实现，只有，只有精确匹配索引所有列的查询才有效，对于每一行数据存储引擎都会对所有的索引列计算一个哈希码。在MySQL中，只有Memory引擎显示的支持哈希索引。因为哈希索引自身只需存储对应的哈希值。所以索引的结构十分紧凑，这也让哈希索引查找的速度非常快。然而哈希索引也有的限制。

1. 哈希索引只包含哈希值和行指针，而不存储字段值，所以不能使用索引中的值来避免读取行。
2. 哈希索引数据并不是按照索引值顺序存储的，所以也就无法用于排序。
3. 哈希索引也不支持部分索引列匹配查找，因为哈希索引始终是使用索引列的全部内容来计算哈希值的。
4. 哈希索引只支持等值比较查询。
5. 如果哈希冲突很多的话，一些索引维护操作的代价会很高。

因为这些限制，哈希索引只适用于某些特定的场合，而一旦适合哈希索引，则它带来的性能提升将非常显著。

还有一些空间数据索引、全文索引就不多做概述。

B+树索引和哈希索引的明显区别是：

如果是等值查询，那么哈希索引明显有绝对优势，因为只需要经过一次算法即可找到相应的键值；当然了，这个前提是，键值都是唯一的。如果键值不是唯一的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据；

从示意图中也能看到，如果是范围查询检索，这时候哈希索引就毫无用武之地了，因为原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索；

同理，哈希索引也没办法利用索引完成排序，以及like ‘xxx%’ 这样的部分模糊查询（这种部分模糊查询，其实本质上也是范围查询）；

哈希索引也不支持多列联合索引的最左匹配规则；

B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题。

总结：

在MySQL中，只有HEAP/MEMORY引擎表才能显式支持哈希索引（NDB也支持，但这个不常用），InnoDB引擎的自适应哈希索引（adaptive hash index）不在此列，因为这不是创建索引时可指定的。

还需要注意到：HEAP/MEMORY引擎表在mysql实例重启后，数据会丢失。

通常，B+树索引结构适用于绝大多数场景，像下面这种场景用哈希索引才更有优势：

在HEAP表中，如果存储的数据重复度很低（也就是说基数很大），对该列数据以等值查询为主，没有范围查询、没有排序的时候，特别适合采用哈希索引

例如这种SQL：
SELECT ... FROM t WHERE C1 = ?; — 仅等值查询

在大多数场景下，都会有范围查询、排序、分组等查询特征，用B+树索引就可以了。