Mysql 二三小疑速记

作者：mulyzhou
原文链接：https://ld246.com/article/1604580107943
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

[TOC]

什么是索引?

索引是一种能帮助 MySQL 提高查询效率的数据结构。

索引分别有哪些优点和缺点?

索引的优点如下:

- 快速访问数据表中的特定信息，提高检索速度。
- 创建唯一性索引，保证数据表中每一行数据的唯一性。
- 加速表与表之间的连接。
- 使用分组和排序进行数据检索时，可以显著减少查询中分组和排序的时间。

索引的缺点:

- 虽然提高了的查询速度，但却降低了更新表的速度，比如 update、insert，因为更新数据时，MyS L 不仅要更新数据，还要更新索引文件;
- 建立索引会占用磁盘文件的索引文件。

使用索引注意事项:

- 使用短索引，短索引不仅可以提高查询速度，更能节省磁盘空间和 I/O 操作;
- 索引列排序，MySQL 查询只使用一个索引，因此如果 where 子句中已经使用了索引的话，那么 or er by 中的列是不会使用索引的，因此数据库默认排序可以符合要求的情况下，不要进行排序操作;量不要包含多个列的排序，如果需要最好给这些列创建复合索引;
- like 语句操作，一般情况下不鼓励使用 like 操作，如果非使用不可， 注意 like "%aaa%" 不会使用

引，而like "aaa%"可以使用索引；

- 不要在列上进行运算；
- 不适用 NOT IN 和 <> 操作。

索引有几种类型？分别如何创建？

MySQL 的索引有两种分类方式：逻辑分类和物理分类。 按照逻辑分类，索引可分为：

- 主键索引：一张表只能有一个主键索引，不允许重复、不允许为 NULL；
- 唯一索引：数据列不允许重复，允许为 NULL 值，一张表可有多个唯一索引，但是一个唯一索引只包含一列，比如身份证号码、卡号等都可以作为唯一索引；
- 普通索引：一张表可以创建多个普通索引，一个普通索引可以包含多个字段，允许数据重复，允许ULL 值插入；
- 全文索引：让搜索关键词更高效的一种索引。

按照物理分类，索引可分为：

- 聚集索引：一般是表中的主键索引，如果表中没有显示指定主键，则会选择表中的第一个不允许为ULL 的唯一索引，如果还是没有的话，就采用 Innodb 存储引擎为每行数据内置的 6 字节 ROWID 为聚集索引。每张表只有一个聚集索引，因为聚集索引的键值的逻辑顺序决定了表中相应行的物理顺。聚集索引在精确查找和范围查找方面有良好的性能表现（相比于普通索引和全表扫描），聚集索引显得弥足珍贵，聚集索引选择还是要慎重的（一般不会让没有语义的自增 id 充当聚集索引）；
- 非聚集索引：该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同（非主键的那一列），一个中可以拥有多个非聚集索引。

各种索引的创建脚本如下：

```
-- 创建主键索引
alter table t add primary key add (`id`);
-- 创建唯一索引
alter table t add unique (`username`);
-- 创建普通索引
alter table t add index index_name (`username`);
-- 创建全文索引
alter table t add fulltext (`username`);
```

主索引和唯一索引有什么区别？

- 主索引不能重复且不能为空，唯一索引不能重复，但可以为空；
- 一张表只能有一个主索引，但可以有多个唯一索引；
- 主索引的查询性能要高于唯一索引。

因为普通索引的查询会多执行一次检索操作。比如主键查询 select * from t where id=10 只需要搜索 d 的这棵 B+ 树，而普通索引查询 select * from t where f=3 会先查询 f 索引树，得到 id 的值之后去搜索 id 的 B+ 树，因为**多执行了一次检索**，所以执行效率就比主键索引要低。

什么叫回表查询？

普通索引查询到主键索引后，回到主键索引树搜索的过程，我们称为回表查询。

参考SQL：

```
mysql> create table T(
id int primary key,
k int not null,
name varchar(16),
index (k))engine=InnoDB;
```

如果语句是 select * from T where ID=500，即主键查询方式，则只需要检索主键 ID 字段。

```
mysql>  select * from T where ID=500;
+-----+---+-------+
| id  | k | name  |
+-----+---+-------+
| 500 | 5 | name5 |
+-----+---+-------+
```

如果语句是 select * from T where k=5，即普通索引查询方式，则需要先搜索 k 索引树，得到 ID 值为 500，再到 ID 索引树搜索一次，这个过程称为回表查询。

```
mysql> select * from T where k=5;
+-----+---+-------+
| id  | k | name  |
+-----+---+-------+
| 500 | 5 | name5 |
+-----+---+-------+
```

也就是说，基于非主键索引的查询需要多扫描一棵索引树。因此，我们在应用中应该尽量使用主键查。

索引的常见存储算法有哪些?

- 哈希存储法：以 key、value 方式存储，把值存入数组中使用哈希值确认数据的位置，如果发生哈冲突，使用链表存储数据；
- 有序数组存储法：按顺序存储，优点是可以使用二分法快速找到数据，缺点是更新效率，适合静态据存储；
- 搜索树：以树的方式进行存储，查询性能好，更新速度快。

InnoDB 为什么要使用 B+ 树，而不是 B 树、Hash、红黑树或二叉树?

因为 B 树、Hash、红黑树或二叉树存在以下问题：

- B 树：不管叶子节点还是非叶子节点，都会保存数据，这样导致在非叶子节点中能保存的指针数量少（有些资料也称为扇出），指针少的情况下要保存大量数据，只能增加树的高度，导致IO操作变多查询性能变低；
- Hash：虽然可以快速定位，但是没有顺序，IO 复杂度高；
- 二叉树：树的高度不均匀，不能自平衡，查找效率跟数据有关（树的高度），并且 IO 代价高；
- 红黑树：树的高度随着数据量增加而增加，IO 代价高。

为什么 InnoDB 要使用 B+ 树来存储索引?

B+Tree 中的 B 是 Balance，是平衡的意思，它在经典 B Tree 的基础上进行了优化，增加了顺序访指针，在B+Tree 的**每个叶子节点增加一个指向相邻叶子节点的指针**，就形成了**带有顺序访问指针**的+Tree，这样就提高了区间访问性能：如果要查询 key 为从 18 到 49 的所有数据记录，当找到 18 后只需顺着节点和指针顺序遍历就可以一次性访问到所有数据节点，极大提到了区间查询效率（无需返上层父节点重复遍历查找减少 IO 操作）。

索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上，这样的，索引查找过程中就要产生磁盘 IO 消耗，相对于内存存取，IO 存取的消耗要高几个数量级，所以索的结构组织要尽量**减少查找过程中磁盘 IO 的存取次数**，从而提升索引效率。 综合所述，InnDB 只有取 B+ 树的数据结构存储索引，才能提供数据库整体的操作性能。

B+树的特征：

1.有k个子树的中间节点包含有k个元素（B树中是k-1个元素），每个元素不保存数据，只用来索引，有数据都保存在叶子节点。

2.所有的叶子结点中包含了全部元素的信息，及指向含这些元素记录的指针，且叶子结点本身依关键的大小自小而大顺序链接。

3.所有的中间节点元素都同时存在于子节点，在子节点元素中是最大（或最小）元素。

B+树的优势：

1.单一节点存储更多的元素，使得查询的IO次数更少。

2.所有查询都要查找到叶子节点，查询性能稳定。

3.所有叶子节点形成有序链表，便于范围查询。