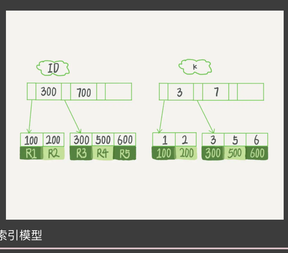
MySQL索引

作用

提高数据查询效率

常见模型

- 哈希表：哈希表这种结构适用于只有等值查询的场景
- 有序数组：有序数组在等值查询和范围查询场景中的性能就都非常优秀，但有序数组索引只适用于静态存储引擎
- 搜索树：在读写上的性能优点，以及适配磁盘的访问模式



聚簇索引

索引模型

B+ 树索引模型

B+ 树能够很好地配合磁盘的读写特性，减少单次查询的磁盘访问次数。

主键长度越小，普通索引的叶子节点就越小，普通索引占用的空间也就越小。

索引类型

- 主键索引：又称为聚簇索引
  - 聚簇索引是一种数据的组织方式，严格意义上来说并不属于索引的范畴。InnoDB 聚簇索引使用 B+Tree 将主键和数据组织在一起，并且仅将数据保存在叶子节点，同时 B+Tree 的节点在 MySQL 中称之为页（Page），一页默认为 16KB
- 非主键索引：又称为二级索引
  - 又称之为辅助索引，是日常使用最为频繁的索引结构，二级索引中并不保存行数据，仅包含索引数据，以及叶子节点中保存的主键

基于主键索引和普通索引的查询有什么区别？

- 主键查询方式，则只需要搜索 ID 这棵 B+ 树；
- 普通索引查询方式，则需要先搜索 k 索引树，得到 ID 的值为 500，再到 ID 索引树搜索一次。这个过程称为回表。

关于主键

首先，对于 InnoDB 存储引擎的一张表来说，我们可以在创建 table 的时候不指定 primary key，但是这并不代表生成的表结构中没有聚簇索引

聚簇索引组织方式的选择

1. 若 table 中指定了主键，那么主键将作为聚簇索引的组织方式
2. 若 table 中未指定主键，那么 InnoDB 会选择一个 UNIQUE NOT NULL 的非空唯一索引作为聚簇索引的组织方式
3. 若 table 中既未指定主键，同时也没有非空唯一索引的话，InnoDB 会生成一个隐藏的主键

总之，InnoDB 一定会找一个东西来组织聚簇索引，这是 InnoDB 运行的根本

B+Tree 这棵 N 叉树的 N 和什么有关系？

N 和 InnoDB Page Size 以及索引的长度有关，Page Size 越小，N 越小；索引长度越长，也会导致 N 越小

索引维护

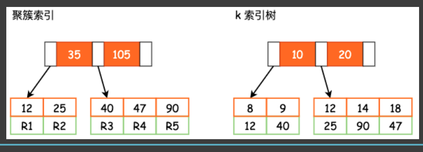
B+ 树为了维护索引有序性，在插入新值的时候需要做必要的维护

以上面这个图为例，如果插入新的行 ID 值为 700，则只需要在 R5 的记录后面插入一个新记录。如果新插入的 ID 值为 400，就相对麻烦了，需要逻辑上挪动后面的数据，空出位置。

而更糟的情况是，如果 R5 所在的数据页已经满了，根据 B+ 树的算法，这时候需要申请一个新的数据页，然后挪动部分数据过去。这个过程称为页分裂。在这种情况下，性能自然会受影响。

除了性能外，页分裂操作还影响数据页的利用率。原本放在一个页的数据，现在分到两个页中，整体空间利用率降低大约 50%。

当然有分裂就有合并。当相邻两个页由于删除了数据，利用率很低之后，会将数据页做合并。合并的过程，可以认为是分裂过程的逆过程。



普通索引与唯一索引

select * from T where k = 12;

查询

- 普通索引：顺着 k 索引树查找主键 ID，当找到 k = 12 这一条数据时，还需要向后查找，判断是否有其他数据也为 12
- 唯一索引：顺着 k 索引树查找主键 ID，当找到 k = 12 这一条数据时直接返回，因为 k 是 unique 的，不可能出现重复

两者效率几乎相同，因为 B+Tree 叶子节点是 page，所以会有相邻的特性，即使向后多找几条数据，大概率也是纯内存操作

更新

Change Buffer

当我们更新一条数据时，如果该数据的 page 在内存中则直接进行更新。若该数据页不在内存中，且不会影响数据一致性的前提下，InnoDB 会将这个修改写入至 change buffer 中（change buffer 随 redo log 持久化），后续再读取该行数据时，将数据读入内存，然后执行 page 和 change buffer 的 merge 操作

使用 change buffer 明显可以加快 update 语句的执行，不需要将 page 从硬盘中读取至内存中

- 普通索引：在更新数据时，如果更新的页不在内存中的话，由于不需要进行唯一性检查，所以直接写入到 change buffer 中，持久化交给 redo log 来做
- 唯一索引：在更新数据时，如果更新的页不在内存中的话，由于需要进行唯一性检查，所以必须把 page 读到内存中，判断后进行操作

在 page 不在内存中的情况，普通索引的更新效率要高于唯一索引的更新效率

新增：在新增数据时，由于主键是唯一的，因此对聚簇索引的写入必然需要将页面读入内存中。但是如果还存在其它索引写入的话，那么非唯一索引也可以使用 change buffer 来优化

综上，如果业务端能够保证数据的唯一性，并且是写多读少，那么使用普通索引比唯一索引会有更高的性能

覆盖索引

如果查询条件使用的是普通索引（或是联合索引的最左原则字段），查询结果是联合索引的字段或是主键，不用回表操作，直接返回结果，减少IO磁盘读写读取正行数据

如果执行的语句是 select ID from T where k between 3 and 5，这时只需要查 ID 的值，而 ID 的值已经在 k 索引树上了，因此可以直接提供查询结果，不需要回表。也就是说，在这个查询里面，索引 k 已经“覆盖了”我们的查询需求，我们称为覆盖索引。

覆盖索引可以减少树的搜索次数，显著提升查询性能

总之，覆盖索引的本质就是只查询一棵 B+Tree 即可找到我们所需要的数据

联合索引

最左前缀原则

B+ 树这种索引结构，可以利用索引的“最左前缀”，来定位记录。

建立联合索引的时候，如何安排索引内的字段顺序。

- 第一原则是，如果通过调整顺序，可以少维护一个索引，那么这个顺序往往就是需要优先考虑采用的。
- 空间

根据创建联合索引的顺序，以最左原则进行where检索，比如（age，name）以age=1 或 age= 1 and name='张三'可以使用索引，单以name='张三' 不会使用索引

一言以蔽之，最左匹配原则就是匹配联合索引的前 N 个字段，或者是字符串索引的前 M 个字符

索引下推

可以在索引遍历过程中，对索引中包含的字段先做判断，直接过滤掉不满足条件的记录，减少回表次数

like 'hello%'and age >10 检索，（name,age) 索引，MySQL5.6版本之前，会对匹配的数据进行回表查询。5.6版本后，会先过滤掉age<10的数据，再进行回表查询，减少回表率，提升检索速度