MySQL索引

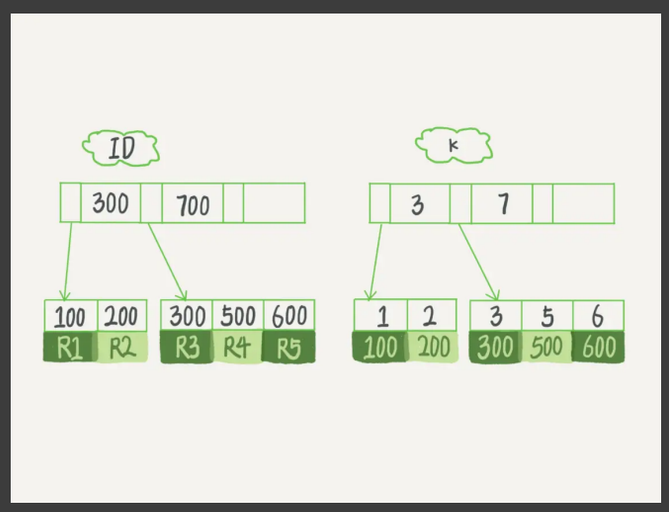
作用

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

InnoDB 的索引模型



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。
- 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。
- 索引类型
  - 主键索引
  - 非主键索引

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。

联合索引

- 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
- 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
- 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