B 树与 B+树在数据库的使用

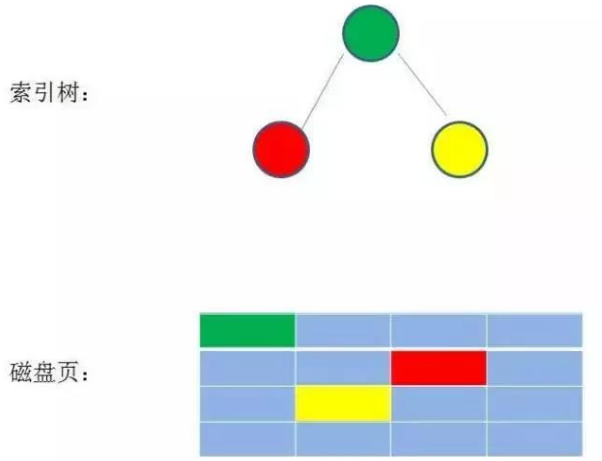
B 树

从算法逻辑上讲，二叉查找树的查找速度和比较速度都是最小的。

但是我们不得不考虑一个现实问题：磁盘 IO

数据库索引是存储在磁盘上的，当数据量比较大的时候，索引的大小或许有几 G 甚至更多

在使用索引时，不一定能将全部索引加载入内存。能做的只有逐一加载每一个磁盘页，这里的磁盘页对应着索引树的节点。



每次顺着节点搜索，都可能将触发一次磁盘 IO。所以在最坏的情况下，**磁盘的 IO 次数等于索引树的高度**。

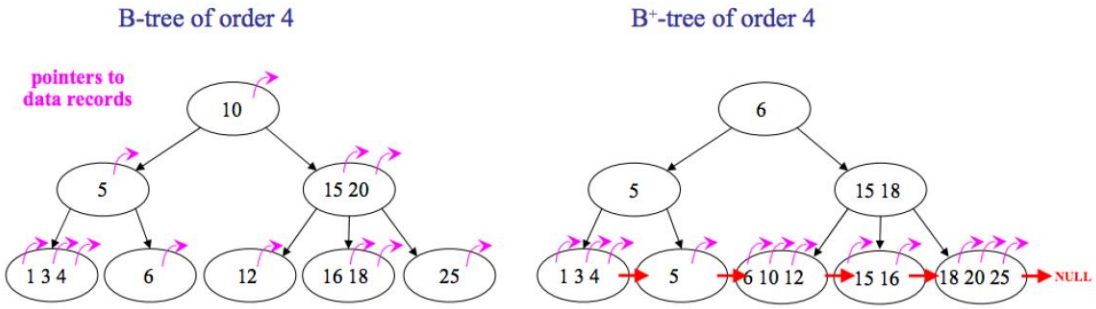
在这种情况下，为了减少磁盘 IO，将树变得“矮胖”会获得更好的性能。

在整个流程中，B 树的比较次数其实比二叉树多，尤其是在某一节点上元素比较多的时候哦。但相对磁盘 IO 的速度，内存中数据比较的时间几乎可以忽略。因此树的高度足够低即可通过减少 IO 次数来提升查找性能。

相比之下内部节点元素多一些影响也不大，只是多了几次内存交互，不超过磁盘页大小即可。

B+树

- A B⁺-tree can be viewed as a B-tree in which each node contains only keys (not pairs), and to which an additional level is added at the bottom with linked leaves



区别有以下两点：

1. B+树中只有叶子节点会带有指向记录的指针（ROWID），而 B 树则所有节点都带有，在内部节点出现的索引项不会再出现在叶子节点中。

2. B+树中所有叶子节点都是通过指针连接在一起，而 B 树不会。

B+树的优点：

1. 非叶子节点不会带上 ROWID，这样，**一个块中可以容纳更多的索引项**，一是可以降低树的高度。二是一个内部节点可以定位更多的叶子节点。

2. 叶子节点之间通过指针来连接，**范围扫描**将十分简单，而对于 B 树来说，则需要在叶子节点和内部节点不停的往返移动。

数据库索引采用 B+树的主要原因是: **B 树在提高了 IO 性能的同时并没有解决元素遍历**效率低下的问题，正是为了解决这个问题，B+树应用而生。B+树只需要去遍历叶子节点就可以实现整棵树的遍历。而且在**数据库中基于范围的查询是非常频繁的**，而 B 树不支持这样的操作或者说效率太低。

因为 B+ Tree 的有序性，所以除了用于查找，还可以用于排序和分组。