btree

B+树是B树的一种变体，有着比B树更高的查询性能

简介

1.B树

二叉查找树的时间复杂度最好情况为O(logN)，最差情况为O(N)，最差情况是所有的数据退化成链表，为了避免最差情况出现，需要让**二叉查找树保持平衡**，比如：

如上图中，22左边的孩子都比22小，而其左边孩子13，17按顺序排放，中间的孩子在22和35之间，右边的孩子比35大。

如上图所示的树称为B树(或B-树、B_树)，它是一种m阶平衡多叉树。当m取2时，便是二叉搜索树，其中m指的是一个结点最多有多少个孩子结点(度)。

对于m阶B树，其具有如下性质：

- 根结点至少有两个子女；
- 每个结点的值的个数为 1 <= n < m；
- 所有的叶子结点都位于同一层；
- 除根结点以外的所有结点（不包括叶子结点）的孩子正好是值个数的加1；
- 每个结点中的值都按照从小到大的顺序排列，每个值的左子树中的所有的值都小于它，而右子树中的所有的值都大于它。

下面来看一个更加清晰的B树：

上图为3阶B树(m=3)，在实际应用中的B树的阶数m都非常大（通常大于100），所以即使存储大量的数据，B树的高度仍然比较小，这有利于树的插入删除。在数据库中我们将B树（和B+树）作为索引结构，可以加快查询速速。

B树的插入

如果插入的结点只有一个数值，直接在该结点插入即可。例如，在上图中插入9，则直接在10结点前面插入9即可。但如果插入44，此时便需要通过结点的向上分裂来完成插入。

插入44：

发现此结点有3个值，不满足3阶B树，因此要进行分裂，将中间的40向上结点移动：

分裂后此B树变成了4阶B树，不满足3阶B树条件，原因是40移动到上结点所致，因此继续向上结点移动，将50移动到上节点：

此时发现又出现3个值的结点，继续进行分裂：

此时便满足条件，完成。

B树的删除按照插入的方法反过来操作即可，即父结点(如果不符合父结点大于左结点小于右结点的条件，则与上层父节点位置调换，直到符合条件为止)不断下移合并，知道符合条件为止。

2、B+树

B+树是B树的一种变体，有着比B树更高的查询性能、B+树和B树除了有一些共同特点外，还有一些新的特点：

- 有k个子树的中间结点包含有k个元素（B树中是k-1个元素），结点只是索引不保存数据，所有数据都保存在叶子节点。
- 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字的大小自小而大顺序链接。
- 所有的中间结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。

下面我们使用数值来表示一棵B+树：

由上图可以看到，B+树的每个结点的最大或最小元素都出现在下一个结点的首或尾。在B+树中，只有叶子节点存储数据，其它中间结点全部是索引。在数据库的聚集索引中，叶子节点直接包含数据库中某一行数据。在非聚集索引中，叶子节点带有指向数据库行的指针。

B+树的查找

B+树的查找有两种方式：从最小值进行顺序查找；从根结点开始，进行随机查找。在查找时，若非终端结点上的关键值等于给定值，并不终止，而是继续向下直到叶子结点（因为叶子结点才存数据）。因此，在B+树中，不管查找成功与否，每次查找都是走了一条从根到叶子结点的路径。其余同B-树的查找类似。

由于B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是B树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以B+树更加适合在区间查询的情况，所以通常B+树用于数据库索引，而B树则常用于文件索引。

B+树的插入

假设我们要向上图插入0，发现没有破坏B+树结构，直接在1，2结点处插入即可。

如果在结点的中间插入并破坏了B+树的结构：

但是如果我们要插入12，则发现破坏了B+树的结构，则：

分裂破坏了结构的结点，并将12移到上结点：

插入完毕。

如果在端点处插入并破坏了B+树的结构：

假如插入16：

分裂后，父结点要配合子结点的端点值：

删除操作，只需将插入操作进行反向操作即可。读者可以想想如何删除16。

B+数的优势：

- 单一节点存储更多的元素，使得查询的IO次数更少。（应用于文件系统、数据库系统）
- 所有查询都要查找到叶子节点，查询性能稳定。
- 所有叶子节点形成有序链表，便于范围查询。

参考地址