B树和B+树

[https://www.cnblogs.com/nullzx/p/8729425.html#top](https://www.cnblogs.com/nullzx/p/8729425.html" \l "top)

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。每个结点中存储了关键字（key）和关键字对应的数据（data），以及孩子结点的指针。我们将一个key和其对应的data称为一个记录。但为了方便描述，除非特别说明，后续文中就用key来代替（key, value）键值对这个整体。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，此时B树中的key就表示键，而data表示了这个键对应的条目在硬盘上的逻辑地址。

1.插入操作是指插入一条记录，即（key, value）的键值对。如果B树中已存在需要插入的键值对，则用需要插入的value替换旧的value。若B树不存在这个key,则一定是在叶子结点中进行插入操作。

2.删除操作是指，根据key删除记录，如果B树中的记录中不存对应key的记录，则删除失败。

1）如果当前需要删除的key位于非叶子结点上，则用后继key（这里的后继key均指后继记录的意思）覆盖要删除的key，然后在后继key所在的子支中删除该后继key。此时后继key一定位于叶子结点上，这个过程和二叉搜索树删除结点的方式类似。删除这个记录后执行第2步

2）该结点key个数大于等于Math.ceil(m/2)-1，结束删除操作，否则执行第3步。

3）如果兄弟结点key个数大于Math.ceil(m/2)-1，则父结点中的key下移到该结点，兄弟结点中的一个key上移，删除操作结束。

否则，将父结点中的key下移与当前结点及它的兄弟结点中的key合并，形成一个新的结点。原父结点中的key的两个孩子指针就变成了一个孩子指针，指向这个新结点。然后当前结点的指针指向父结点，重复上第2步。

有些结点它可能即有左兄弟，又有右兄弟，那么我们任意选择一个兄弟结点进行操作即可。

B+树：维基百科上所定义的方式，即关键字个数比孩子结点个数小1，这种方式是和B树基本等价的。---有多种定义方式，可以着重说中间节点只有索引不带数据，数据都在叶子节点。

除此之外B+树还有以下的要求。

1）B+树包含2种类型的结点：内部结点（也称索引结点）和叶子结点。根结点本身即可以是内部结点，也可以是叶子结点。根结点的关键字个数最少可以只有1个。

2）B+树与B树最大的不同是内部结点不保存数据，只用于索引，所有数据（或者说记录）都保存在叶子结点中。

3） m阶B+树表示了内部结点最多有m-1个关键字（或者说内部结点最多有m个子树），阶数m同时限制了叶子结点最多存储m-1个记录。

4）内部结点中的key都按照从小到大的顺序排列，对于内部结点中的一个key，左树中的所有key都小于它，右子树中的key都大于等于它。叶子结点中的记录也按照key的大小排列。

5）每个叶子结点都存有相邻叶子结点的指针，叶子结点本身依关键字的大小自小而大顺序链接。

## **B+树的优势**

<https://yq.aliyun.com/articles/625869>#

1. **单一节点存储更多元素**。B+树中间节点没有卫星数据（也就是说只包含索引信息），所以**每个非叶子节点可以包含更多的内容，同样大小的磁盘页可以容纳更多的节点元素**。也就是说B+树会在相同数据量的情况下比B树更加“矮胖”，查询的IO次数更少。
2. **查询效率稳定**。B+树的查询必须最终找到叶子节点，而B树如果在中间节点找到匹配的即可（最好情况是只查根节点，最差是查到叶子节点），而B+树每一次都是稳定的。**B-树的好处是，虽然查询性能不稳定，但平均的查询速度快一些**。试想一个数据库的查询，有时候执行10毫秒，有时候执行100毫秒，肯定是不太合适的。还不如每次都执行30毫秒。
3. **范围查询简便**。B树的范围查询只能依靠繁琐的中序遍历，找到下限和上限。而**B+树的范围查询很简单，只需要在叶子节点那一层的链表上做遍历就行**。

目前**大部分数据库系统和文件系统都采用B树或者变种的B+树来作为索引结构**。

## **为什么MySQL数据库中使用B+树**

**1.局部性原理与磁盘预读**  
由于磁盘的存取速度与内存之间鸿沟,为了提高效率,要尽量减少磁盘I/O.磁盘往往不是严格按需读取，而是每次都会预读,磁盘读取完需要的数据,会顺序向后读一定长度的数据放入内存。而这样做的理论依据是计算机科学中著名的局部性原理：

当一个数据被用到时，其附近的数据也通常会马上被使用，程序运行期间所需要的数据通常比较集中

由于磁盘顺序读取的效率很高(不需要寻道时间，只需很少的旋转时间)，因此对于具有局部性的程序来说，预读可以提高I/O效率.预读的长度一般为页(page)的整倍数。

**2.数据库索引采用B+树的主要原因**  
根据上面的局部性原理和磁盘预读，B树中用了这个技巧：每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个结点只需一次I/O。  
B树在提高了IO性能的同时并没有解决元素遍历的效率低下的问题，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**（比如查询某段时间之内的数据）的，而B树不支持这样的操作或者说**效率太低**（标红为效率低的原因）。

**带权路径长度最小**的树就叫最优二叉树，也就是哈夫曼树。要使带权路径长度最小，那么权值大的点就应该离根节点越近。

堆是一个完全二叉树，大顶堆就是每个节点都不大于它的父节点。  
插入和删除时间复杂度都是O(logn)。

红黑树的插入、删除、查找最坏时间复杂度都是O(logn)。红黑树是一种平衡二叉查找树。

主要用于存储有序的数据，它的时间复杂度是O(logn)，非常高效  
此外附加特性还有：

1.节点是红色或黑色。

2.根节点是黑色。

3.每个叶子节点都是黑色的空节点（NIL节点）。

4 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

5.从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

上述规则限制保证了红黑树的自平衡。从根到叶子的最长路径不会超过最短路径的2倍。

JDK的TreeMap和TreeSet底层就是红黑树实现的，Java8中的HashMap也用到了红黑树。

<https://my.oschina.net/u/3272058/blog/1914452>

左旋：对节点x进行左旋，意味着将“x的右孩子变成x的父亲”，而将“x原先的右孩子的左孩子变成x的右孩子”。即左旋中的“左”是指将别旋转的节点变成一个左节点

右旋：对节点x进行右旋，意味着将“x的左孩子变成x的父亲，而将”x原先的左孩子的右孩子变成x的右孩子“。即右旋中的”右“是指将被旋转的节点变成一个右节点

**插入规则**

新插入的节点都为红色

**红黑树插入的4种情形**

（1）新节点位于根节点，其没有父节点时，处理思路：将该节点直接设为黑色即可

（2）新节点的父节点已然是黑色时，处理思路：不用动，这已然是一颗红黑树

（3）父节点和叔节点都是红色时，处理思路：a.将父节点和叔节点设为黑色;b.将祖父节点设为红色;c.将祖父节点设为当前节点，并继续对新当前节点进行操作

（4）父节点是红色，叔节点是黑色时，又分如下四种情况：

* 当前节点是父亲的左孩子，父亲是祖父的左孩子（Left-Left），处理思路：a.将祖父节点右旋;b.交换父节点和祖父节点的颜色
* 当前节点是父亲的右孩子，父亲是祖父的左孩子（Right-Left），处理思路：a.将父节点左旋，并将父节点作为当前节点; b.然后再使用Left Left情形
* 当前节点是父亲的右孩子，父亲是祖父的右孩子（Right-Right），处理思路：a.将祖父节点左旋;b.交换父节点和祖父节点的颜色
* 当前节点是父亲的左孩子，父亲是祖父的右孩子（Left-Right），处理思路：a.将父节点右旋，并将父节点作为当前节点; b.然后再使用Right Right情形

**红黑树删除的情形**

一、从树中删除节点X（**以寻找后继节点的方式进行删除**）

情况①：如果X没有孩子，且如果X是红色，直接删除X；如果X是黑色，则**以X为当前节点**进行旋转调色，最后删掉X

情况②：如果X只有一个孩子C，交换X和C的数值，再对新X进行删除。根据红黑树特性，此时X不可能为红色，因为红色节点要么没有孩子，要么有两个黑孩子。此时**以新X为当前节点**进行情况①的判断

情况③：如果X有两个孩子，则从后继中找到最小节点D，交换X和D的数值，再对新X进行删除。此时**以新X为当前节点**进行情况①或②的判断

二、旋转调色（**N=旋转调色的当前节点[等于情况①中的X]，P=N的父亲，W=N的兄弟，Nf=N的远侄子，Nn=N的近侄子**）

情况1：N是根或者N是红色，则：直接将N设为黑色

情况2：N不是根且N是黑色，且W为红色，则：将W设为黑色，P设为红色，对P进行旋转(N为P的左子时进行左旋，N为P的右子时进行右旋)，将情况转化为情况1、2、3、4、5

情况3：N不是根且N是黑色，且W为黑色，且W的左右子均为黑色，则：将W设为红色，将P设为当前节点进行旋转调色，将情况转化为情况1、2、3、4、5

情况4：N不是根且N是黑色，且W为黑色，且Nf为黑色，Nn为红色，则：交换W与Nn的颜色，并对W进行旋转(N为P的左子进行右旋，N为P的右子进行左旋)，旋转后N的新兄弟W有一个红色WR，则转换为情况5

情况5：N不是根且N是黑色，且W为黑色，且Nf为红色，Nn为黑色，则：将W设为P的颜色，P和Nf设为黑色，并对P进行旋转(N为P的左子进行左旋，N为P的右子进行右旋)，N设为根

二叉查找树（BST）具备什么特性呢？

1.左子树上所有结点的值均小于或等于它的根结点的值。

2.右子树上所有结点的值均大于或等于它的根结点的值。

3.左、右子树也分别为二叉排序树。

查询时，查找所需的而最大次数等于二叉查找树的高度。