http://blog.sina.com.cn/s/blog\_64d0b03c0101alsq.html

BTree,B-Tree,B+Tree,B\*Tree

【摘要】

      最近在看Mysql的存储引擎中索引的优化,神马是索引,支持啥索引.全是浮云，目前Mysql的MyISAM和InnoDB都支持B-Tree索引,InnoDB还支持B+Tree索引，Memory还支持Hash.今天从最基础的学起,学习了解BTree,B-Tree和B+Tree。

【主题】

* B-Tree 介绍
* B-Tree 特性搜索插入等
* B+Tree 介绍
* B\*Tree 介绍

【内容】

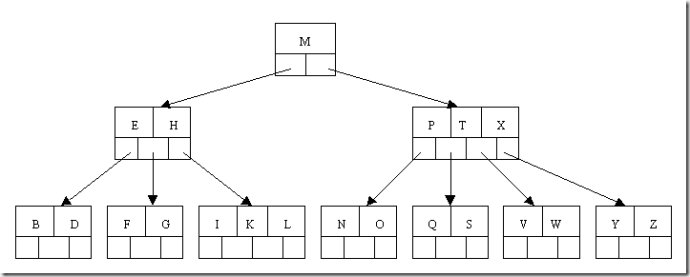
1. B-Tree 介绍

     1970年，R.Bayer和E.mccreight提出了一种适用于外查找的树，它是一种平衡的多叉树，称为B树，其定义如下：

*一棵m阶的B树满足下列条件：*

* 树中每个结点至多有m个孩子；
* 除根结点和叶子结点外，其它每个结点至少有m/2个孩子；
* 若根结点不是叶子结点，则至少有2个孩子；
* 所有叶子结点(失败节点)都出现在同一层，叶子结点不包含任何关键字信息；
* 所有非终端结点中包含下列信息数据 ( n, A0 , K1 , A1 , K2 , A2 , … , Kn , An )，其中： Ki (i=1,…,n)为关键字，且Ki < Ki+1 , Ai (i=0,…,n)为指向子树根结点的指针, n为关键字的个数
* 非叶子结点的指针：P[1], P[2], …, P[M]；其中P[1]指向关键字小于K[1]的子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1], K[i])的子树；

     在B树中，每个结点中关键字从小到大排列，并且当该结点的孩子是非叶子结点时，该k-1个关键字正好是k个孩子包含的关键字的值域的分划。因为叶子结点不包含关键字，所以可以把叶子结点看成在树里实际上并不存在外部结点，指向这些外部结点的指针为空，叶子结点的数目正好等于树中所包含的关键字总个数加1。B树中的一个包含n个关键字，n+1个指针的结点的一般形式为：   （n,P0,K1,P1,K2,P2,…,Kn,Pn）其中，Ki为关键字，K1 <…   Pi   是指向包括Ki到Ki+1之间的关键字的子树的指针。

[](http://blog.sina.com.cn/===lw_pic_up_sina===1294198616.1494)

*-- 图1 B-tree示意图*

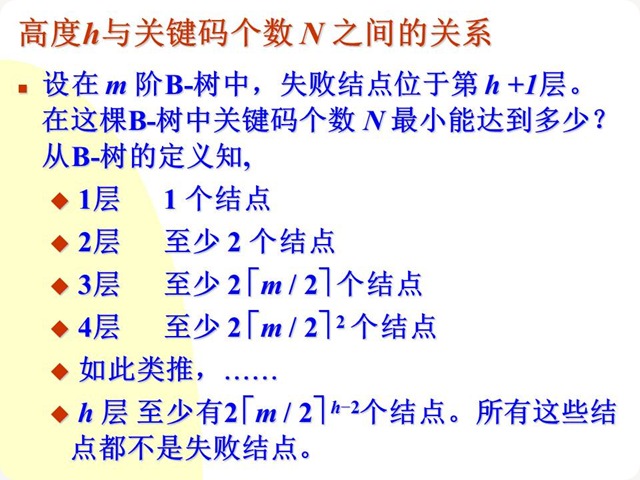
2. B-Tree特性

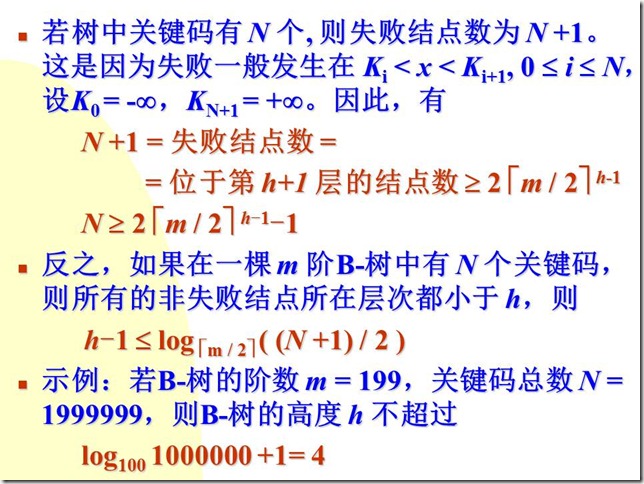
2.1 B-Tree 特性

* 关键字集合分布在整颗树中；
* 任何一个关键字出现且只出现在一个结点中；
* 搜索有可能在非叶子结点结束；
* 其搜索性能等价于在关键字全集内做一次二分查找；
* 自动层次控制；

2.2 B-Tree搜索原理

B-树的搜索，从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为空，或已经是叶子结点；因此，B-Tree的查找过程是一个顺指针查找结点和在结点的关键字中进行查找的交叉进行的过程。

[](http://blog.sina.com.cn/===lw_pic_up_sina===1294198617.9441)

[](http://blog.sina.com.cn/===lw_pic_up_sina===1294198619.8066)

*-- 图2 高度与关键码的计算过程*

2.3 B-Tree 插入

     B-树是从空树起，逐个插入关键码而生成的。

     在B-树，每个非失败结点的关键码个数都在[ m/2 -1, m-1]之间。插入在某个叶结点开始。如果在关键码插入后结点中的关键码个数超出了上界 m-1，则结点需要“分裂”，否则可以直接插入。

     实现结点“分裂”的原则是：

     设结点 A 中已经有 m-1 个关键码，当再插入一个关键码后结点中的状态为( m, A0, K1, A1, K2, A2, ……, Km, Am)其中 Ki < Ki+1, 1 =< m

这时必须把结点 p 分裂成两个结点 p 和 q，它们包含的信息分别为：

     结点 p：( m/2 -1, A0, K1, A1, ……, Km/2 -1, Am/2 -1)

     结点 q：(m - m/2, Am/2, Km/2+1, Am/2+1, ……, Km, Am)

     位于中间的关键码 Km/2 与指向新结点 q 的指针形成一个二元组 ( Km/2, q )，插入到这两个结点的双亲结点中去。

3. B+Tree

3.1 B+Tree定义

     B+树可以看作是B树的一种变形，在实现文件索引结构方面比B树使用得更普遍。

*一棵 m 阶B+树可以定义如下：*

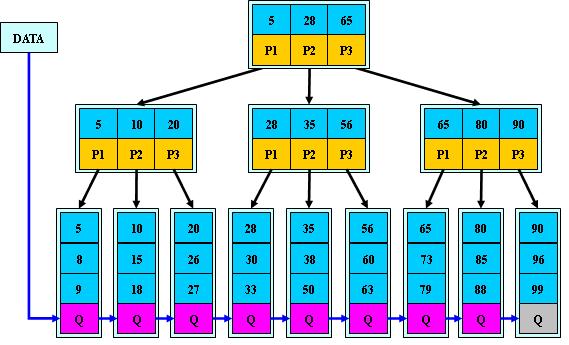
* 树中每个非叶结点最多有 m 棵子树；
* 根结点 (非叶结点) 至少有 2 棵子树。除根结点外, 其它的非叶结点至少有 ém/2ù 棵子树；有 n 棵子树的非叶结点有 n-1 个关键码。
* 所有叶结点都处于同一层次上，包含了全部关键码及指向相应数据对象存放地址的指针，且叶结点本身按关键码从小到大顺序链接；
* 每个叶结点中的子树棵数 n 可以多于 m，可以少于 m，视关键码字节数及对象地址指针字节数而定。
* 若设结点可容纳最大关键码数为 m1，则指向对象的地址指针也有 m1 个。
* 结点中的子树棵数 n 应满足 n 属于[m1/2, m1]
* 若根结点同时又是叶结点，则结点格式同叶结点。
* 所有的非叶结点可以看成是索引部分，结点中关键码 Ki 与指向子树的指针 Pi 构成对子树 (即下一层索引块) 的索引项 ( Ki, Pi )，Ki 是子树中最小的关键码。
* 特别地，子树指针 P0 所指子树上所有关键码均小于 K1。结点格式同B树。
* 叶结点中存放的是对实际数据对象的索引。
* 在B+树中有两个头指针：一个指向B+树的根结点，一个指向关键码最小的叶结点。

*B+Tree与B-Tree区别*

1. 非叶子结点的子树指针与关键字个数相同；
2. 非叶子结点的子树指针P[i]，指向关键字值属于[K[i], K[i+1])的子树（B-树是开区间）；
3. 为所有叶子结点增加一个链指针；
4. 所有关键字都在叶子结点出现；

*对B+树进行两种搜索运算*

* 循叶结点链顺序搜索
* 另一种是从根结点开始，进行自顶向下，直至叶结点的随机搜索。



*-- 图3 B+Tree示意图*

3.2 B+Tree特性

     B+Tree的搜索与B-Tree也基本相同，区别是B+Tree只有达到叶子结点才命中（B-Tree可以在非叶子结点命中），其性能也等价于在关键字全集做一次二分查找；

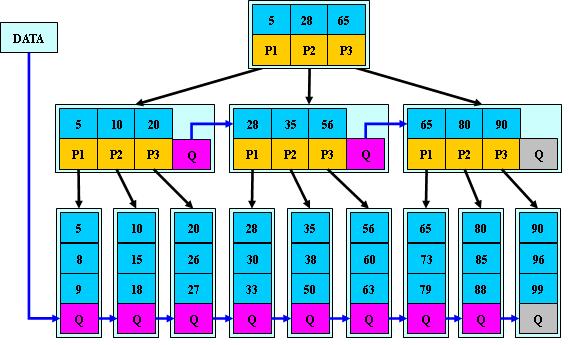
*B+Tree的特性*

* 所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好是有序的；
* 不可能在非叶子结点命中；
* 非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储（关键字）数据的数据层；
* 更适合文件索引系统

4. B\*Tree

4.1 B\*Tree

     B\*Tree是B+树的变体，在B+Tree的非根和非叶子结点再增加指向兄弟的指针；



*-- 图4 B\*Tree示意图*

     B\*树定义了非叶子结点关键字个数至少为(2/3)\*M，即块的最低使用率为2/3（代替B+树的1/2）；

     B+树的分裂：当一个结点满时，分配一个新的结点，并将原结点中1/2的数据复制到新结点，最后在父结点中增加新结点的指针；B+树的分裂只影响原结点和父结点，而不会影响兄弟结点，所以它不需要指向兄弟的指针；

     B\*树的分裂：当一个结点满时，如果它的下一个兄弟结点未满，那么将一部分数据移到兄弟结点中，再在原结点插入关键字，最后修改父结点中兄弟结点的关键字（因为兄弟结点的关键字范围改变了）；如果兄弟也满了，则在原结点与兄弟结点之间增加新结点，并各复制1/3的数据到新结点，最后在父结点增加新结点的指针；

     所以，B\*树分配新结点的概率比B+树要低，空间使用率更高；

【结束】

***自我总结：***

* B-树：多路搜索树，每个结点存储M/2到M个关键字，非叶子结点存储指向关键字范围的子结点；所有关键字在整颗树中出现，且只出现一次，非叶子结点可以命中；
* B+树：在B-树基础上，为叶子结点增加链表指针，所有关键字都在叶子结点中出现，非叶子结点作为叶子结点的索引；B+树总是到叶子结点才命中；
* B\*树：在B+树基础上，为非叶子结点也增加链表指针，将结点的最低利用率从1/2提高到2/3；

***参考资料：***

* ***CSDN sx.manesking.bread·***[***BTree,B-Tree,B+Tree,B\*Tree都是什么***](http://blog.csdn.net/manesking/archive/2007/02/09/1505979.aspx)
* ***Stanford.Nick Parlante·***[***Binary Tree***](http://cslibrary.stanford.edu/110/BinaryTrees.html)
* http://hi.baidu.com/sobuhu/item/ae568f0cbbb86dcf75cd3c56