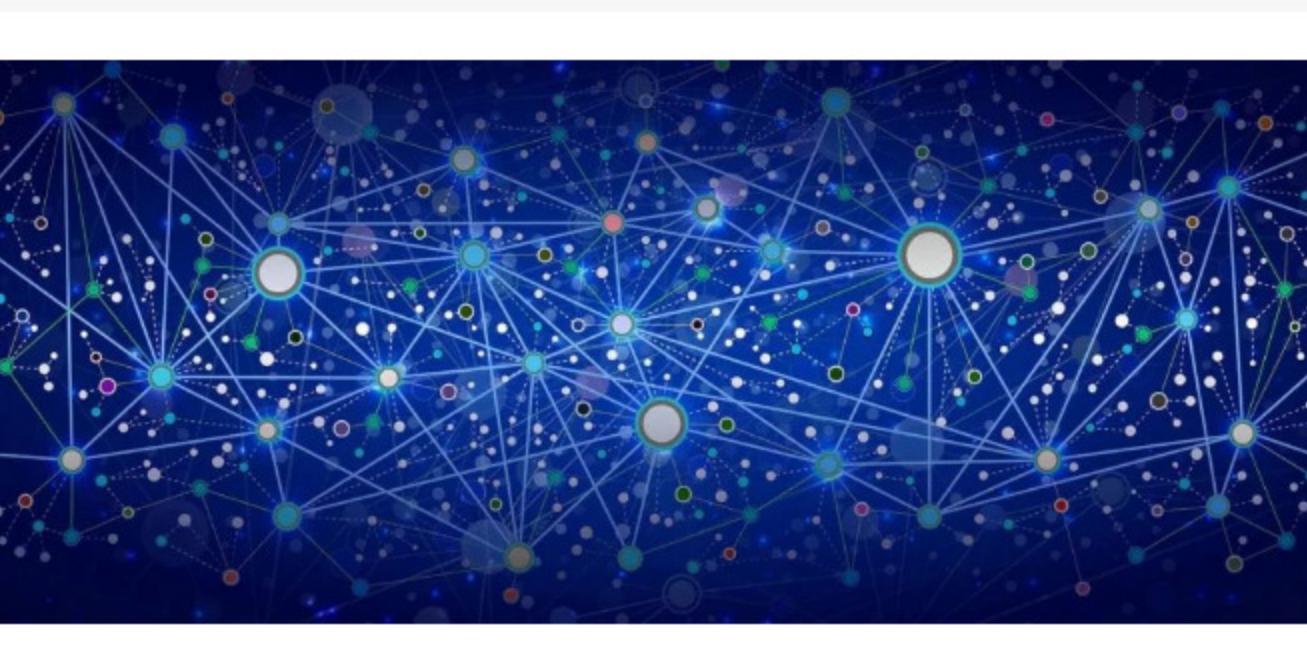
面试时你心里有没有点B+树和B树?

程序员面试 6月27日



MySql中，主要有四种类型的索引，分别是B-Tree索引，Hash索引，FullText索引和R-Tree索引。一般来说，MySQL中的B-Tree索引的物理文件大多都是以Balance Tree的结构来存储的也就是所有实际需要的数据都存放于Tree的Leaf Node(叶子节点)，而且到任何一个LeafNode的最短路径的长度都是完全相同的。

B+Tree在B-Tree数据结构的基础上做了很小改动，在每个LeafNode上面存放索引键的相关信息之外，还存储了指向与该Leaf Node相邻的后一个LeafNode的指针信息（增加了顺序访问指针），这主要是为了加快检索多个相邻LeafNode的效率考虑

InnoDB是MySQL的默认存储引擎（MySQL5.5.5之前是MyISAM）

接下来我们就讲讲B+树的由来，他的爷爷平衡二叉树以及爸爸B树。

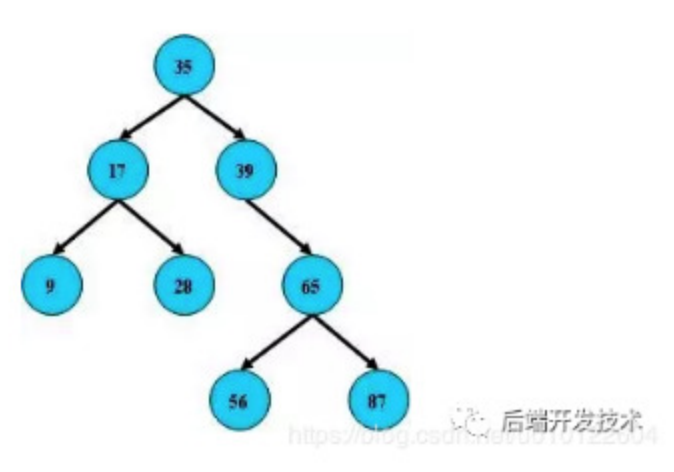
平衡二叉树

平衡二叉树是基于二分法的策略提高数据的查找速度的二叉树的数据结构。平衡二叉树是采用二分法思维把数据按规则组装成一个树形结构的数据，用这个树形结构的数据减少无关数据的检索，大大的提升了数据检索的速度；平衡二叉树的数据结构组装过程有以下规则：

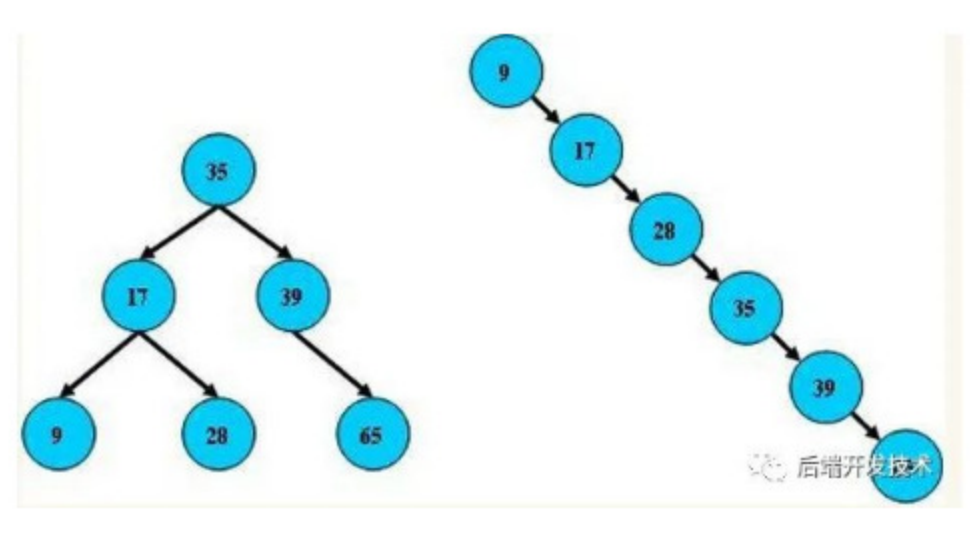
1. 所有非叶子节点至多拥有两个儿子（Leaf和Right）

2. 左右结点存储一个关键字

3. 非叶子节点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树



平衡树的层级结构：因为平衡二叉树查询性能和树的层级（h高度）成反比，h值越小查询越快、为了保证树的结构左右两端数据大致平衡降低二叉树的查询难度一般会采用一种算法机制实现节点数据结构的平衡，实现了这种算法的有比如Treap、红黑树，使用平衡二叉树能保证数据的左右两边的节点层级相差不会大于1.，通过这样避免树形结构由于删除增加变成线性链表影响查询效率，保证数据平衡的情况下查找数据的速度近于二分法查找。



总结平衡二叉树特点：

（1）非叶子节点最多拥有两个子节点

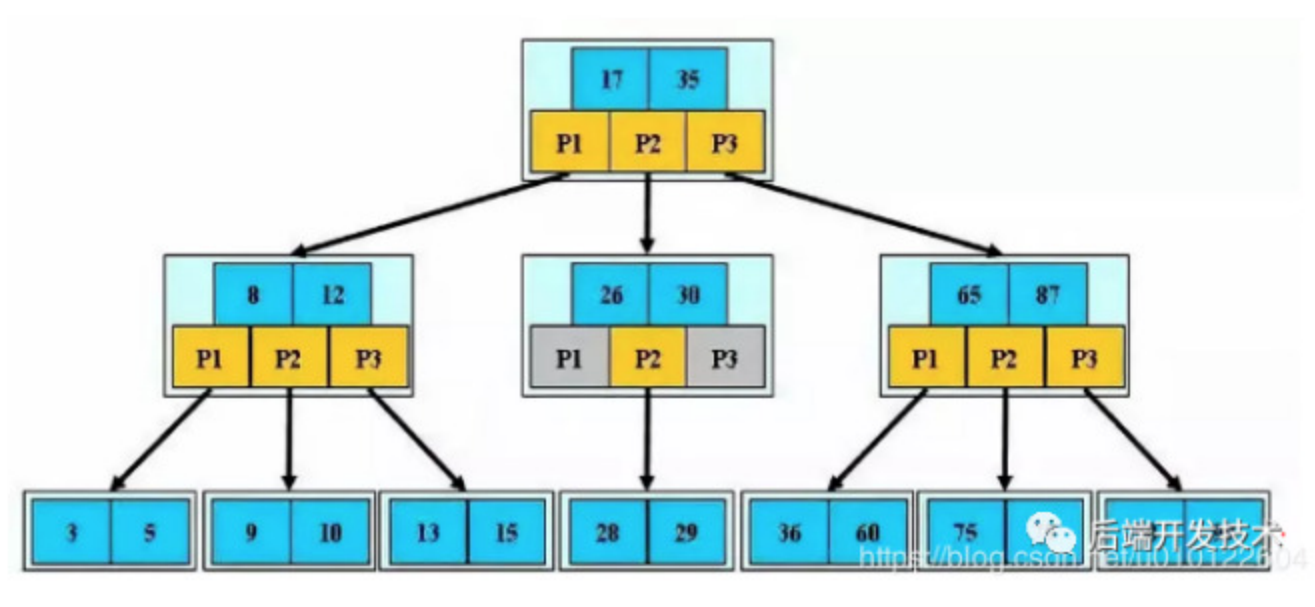
（2）非叶子节值大于左边子节点、小于右边子节点

（3）树的左右两边的层级数相差不会大于1

（4）没有值相等重复的节点

B-树

是一种多路搜索树（并不是二叉的），B树和平衡二叉树稍有不同的是B树属于多叉树又名平衡多路查找树（查找路径不只两个），数据库索引技术里大量使用者B树和B+树的数据结构，让我们来看看他有什么特点。



1、排序方式：所有节点关键字是按递增次序排列，并遵循左小右大原则

2、子节点数：非叶节点的子节点数>1，且<=M ，且M>=2，空树除外（注：M阶代表一个树节点最多有多少个查找路径，M=M路,当M=2则是2叉树,M=3则是3叉）

3、关键字数：枝节点的关键字数量大于等于ceil(m/2)-1个且小于等于M-1个（注：ceil()是个朝正无穷方向取整的函数 如ceil(1.1)结果为2)

4、所有叶子节点均在同一层、叶子节点除了包含了关键字和关键字记录的指针外也有指向其子节点的指针只不过其指针地址都为null对应下图最后一层节点的空格子

B-树的搜索

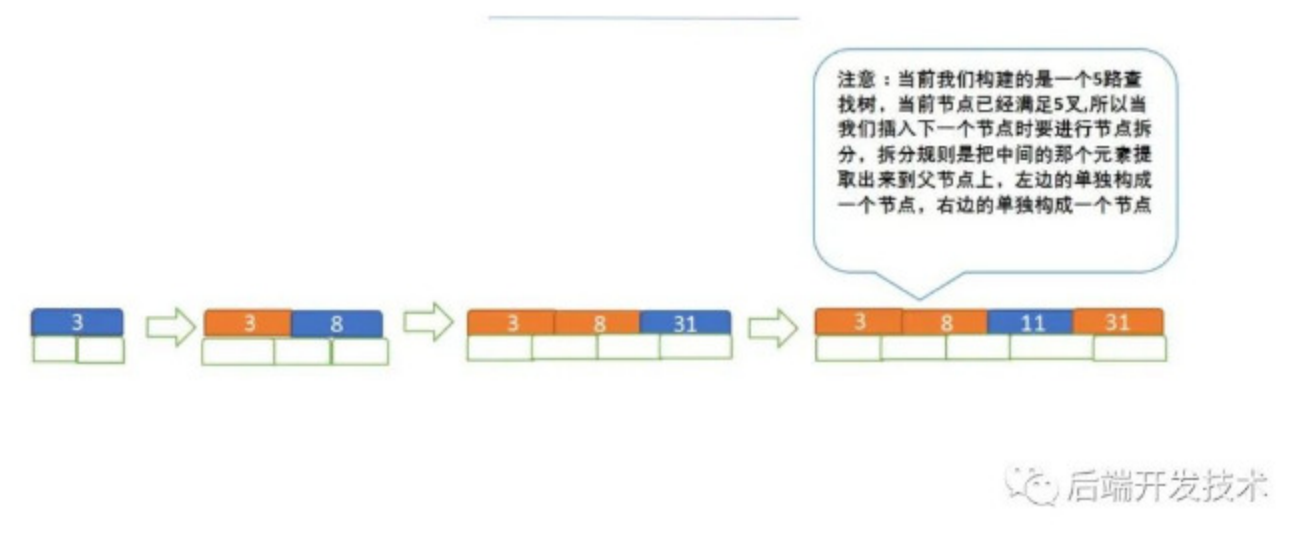
从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为空，或已经是叶子结点；

由于限制了除根结点以外的非叶子结点，至少含有M/2个儿子，确保了结点的至少利用率。所以B-树的性能总是等价于二分查找（与M值无关），也就没有B树平衡的问题，由于M/2的限制，在插入结点时，如果结点已满，需要将结点分裂为两个各占M/2的结点，删除结点时，需将两个不足M/2的兄弟节点合并。

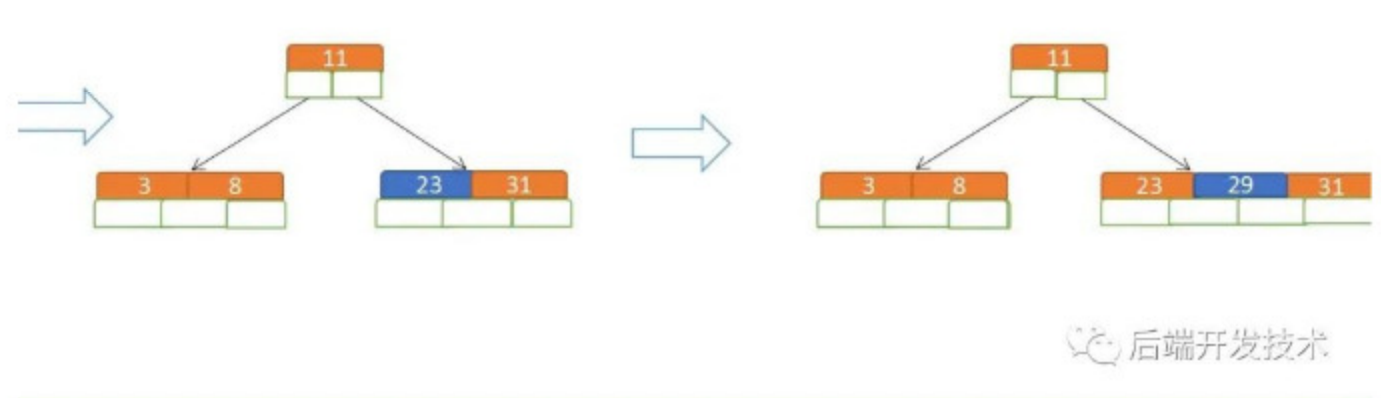
插入：

定义一个5阶树（平衡5路查找树;），现在我们要把3、8、31、11、23、29、50、28 这些数字构建出一个5阶树出来;

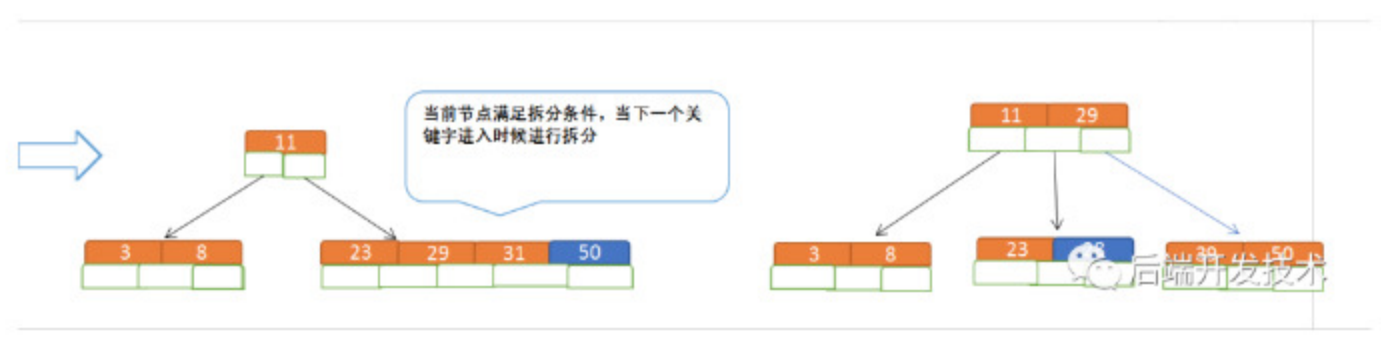
1.先插入 3、8、31、11



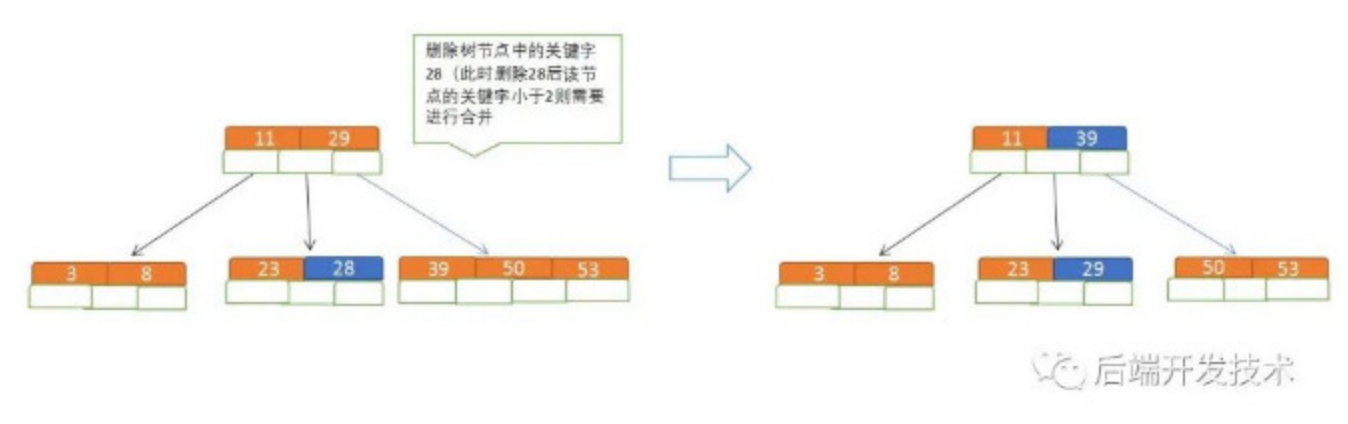
2.再插入23、29



3.再插入50、28



删除：



B+树

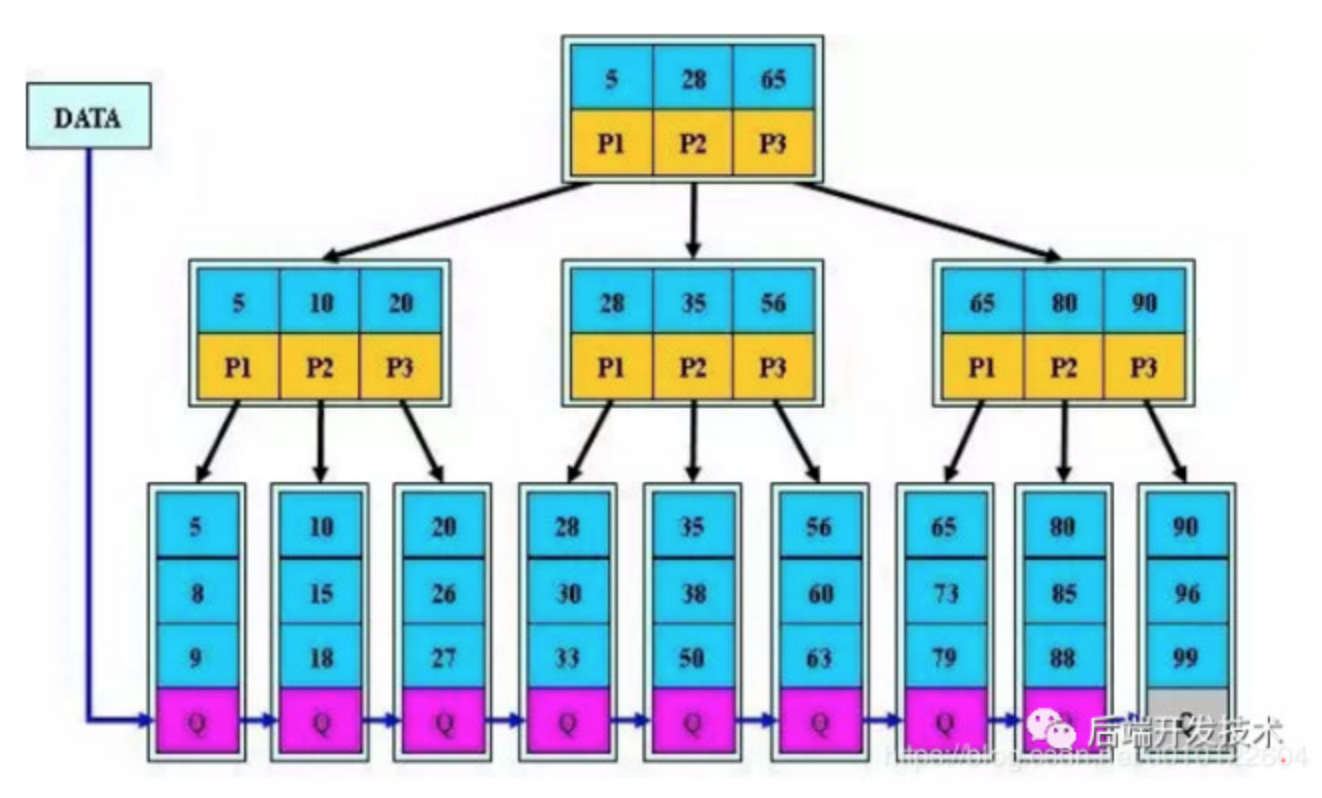
B+树是B-树的变体，也是一种多路搜索树。相对于B树来说B+树更充分的利用了节点的空间，让查询速度更加稳定，其速度完全接近于二分法查找。规则：

1、B+跟B树不同B+树的非叶子节点不保存关键字记录的指针，只进行数据索引，这样使得B+树每个非叶子节点所能保存的关键字大大增加

2、B+树叶子节点保存了父节点的所有关键字记录的指针，所有数据地址必须要到叶子节点才能获取到。所以每次数据查询的次数都一样

3、B+树叶子节点的关键字从小到大有序排列，左边结尾数据都会保存右边节点开始数据的指针。

4、非叶子节点的子节点数=关键字数



1、其定义基本与B-树同，除了：

2、非叶子结点的子树指针与关键字个数相同；

3、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）；

5、为所有叶子结点增加一个链指针；

6、所有关键字都在叶子结点出现

B+树的特性

1、B+树的层级更少：相较于B树B+每个非叶子节点存储的关键字数更多，树的层级更少所以查询数据更快

2、B+树查询速度更稳定：B+所有关键字数据地址都存在叶子节点上，所以每次查找的次数都相同所以查询速度要比B树更稳定

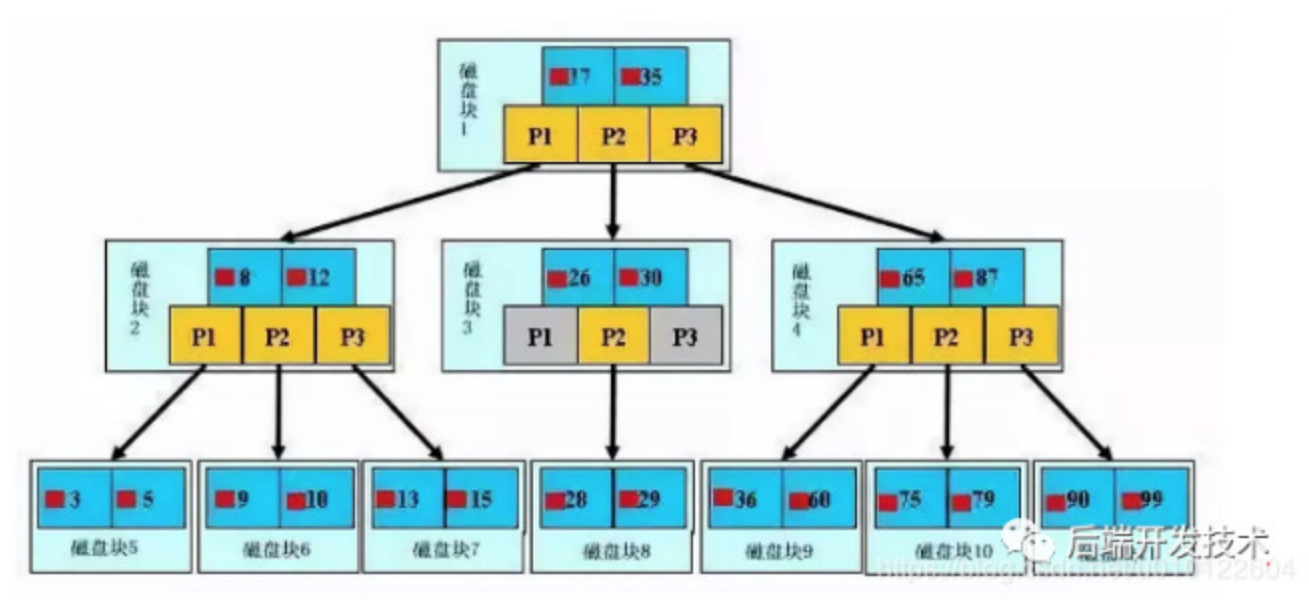
3、B+树天然具备排序功能：所有关键字都出现在叶子结点的链表中（稠密索引），B+树所有的叶子节点数据构成了一个有序链表，在查询大小区间的数据时候更方便，数据紧密性很高，缓存的命中率也会比B树高。

4、B+树全节点遍历更快：B+树遍历整棵树只需要遍历所有的叶子节点即可，，而不需要像B树一样需要对每一层进行遍历，这有利于数据库做全表扫描。

B树相对于B+树的优点是，如果经常访问的数据离根节点很近，而B树的非叶子节点本身存有关键字其数据的地址，所以这种数据检索的时候会要比B+树快。

5、更适合文件索引系统

B+树索引原理



上图，是一颗b+树，关于b+树的定义可以参见B+树，这里只说一些重点，浅蓝色的块我们称之为一个磁盘块，可以看到每个磁盘块包含几个数据项（深蓝色所示）和指针（黄色所示），如磁盘块1包含数据项17和35，包含指针P1、P2、P3，P1表示小于17的磁盘块，P2表示在17和35之间的磁盘块，P3表示大于35的磁盘块。真实的数据存在于叶子节点即3、5、9、10、13、15、28、29、36、60、75、79、90、99。非叶子节点只不存储真实的数据，只存储指引搜索方向的数据项，如17、35并不真实存在于数据表中。

B+树查找过程

如果要查找数据项29，那么首先会把磁盘块1由磁盘加载到内存，此时发生一次IO，在内存中用二分查找确定29在17和35之间，锁定磁盘块1的P2指针，内存时间因为非常短（相比磁盘的IO）可以忽略不计，通过磁盘块1的P2指针的磁盘地址把磁盘块3由磁盘加载到内存，发生第二次IO，29在26和30之间，锁定磁盘块3的P2指针，通过指针加载磁盘块8到内存，发生第三次IO，同时内存中做二分查找找到29，结束查询，总计三次IO。真实的情况是，3层的b+树可以表示上百万的数据，如果上百万的数据查找只需要三次IO，性能提高将是巨大的，如果没有索引，每个数据项都要发生一次IO，那么总共需要百万次的IO，显然成本非常非常高

B+树性质

1、通过上面的分析，我们知道IO次数取决于b+数的高度h，假设当前数据表的数据为N，每个磁盘块的数据项的数量是m，则有h=log(m+1)N，当数据量N一定的情况下，m越大，h越小；而m = 磁盘块的大小 / 数据项的大小，磁盘块的大小也就是一个数据页的大小，是固定的，如果数据项占的空间越小，数据项的数量越多，树的高度越低。这就是为什么每个数据项，即索引字段要尽量的小，比如int占4字节，要比bigint8字节少一半。这也是为什么b+树要求把真实的数据放到叶子节点而不是内层节点，一旦放到内层节点，磁盘块的数据项会大幅度下降，导致树增高。当数据项等于1时将会退化成线性表。

2、当b+树的数据项是复合的数据结构，比如(name,age,sex)的时候，b+数是按照从左到右的顺序来建立搜索树的，比如当(张三,20,F)这样的数据来检索的时候，b+树会优先比较name来确定下一步的所搜方向，如果name相同再依次比较age和sex，最后得到检索的数据；但当(20,F)这样的没有name的数据来的时候，b+树就不知道下一步该查哪个节点，因为建立搜索树的时候name就是第一个比较因子，必须要先根据name来搜索才能知道下一步去哪里查询。比如当(张三,F)这样的数据来检索时，b+树可以用name来指定搜索方向，但下一个字段age的缺失，所以只能把名字等于张三的数据都找到，然后再匹配性别是F的数据了， 这个是非常重要的性质，即索引的最左匹配特性。