# 高级数据结构篇（Btree&B+Tree）

作为一个IT从业者大家对数据库肯定是都知道的，大家应该知道在数据库中有个索引，在一张表中用了索引与不用索引那查找效率简直就是天壤之别，但是大家有没思考过，你经常用的索引是什么样的数据结构呢？它为什么又能这么高效的查找呢？究竟使用了什么样的算法呢？

select \* from user where user\_id = 100

select \* from user where user\_id > 10

select \* from user where user\_id = 100 and name = “zzzz”

如果是你你该如何利用我们已经学过的知识来设计上面这个sql的索引结构呢？

1.链表:O(n) 千万级

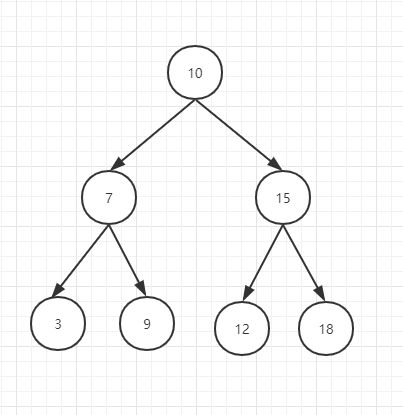
2.数组

3.二叉搜索树

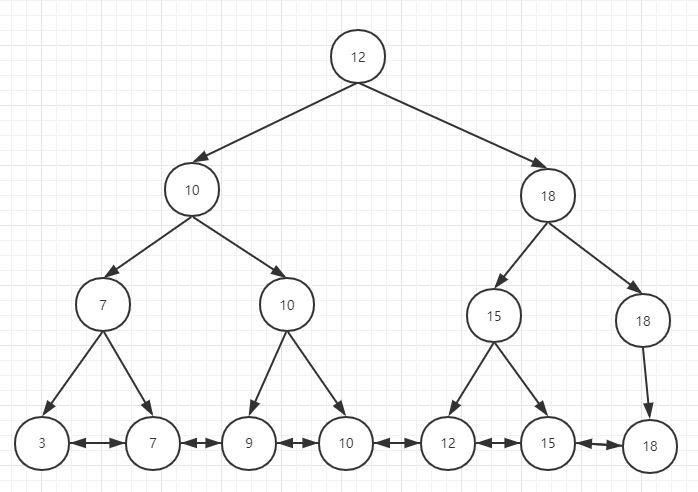
4.平衡树

5.红黑树

6.Hash



改造二叉搜索树：



大家看我改造后的二叉搜索树，

执行 select \* from table where id = 10

执行 select \* from table where id > 12

select \* from table where id > 12 and id < 20

效果：

能解决我们上面所有的sql语句；

效率 logn

2^32=21亿；

IO：指的是从磁盘读取数据。32层就要读取32次。CPU,内存，IO；

IO从磁盘读一次会读多少数据？计算机组成原理。Page的。页，4KB。

Int占多少空间？4B

改造二叉搜索树：

通过以上可知我们能够把sql语句的功能都能实现，而且效率也很高，那是不是这种结构

就可以作为mysql的索引了呢？

大家想想，右边这个结构我们是存在哪里的？有什么问题？怎么解决？

问题1：搜索效率：32次

问题2：查询次数：

问题3：磁盘IO：解决这个问题；

B+Tree的性质：

M阶的B+Tree:叉树

（1）每个节点最多有m个子节点

（2）除根节点外，每个节点至少有m/2个子节点，注意如果结果除不尽，就向上取整，比如5/2=3。 3/2=ceil(1.5)=2

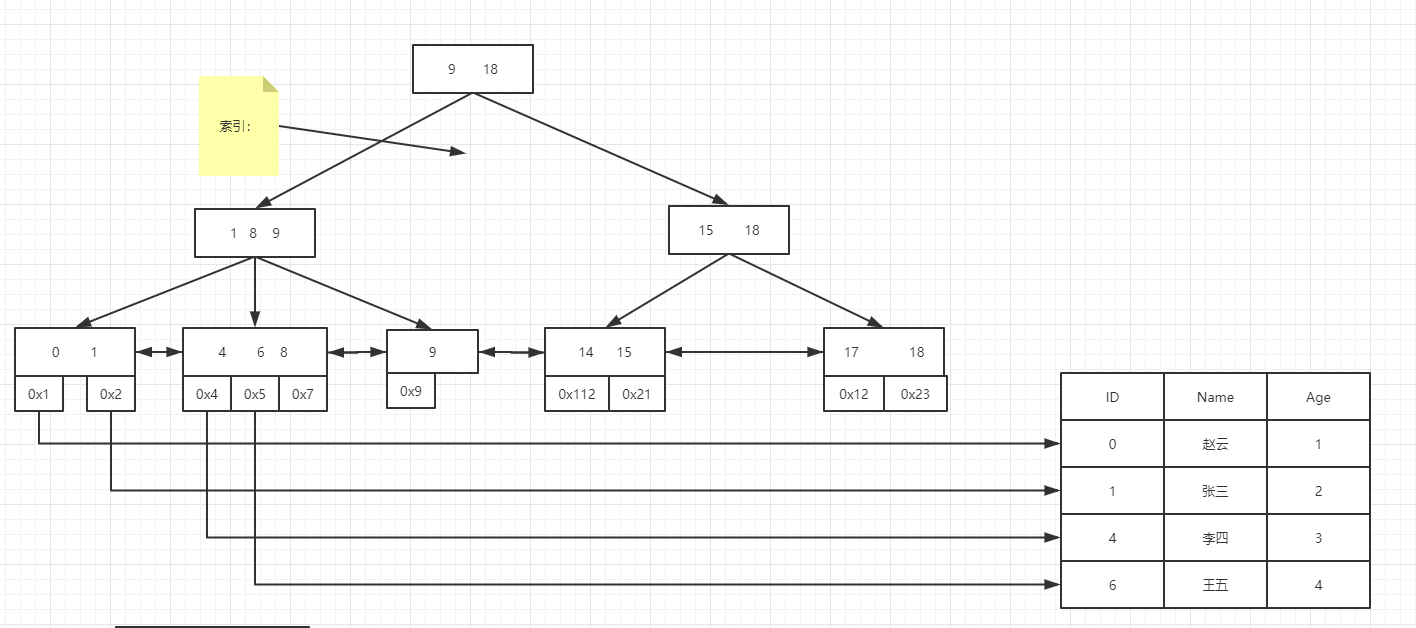
（3）根节点要么是空，要么是独根，否则至少有2个子节点

（4）有k个子节点的节点必有k个关键码:就是 有m个数据就有m个叉叉；

（5）叶节点的高度一致：这个的 好处是什么？

满树？连续?

构建一个B+Tree的过程



B+Tree在Mysql中的分析

1.阶数：通过上面的分析我们知道这个阶数很重要直接决定了B+Tree的查找效率以及性能，那么在Mysql中我们如何设定这阶数呢？

通过mysql的页大小决定，一般是16k。那么一个主键类型为bigint的字段建索引大约的消耗空间是多少呢？

Int：8字节，指针一个也算4字节。

一页的节点：16kb/ (8b+8b)=1k 键值+指针

我们刚举例的3阶B+Tree你们计算下可以存多少索引值？

1024\*1024\*1024=多少？

可想而知如果在理想情况下 我们的mysql查询是不是很高效，一般最多也就4阶左右的B+Tree。

根据我们所学的数据结构知识，我们应该如何正确的建立索引：

1.索引不能太多，因为B+tree的插入和删除是要维护的，太多的索引会导致插入变慢。

2.建了索引的字段不能使用like ‘%%’否则是失效的

3.建索引的字段类型不能太大，字段越小阶数就越大，效率就越高，int 和 bigint，varchar（10），varchar（100），text，lontext；B+Tree。全文索引

4.建索引的字段值不能太多一样的，数学里面有个叫什么散列多一些（离散），比如我们把性别建索引会出现啥情况？左边都是一样的值 过滤不了一半。User sex单独建索引 0 1

5.联合索引的最左匹配原则。Select \* from user where name = ‘zzzz’ and id = 1 我的对（ id,name)建的索引，mysql解析的时候会自动优化。

Select \* from user where name = ‘zzzzz’ and age=10我的对（ id,name,age)建的索引

6.NOT IN 是不会走索引的 not in (1,2,3)

In的值太多 mysql会报错的。上线的

B-Tree和B+Tree的区别

1.B-tree所有的节点都会存数据

2.b-tree叶子节点没有链表

我们认为b-tree是b+tree的过度。

二叉查找树，红黑树，B-Tree，B+Tree 的区别：

二叉查找树：二叉搜索树，优点查找快，但是在某些情况下会退化成链表，它是所有

高效查找树的基础，如果不懂这个 其他的也学不懂。根本性的东西，最好能自己手写出来。

红黑树：内存查找高效树，不适合大数据量 也不适合磁盘存储的。具体的分析就是IO浪费以及读取资源浪费，还有就是树的深度会很大。适合一些底层系统做内存运算

B-Tree:可以认为是B+Tree过度。只需要知道BTree就可以

B+Tree：最适合大数据的磁盘索引，经典的MySql，所有的数据都存在叶子节点。其他都是索引，增加了系统的稳定性以及遍历以及查找效率

不同：关键字和Key值。数据存储的地方，双向链表。

M阶：这个由磁盘的页面大小决定，磁盘快和页内存都是4KB。我们的节点数也就是我们的M值 应该要尽可能的跟他一样。1 0.75的原则HashMap。这样的好处就是为了我们一次刚好能全部拿出一个节点里面存的所有的数据。