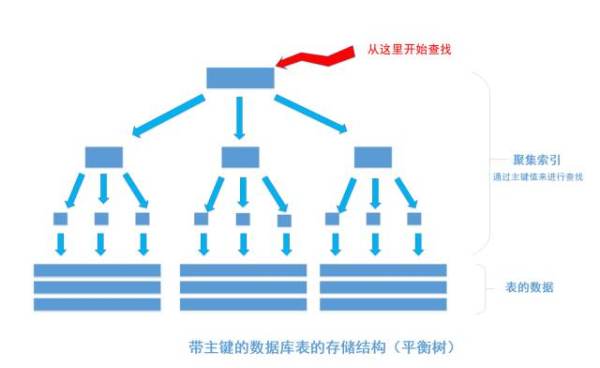
**数据库索引原理**

1描述：

想要理解索引原理必须清楚一种数据结构「平衡树」(非二叉)，也就是b tree或者 b+ tree，重要的事情说三遍：“平衡树，平衡树，平衡树”。当然， 有的数据库也使用哈希桶作用索引的数据结构 ， 然而， 主流的RDBMS都是把平衡树当做数据表默认的索引数据结构的。

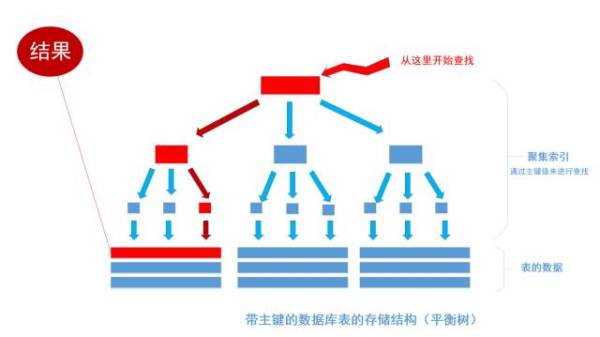
我们平时建表的时候都会为表加上主键， 在某些关系数据库中， 如果建表时不指定主键，数据库会拒绝建表的语句执行。 事实上， 一个加了主键的表，并不能被称之为「表」。一个没加主键的表，它的数据无序的放置在磁盘存储器上，一行一行的排列的很整齐， 跟我认知中的「表」很接近。如果给表上了主键，那么表在磁盘上的存储结构就由整齐排列的结构转变成了树状结构，也就是上面说的「平衡树」结构，换句话说，就是整个表就变成了一个索引。没错， 再说一遍， 整个表变成了一个索引，也就是所谓的**「聚集索引」**。 这就是为什么一个表只能有一个主键， 一个表只能有一个「聚集索引」，因为主键的作用就是把「表」的数据格式转换成「索引（平衡树）」的格式放置。



其中树的所有结点（底部除外）的数据都是由主键字段中的数据构成，也就是通常我们指定主键的id字段。最下面部分是真正表中的数据。 假如我们执行一个SQL语句：

select \* from table where id = 1256;

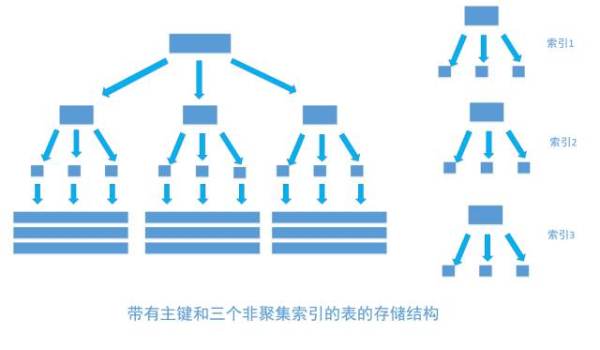
首先根据索引定位到1256这个值所在的叶结点，然后再通过叶结点取到id等于1256的数据行。 这里不讲解平衡树的运行细节， 但是从上图能看出，树一共有三层， 从根节点至叶节点只需要经过三次查找就能得到结果。如下图



假如一张表有一亿条数据 坏的情况下需要匹配一亿次才能得到结果，用大O标记法就是O(n)最坏时间复杂度。如果把这张表转换成平衡树结构（一棵非常茂盛和节点非常多的树），假设这棵树有10层，那么只需要10次IO开销就能查找到所需要的数据， 速度以指数级别提升，用大O标记法就是O(log n)，n是记录总树，底数是树的分叉数，

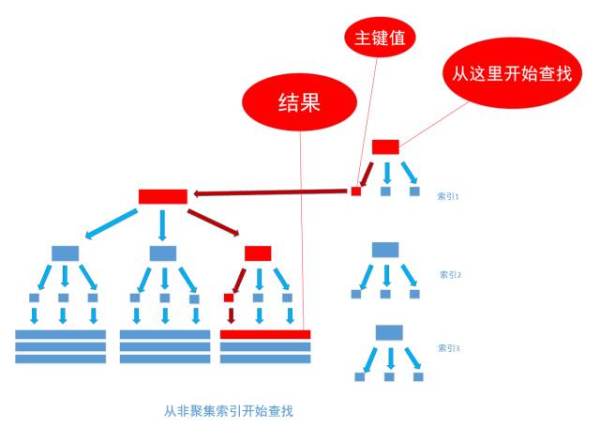
然而， 事物都是有两面的， 索引能让数据库查询数据的速度上升， 而使写入数据的速度下降，原因很简单的， 因为平衡树这个结构必须一直维持在一个正确的状态， 增删改数据都会改变平衡树各节点中的索引数据内容，破坏树结构， 因此，在每次数据改变时， DBMS必须去重新梳理树（索引）的结构以确保它的正确，这会带来不小的性能开销，也就是为什么索引会给查询以外的操作带来副作用的原因。

**非聚集索引**和聚集索引一样， 同样是采用平衡树作为索引的数据结构。索引树结构中各节点的值来自于表中的索引字段， 每个索引（非聚集索引）互相之间不存在关联



每次给字段建一个新索引， 字段中的数据就会被复制一份出来， 用于生成索引。 因此， 给表添加索引，会增加表的体积， 占用磁盘存储空间。

非聚集索引和聚集索引的区别在于， 通过聚集索引可以查到需要查找的数据， 而通过非聚集索引可以查到记录对应的主键值 ， 再使用主键的值通过聚集索引查找到需要的数据，如下图



然而， 有一种例外可以不使用聚集索引就能查询出所需要的数据， 这种非主流的方法 称之为**「覆盖索引」**查询，也就是平时所说的复合索引或者多字段索引查询。

