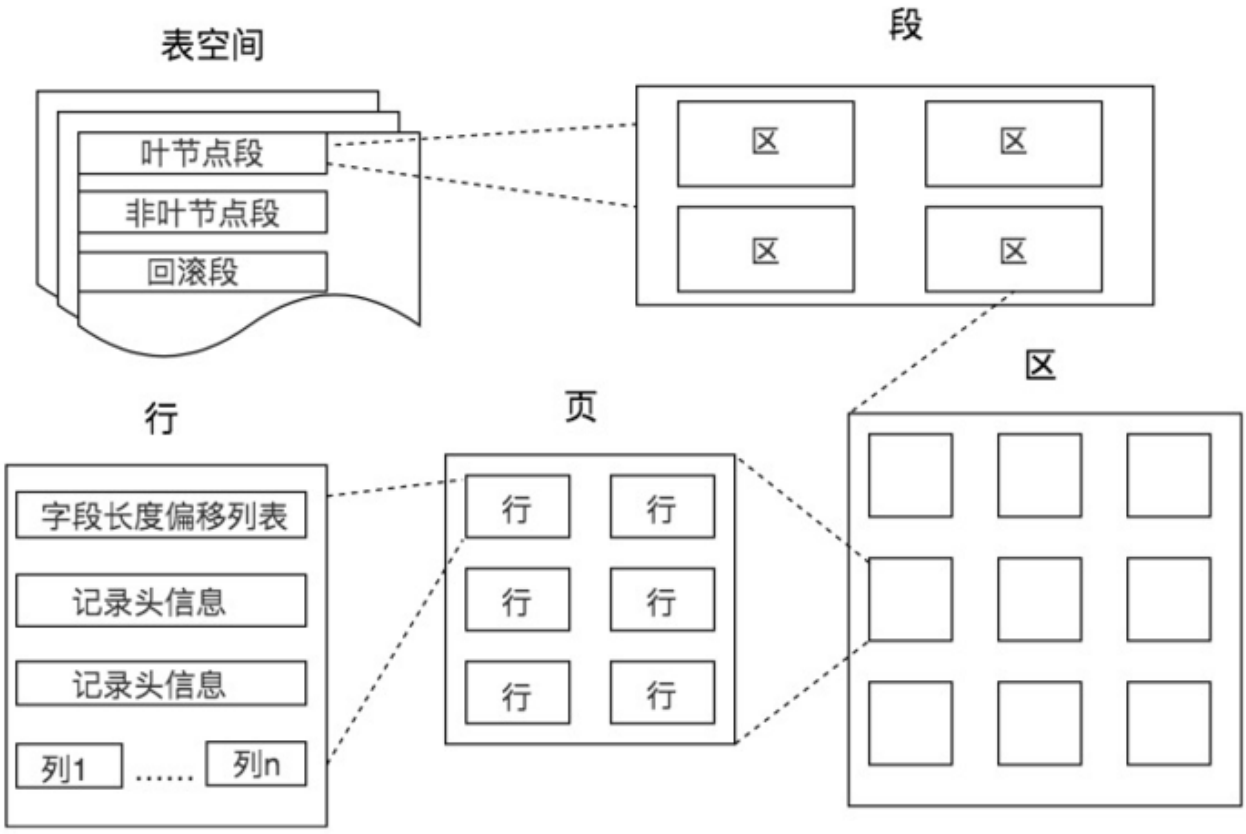
MySQL的存储结构

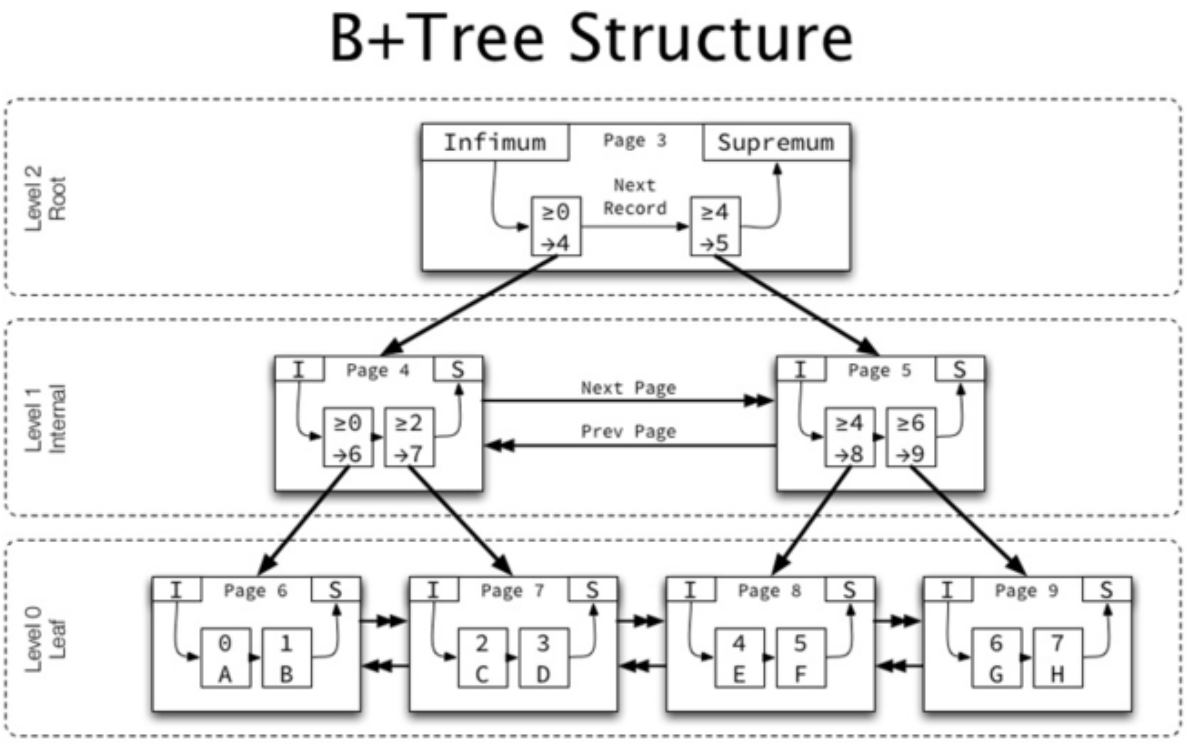
表存储结构



> 单位：表>段>区>页>行

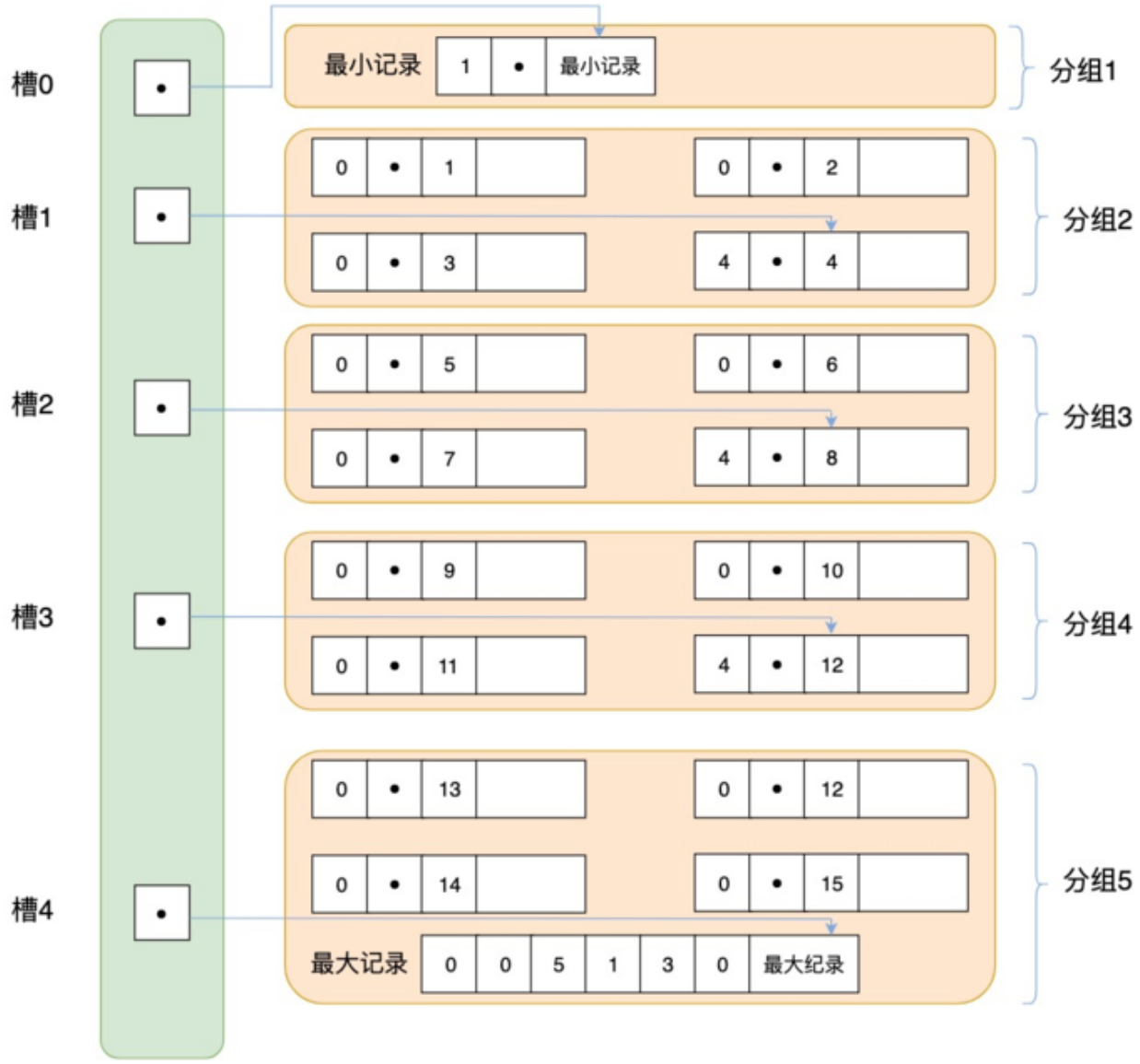
在数据库中， 不论读一行，还是读多行，都是将这些行所在的页进行加载。也就是说存储空间的基本单位是页。
一个页就是一棵树B+树的节点，数据库I/O操作的最小单位是页，与数据库相关的内容都会存储在页的结构里。

B+树索引结构



1. 在一棵B+树中，每个节点为都是一个页，每次新建节点的时候，就会申请一个页空间
2. 同一层的节点为之间，通过页的结构构成了一个双向链表
3. 非叶子节点为，包括了多个索引行，每个索引行里存储索引键和指向下一层页面的指针
4. 叶子节点为，存储了关键字和行记录，在节点内部（也就是页结构的内部）记录之间是一个单向的表

B+树页节点结构



有以下几个特点

1. 将所有的记录分成几个组， 每组会存储多条记录，
2. 页目录存储的是槽（slot），槽相当于分组记录的索引，每个槽指针指向了不同组的最后一个记录
3. 我们通过槽定位到组，再查看组中的记录

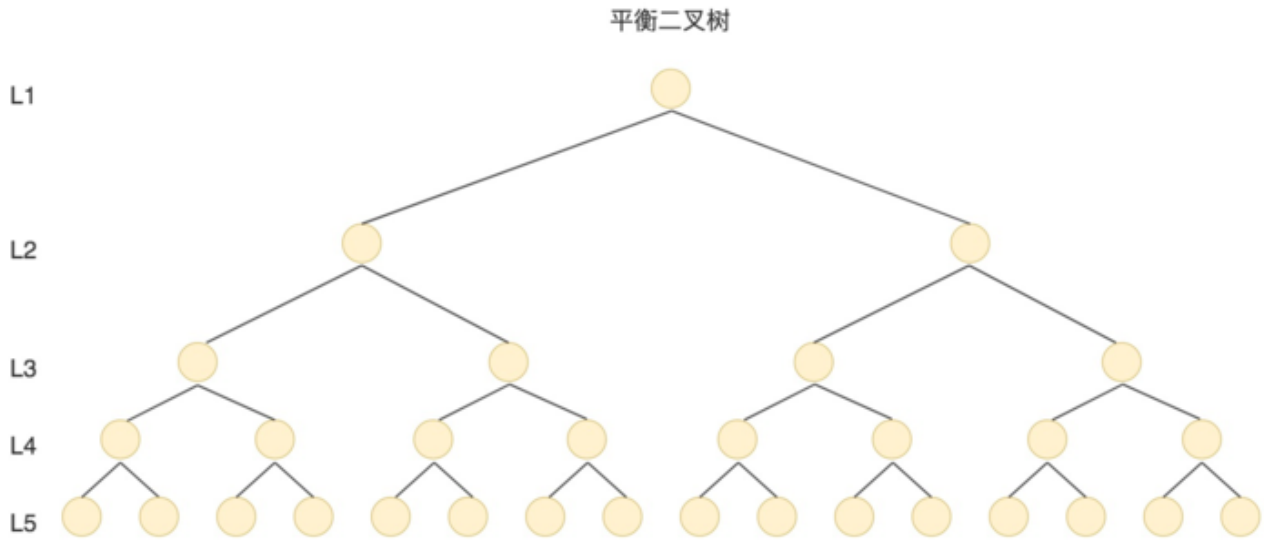
页的主要作用是存储记录，在页中记录以单链表的形式进行存储。
单链表优点是插入、删除方便，缺点是检索效率不高，最坏的情况要遍历链表所有的节点。因此页目录中提供了二分查找的方式，来提高记录的检索效率。

为什么要用B+树索引

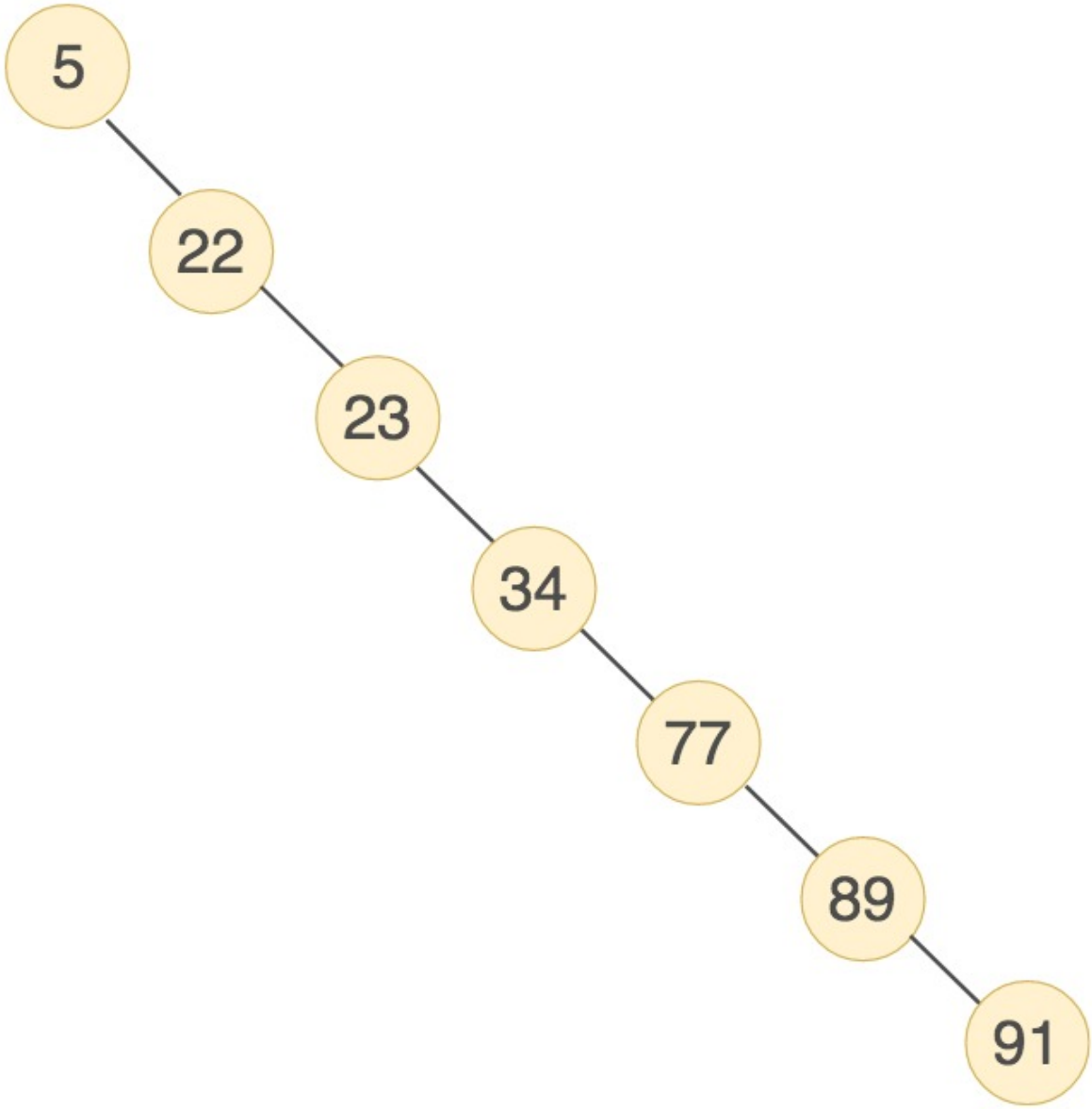
数据库访问数据要通过页，一个页就是一个B+树节点，访问一个节点相当于一次I/O操作，所以越快能找到节点，查找性能越好。
B+树的特点就是够矮够胖，能有效地减少访问节点次数从而提高性能。

下面，我们来对比一个二叉树、多叉树、B树和B+树。

二叉树

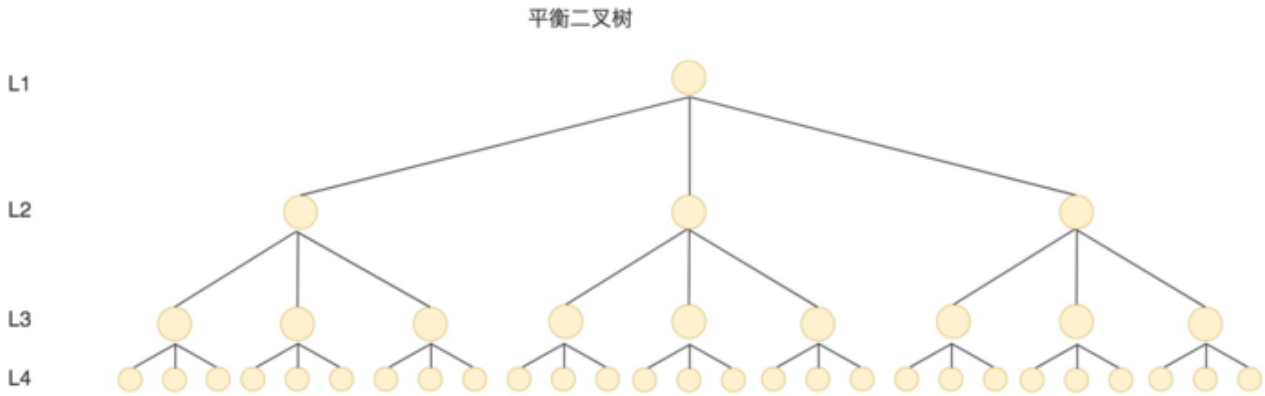


二叉树是一种二分查找树，有很好的查找性能，相当于二分查找。
但是当N比较大的时候，树的深度比较高。数据查询的时间主要依赖于磁盘IO的次数，二叉树深度越大，查找的次数越多，性能越差。
最坏的情况是退化成了链表，如下图



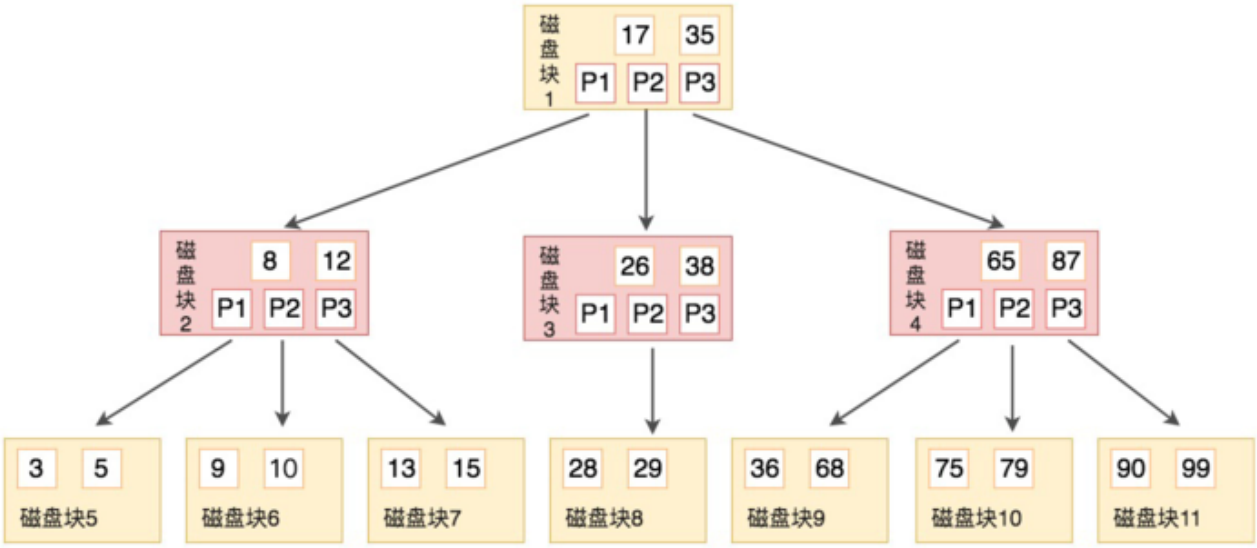
为了让二叉树不至于退化成链表，人们发明了AVL树（平衡二叉搜索树）：任何结点的左子树和右子树高度最多相差1

多叉树



多叉树就是节点可以是M个，能有效地减少高度，高度变小后，节点变少I/O自然少，性能比二叉树好了

B树

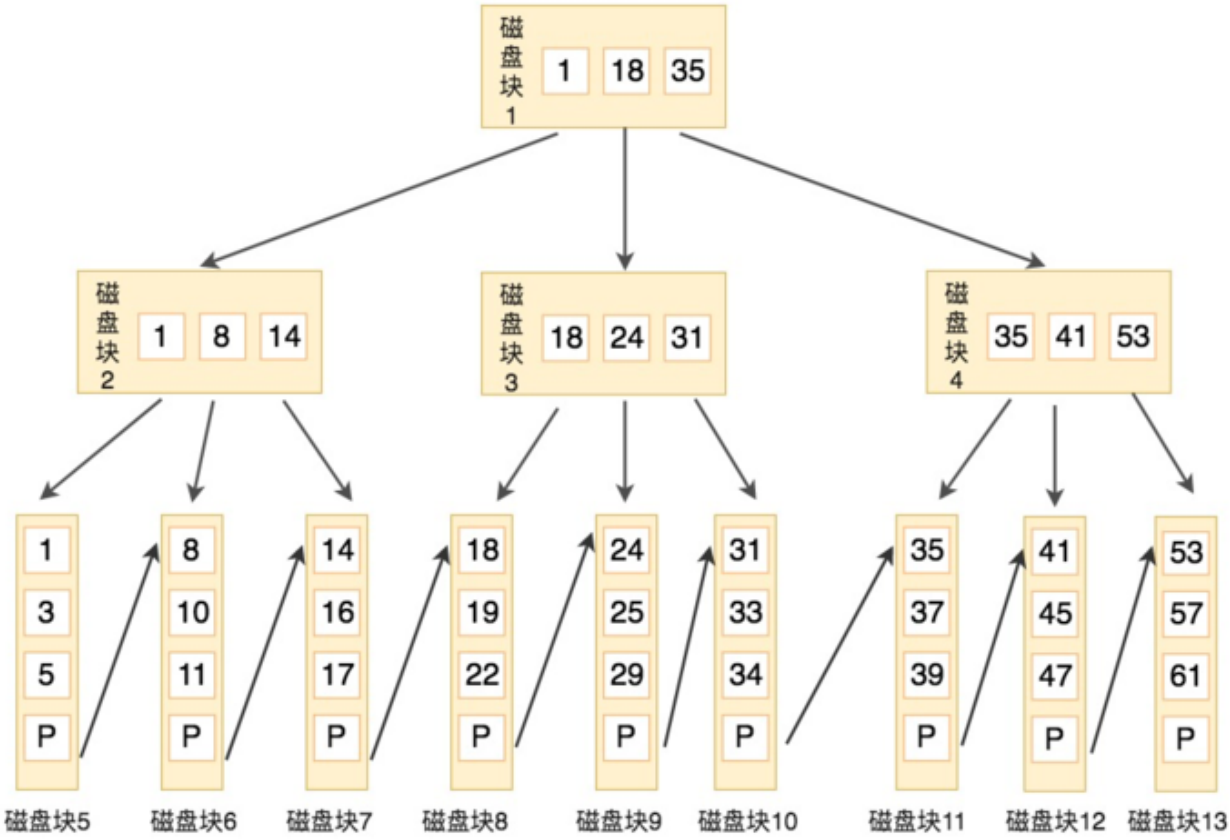


B树简单地说就是多叉树，每个叶子会存储数据，和指向下一个节点的指针。

例如要查找9，步骤如下

1. 我们与根节点的关键字 (17，35）进行比较，9 小于 17 那么得到指针 P1；
2. 按照指针 P1 找到磁盘块 2，关键字为（8，12），因为 9 在 8 和 12 之间，所以我们得到指针 P2；
3. 按照指针 P2 找到磁盘块 6，关键字为（9，10），然后我们找到了关键字 9。

B+树



B+树是B树的改进，简单地说是：只有叶子节点才存数据，非叶子节点是存储的指针；所有叶子节点构成一个有序链表

例如要查找关键字16，步骤如下

1. 与根节点的关键字 (1，18，35) 进行比较，16 在 1 和 18 之间，得到指针 P1（指向磁盘块 2）
2. 找到磁盘块 2，关键字为（1，8，14），因为 16 大于 14，所以得到指针 P3（指向磁盘块 7）
3. 找到磁盘块 7，关键字为（14，16，17），然后我们找到了关键字 16，所以可以找到关键字 16 所对应的数据。

B+树与B树的不同：

1. B+树非叶子节点不存在数据只存索引，B树非叶子节点存储数据
2. B+树使用双向链表串连所有叶子节点，区间查询效率更高，因为所有数据都在B+树的叶子节点，但是B树则需要通过中序遍历才能完成查询范围的查找。
3. B+树每次都必须查询到叶子节点才能找到数据，而B树查询的数据可能不在叶子节点，也可能在，这样就会造成查询的效率的不稳定
4. B+树查询效率更高，因为B+树矮更胖，高度小，查询产生的I/O最少。

这就是MySQL使用B+树的原因，就是这么简单！