**一、MySQL索引（hash与btree的区别）**

**1.1简述**

MySQL存储引擎支持每个表至少16个所有，总索引长度至少为256字节。索引的存储类型目前主要是2种：

MyISAM btree

InnoDB btree

Memory/Heap hash，btree（默认存储引擎使用hash）

**1.2索引hash与btree的区别**

◆ hash检索效率非常高，可一次性定位，btree需要从根节点到枝节点，最后才能访问到页节点，进行多次的I/O访问，效率上hash远胜btree。

◆ hash的限制和弊端

①无法使用范围查找，仅支持 =,<=>,IN,IS NULL,IS NOT NULL等确定值检索。 ②hash索引无法被用来避免数据的排序操作。由于hash索引中存放的是经过hash计算后的hash值，且hash值的大小关系不一定和hash运算前的键值完全一样，所以数据库无法利用索引的数据来避免任何排序运算。

③hash索引不能利用部分索引键查询。对于组合索引，hash在计算hash值的时候是组合索引键合并后再一起计算hash值，而不是单独计算hash值，所以通过组合索引的前面一个或几个索引键进行查询的时候，hash索引无法被利用。

④hash索引在任何时候都不能避免表扫描。hash索引是将索引键通过hash运算之后，将hash运算结果的hash值和所对应的行指针信息存放于一个hash表中，由于不同索引键存在相同hash值，所以即使取满足某个hash键值的数据的记录条数，也无法从hash索引中直接完成查询，还要通过访问表中的实际数据进行对比并得到结果。

⑤hash索引遇到大量hash值相等的情况后性能并不一定就比B-Tree索引高。对于选择性比较低的索引键，如果创建hash索引，那么将会存在大量记录指针信息存在同一个hash值关联，这样定位一条记录就会很麻烦，会浪费多次表数据的访问，性能低下。 ◆ B-Tree索引的物理文件大多都以Balance Tree的结果存储，即实际需要的数据都存放于Tree的leaf node（叶子节点），到达任何一个Leaf Node的最短路径的长度都是完全相同的，所以称为平衡树。InnoDB存储引擎的数据结构实际是B+Tree，即在数据结构的基础上做了很小的改造，在每个Leaf Node上存储索引键信息外，还存储了指向与该Leaf Node相邻的后一个Leaf Node的指针信息，为了加快检索多个相邻Leaf Node的效率。

**1.3 B-Tree与B+Tree**

系统从磁盘读取数据到内存是以磁盘块（block）为基础单位的（一般mysql的单个block大小未16KB，linux为4KB），位于同一个磁盘块中的数据会被一次性读取出来，而不是需要什么取什么。InnoDB存储引擎中有页的概念，页是其磁盘管理的最小单位。可通过innodb\_page\_size将页的大小设置为4k、8k、16k。系统的一个磁盘块的存储空间一般不会很大，因此InnoDB每次申请磁盘空间的时候都会是若干地址连续磁盘块来达到页的大小16KB。InnoDB在把磁盘数据读入到磁盘时会以页为基本单位，在查询数据时如果一个页中的每条数据都能有助于定位数据记录的位置，将会减少磁盘I/O，提高效率。

◆ B-Tree（平衡多路查找树），是为磁盘等外存储设备设计的一种平衡查找树。其特性如下：（一个m阶的B-Tree）

① 每个节点最多有m个子节点。

② 除了根节点和叶子节点外，其他节点至少有Ceil(m/2)个子节点

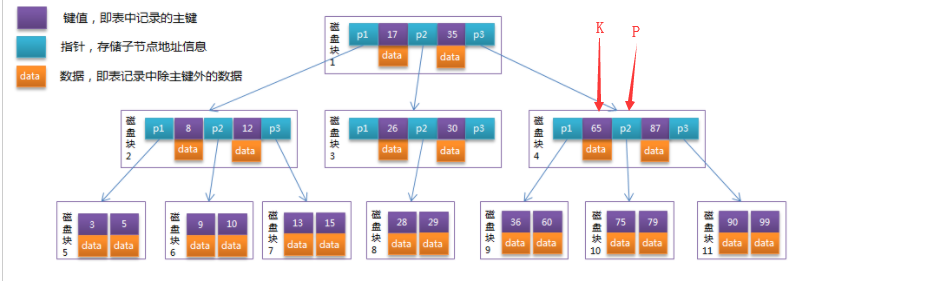
③ 若根节点不是叶子节点，则至少有2个子节点。

④ 所有叶子节点都在同一层，且不包含其他关键字信息。

⑤ 每个非终端节点包含n个关键字信息，n满足：ceil(m/2) - 1 ≤ n ≤ m - 1 。

⑥ Ki（i=1,...n）为关键字，且关键字升序排列。

⑦ Pi（i=1,...n）为指向子树的指针，P(i-1)指向的子树的所有节点关键字均小于Ki，但都大于K(i-1)。



如上图，模拟查找关键字29的过程：

1.根据跟节点找到磁盘块1，读入内存。【磁盘I/O第1次】

2.比较关键字29的区间（17,35），找到磁盘块1的指针P2。

3.根据P2指针找到磁盘块3，读入内存。【磁盘I/O第2次】

4.比较关键字29在区间（26,30），找到磁盘块3的指针P2。

5.根据P2指针找到磁盘块8，读入内存。【磁盘I/O第3次】

6.在磁盘块8中的关键字列表中找到关键字29，去取对应数据。

分析：以上查询需要3次磁盘I/O操作、3次内存查找操作。由于内存中的关键字是一个有序表结构，可利用二分法查找提高效率。而3次磁盘I/O是影响整个B-Tree查找效率的决定性因素。

◆ B+Tree，在B-Tree基础上的一种优化，使其更适合实现外存储索引结构，InnoDB用的就是B+Tree。

由于B-Tree的每个节点不仅包含数据的key值，还有data值。而每个页的存储空间是有限的，如果data数据比较大时将会导致每个节点（即一个页）能存储的key的数量很小，当存储的数据量很大时会导致B-Tree的深度很大，增加查询时磁盘的I/O次数，影响效率。在B+Tree中，所有数据记录节点都是安卓键值大小顺序存放在同一层叶子节点上，而叶子结点上只存储key值信息，这样可以大大增加每个节点存储的key值的数量，降低B+Tree的深度，减少磁盘I/O次数。

B+Tree与B-Tree的不同：

① 非叶子节点只存储键值信息。

② 所有叶子节点之间都有一个链指针。

③ 数据记录都存放在叶子节点上。

