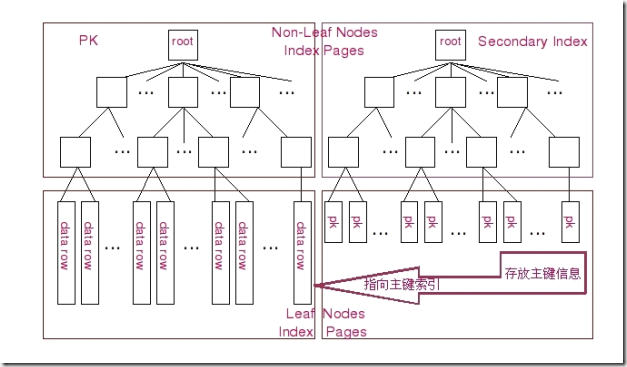
Hash 索引结构的特殊性，其检索效率非常高，索引的检索可以一次定位，不像B-Tree索引需要从根节点到枝节点，最后才能访问到页节点这样多次的IO访问，所以 Hash 索引的查询效率要远高于 B-Tree索引。

可能很多人又有疑问了，既然Hash 索引的效率要比 B-Tree 高很多，为什么大家不都用 Hash 索引而还要使用 B-Tree索引呢？任何事物都是有两面性的，Hash 索引也一样，虽然 Hash 索引效率高，但是 Hash索引本身由于其特殊性也带来了很多**限制和弊端**，主要有以下这些。  
  
（1）Hash索引仅仅能满足"=","IN"和"<=>"查询，不能使用范围查询。  
  
由于 Hash 索引比较的是进行 Hash 运算之后的 Hash值，所以它只能用于等值的过滤，不能用于基于范围的过滤，因为经过相应的 Hash[算法](http://lib.csdn.net/base/datastructure)处理之后的 Hash 值的大小关系，并不能保证和Hash运算前完全一样。  
  
（2）Hash 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 计算之后的 Hash值，而且Hash值的大小关系并不一定和 Hash运算前的键值完全一样，所以[数据库](http://lib.csdn.net/base/mysql)无法利用索引的数据来避免任何排序运算；  
  
（3）Hash索引不能利用部分索引键查询。  
  
对于组合索引，Hash 索引在计算 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 索引也无法被利用。  
  
（4）Hash索引在任何时候都不能避免表扫描。  
  
前面已经知道，**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，还是要通过访问表中的实际数据进行相应的比较，并得到相应的结果。  
  
（5）Hash索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高。

**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于同一个Hash值相关联。这样要定位某一条记录时就会非常麻烦，会浪费多次表数据的访问，而造成整体性能低下

2. B-Tree索引   
  
     B-Tree 索引是 MySQL 数据库中使用最为频繁的索引类型，除了 Archive 存储引擎之外的其他所有的存储引擎都支持B-Tree 索引。不仅仅在 MySQL 中是如此，实际上在其他的很多数据库管理系统中B-Tree索引也同样是作为最主要的索引类型，这主要是因为 B-Tree索引的存储结构在数据库的数据检 索中有非常优异的表现。   
     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 ，而且到任何一个 Leaf Node的最短路径的长度都是完全相同的，所以我们大家都称之为 B-Tree 索引当然，可能各种数据库（或 MySQL的各种存储引擎）在存放自己的 B-Tree 索引的时候会对存储结构稍作改造。如 Innodb 存储引擎的 B-Tree索引实际使用的存储结构实际上是 B+Tree ，也就是在 B-Tree数据结构的基础上做了很小的改造，在每一个   
Leaf Node 上面出了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，这主要是为了加快检索多个相邻 Leaf Node 的效率考虑。   
     在 Innodb 存储引擎中，存在两种不同形式的索引，一种是 Cluster 形式的主键索引（ Primary Key），另外一种则是和其他存储引擎（如 MyISAM 存储引擎）存放形式基本相同的普通 B-Tree 索引，这种索引在 Innodb存储引擎中被称为 Secondary Index。下面我们通过图示来针对这两种索引的存放   
形式做一个比较。   
  
       
  
     图示中左边为 Clustered 形式存放的 Primary Key ，右侧则为普通的 B-Tree 索引。两种 Root Node和 Branch Nodes 方面都还是完全一样的。而 Leaf Nodes 就出现差异了。在 Prim中， Leaf Nodes存放的是表的实际数据，不仅仅包括主键字段的数据，还包括其他字段的数据据以主键值有序的排列。而 Secondary Index则和其他普通的 B-Tree 索引没有太大的差异，Leaf Nodes 出了存放索引键 的相关信息外，还存放了 Innodb的主键值。   
  
     所以，在 Innodb 中如果通过主键来访问数据效率是非常高的，而如果是通过 Secondary Index 来访问数据的话，Innodb 首先通过 Secondary Index 的相关信息，通过相应的索引键检索到 Leaf Node之后，需要再通过Leaf Node 中存放的主键值再通过主键索引来获取相应的数据行。MyISAM存储引擎的主键索引和非主键索引差别很小，只不过是主键索引的索引键是一个唯一且非空 的键而已。而且 MyISAM 存储引擎的索引和Innodb 的 Secondary Index 的存储结构也基本相同，主要的区别只是 MyISAM 存储引擎在 Leaf Nodes上面出了存放索引键信息之外，再存放能直接定位到 MyISAM 数据文件中相应的数据行的信息（如 Row Number），但并不会存放主键的键值信息