MySQL索引背后的数据结构及算法原理

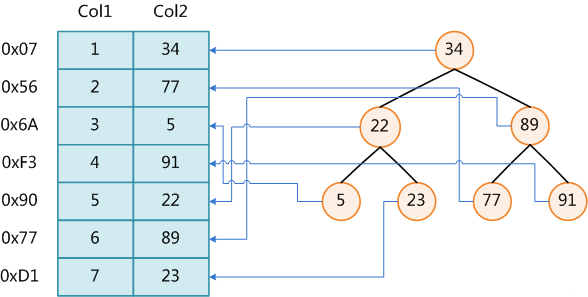
MySQL支持多种存储引擎，各种存储引擎对索引的支持不相同，因此MySQL支持多种索引类型，如BTree索引、哈希索引、全文索引等等，下面我们重点讨论BTree所以。

1、索引的本质

索引(Index)是帮助MySQL高效获取数据的数据结构，本质就是数据结构。

查询数据的速度能尽可能的快，那就要用到查询算法，常见的查询算法顺序查找(o(n))、二分查找(要求被检索的数据有序)、二叉树查找(只能应用在二叉查找树上)，数据本身的组织结构不可能完全满足各种数据结构，所以，在数据之外，数据库系统还维护了满足特定查找算法的数据结构，

这些数据结构以某种方式指向数据，这种数据结构，就是索引。



为了加快Col2的查找，维护一个二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，复杂度o(log2n)。

2、B-Tree(多叉平衡查找树)

目前大部分的数据库系统及文件系统都采用B-Tree或其变种B+Tree作为索引结构。

B-Tree的定义：(d为大于1的一个正整数，称为度；h为一个正整数，称为高度)

1）每个非叶子节点由n-1个key和n个指针组成，其中d<=n<=2d

2）每个叶子节点最少包含一个key和两个指针，最多包含2d-1个key和2d个指针，叶节点的指针均为null

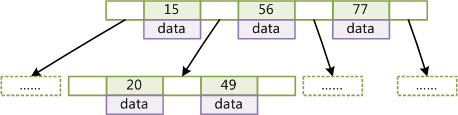
3）所有叶节点具有相同的深度，等于树高h

4）key和指针互相间隔，节点两端是指针

5）一个节点中的key从左到右递增排列

6）每个指针要么为null，要么指向另外一个节点

7）……



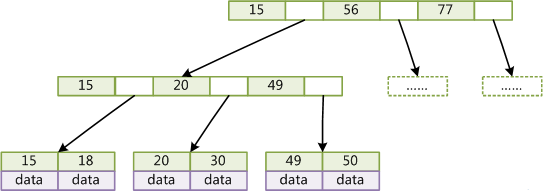
上图为一个d=2的B-Tree

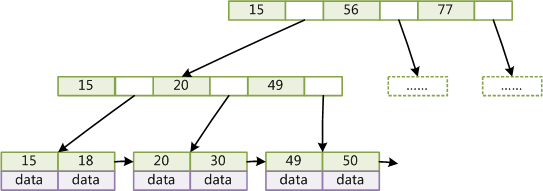
查找流程：从根节点进行二分查找，如果找到则返回对应节点的data，否则对相应区间的指针指向的节点递归查找，直到找到或找到null指针

3、B+Tree

B+Tree的定义：

1）每个节点的指针上限为2d而不是2d+1

2）内节点不存储data，只存储key，叶子节点不存储指针

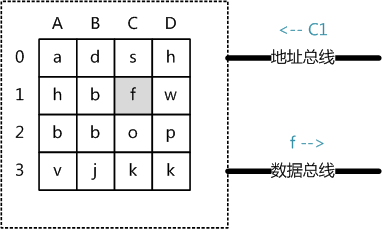
带有顺序访问指针的B+Tree

如上图，每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的B+Tree，这样是为了提高区间访问的性能，如果要查询key为从18到49的所有数据记录，当找到18后，只需要顺着节点和指针顺序遍历就可以一次性访问到所有数据节点。

为什么说普遍采用B+Tree作为索引结构？需要了解磁盘相关的知识

一般来说，索引文件也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储在磁盘上。那这样的话，索引查找的过程中就要产生磁盘I/O消耗，相较于内存操作，I/O操作的消耗要高几个数量级。

主存存取原理

现代计算机主存基本都是随机读写存储器(RAM)，从抽象的角度来看，它是一系列的存储单元组成的矩阵，每个存储单元存储固定大小的数据。每个存储单元有唯一的地址，其实际构造比较复杂，简化理解为：通过一个行地址和一个列地址可以唯一定位一个存储单元

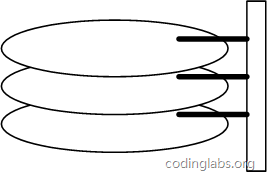
读主存：将地址信号放到地址总线上传给主存，主存读到地址信号后，解析信号并定位到指定存储单元，然后将此存储单元数据放到数据总线上，供其他部件读取。

写主存：将要写入单元地址和数据分别放在地址总线和数据总线上，主存读取两个总线的内容，做相应的写操作。

两次存取的数据的“距离”不会对时间有任何影响

磁盘存取原理：

一个磁盘由大小相同且同轴的圆形盘片组成，磁盘可以转动（各个磁盘必须同步转动）。在磁盘的一侧有磁头支架，磁头支架固定了一组磁头，每个磁头负责存取一个磁盘的内容。磁头不能转动，但是可以沿磁盘半径方向运动（实际是斜切向运动），每个磁头同一时刻也必须是同轴的，即从正上方向下看，所有磁头任何时候都是重叠的（不过目前已经有多磁头独立技术，可不受此限制）。



盘片被划分成一系列同心环，圆心是盘片中心，每个同心环叫做一个磁道，所有半径相同的磁道组成一个柱面。磁道被沿半径线划分成一个个小的段，每个段叫做一个扇区，每个扇区是磁盘的最小存储单元。为了简单起见，我们下面假设磁盘只有一个盘片和一个磁头。

当需要从磁盘读取数据时，系统会将数据逻辑地址传给磁盘，磁盘的控制电路按照寻址逻辑将逻辑地址翻译成物理地址，即确定要读的数据在哪个磁道，哪个扇区。为了读取这个扇区的数据，需要将磁头放到这个扇区上方，为了实现这一点，磁头需要移动对准相应磁道，这个过程叫做寻道，所耗费时间叫做寻道时间，然后磁盘旋转将目标扇区旋转到磁头下，这个过程耗费的时间叫做旋转时间。

局部性原理与磁盘预读

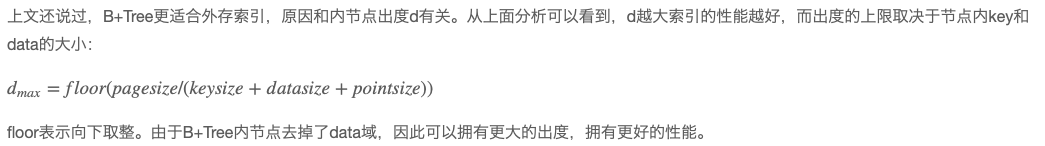
为了提高磁盘的效率，尽量减少磁盘的I/O次数，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。

局部性原理：当一个数据被用到时，其附近的数据也通常会马上被使用。

不需要寻道时间，只需很少的旋转时间，所以磁盘顺序读取的效率很高

预读的长度一般为页(page)的整数倍。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分隔为连续的大小相等的块，每个存储块称为一页（许多操作系统中页的大小通常为4k）主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后异常返回。

一个节点的大小等于一页，每个节点只需要一次I/O就可以完全载入。加上计算机存储分配是按页对齐的。



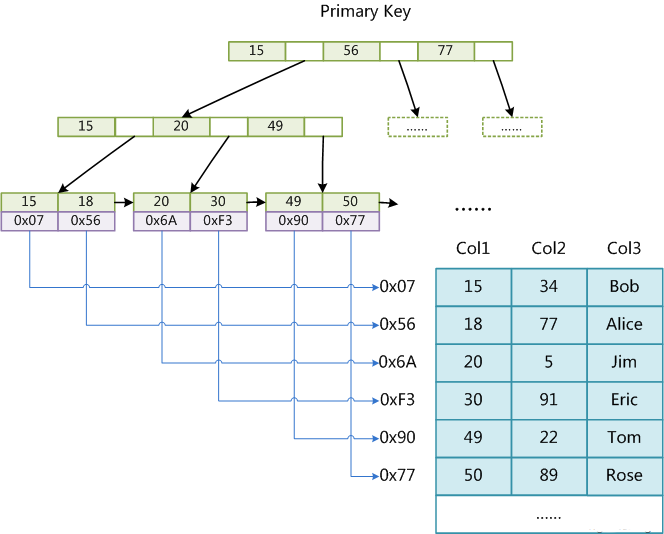
每个节点的大小确定了，是一页的大小，那么去掉data域就可以获得更大的出度。

4、MySQL索引的实现

不同的存储引擎使用的索引结构不同。下面分别介绍MyISAM和InnoDB存储引擎的索引实现。

MyISAM索引实现

使用B+Tree作为索引结构，叶节点的data域存放数据记录的地址



MyISAM中索引检索的算法为首先按照B+Tree搜索算法搜索索引，如果指定的Key存在，则取出其data域的值，然后以data域的值为地址，读取相应数据记录。

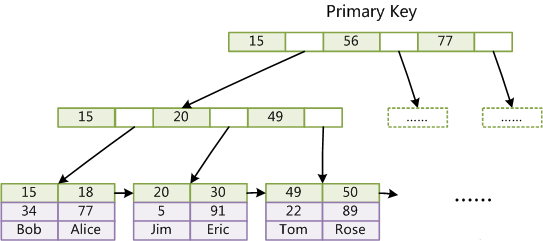
MyISAM的索引方式也叫做“非聚集”的，之所以这么称呼是为了与InnoDB的聚集索引区分。

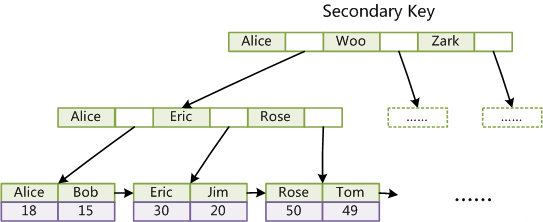
InnoDB索引实现

它也使用B+Tree作为索引结构，但是具体实现方式与MyISAM截然不同。

区别：1）它的数据文件就是索引文件，但是MyISAM的数据文件和索引文件是分开的

2）InnoDB的辅助索引都引用主键作为data域





data域保存了完整的数据记录，这种索引称为聚集索引，因为InnoDB的数据文件本身要按主键聚集，所以InnoDB要求表必须有主键，如果没有显示的指定，mysql会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在，则mysql会自动为innoDB表生成一个隐含字段作为主键。

聚集索引这种实现方式使得按主键的搜索十分高效，但是辅助索引搜索需要检索两遍索引：首先检索辅助索引获得主键，然后用主键到主索引中检索获得记录。

思考：

1、为什么不建议使用过长的字段作为主键

2、为什么在innodb中不建议使用非单调的字段作为主键