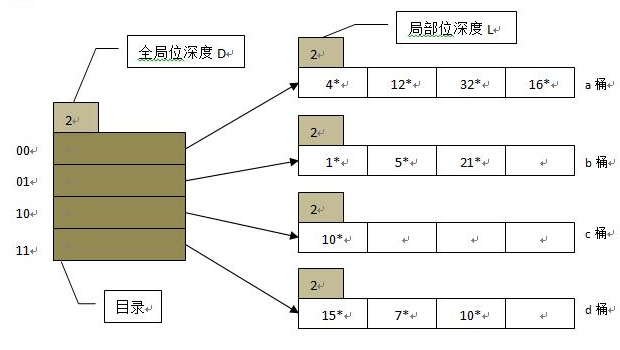
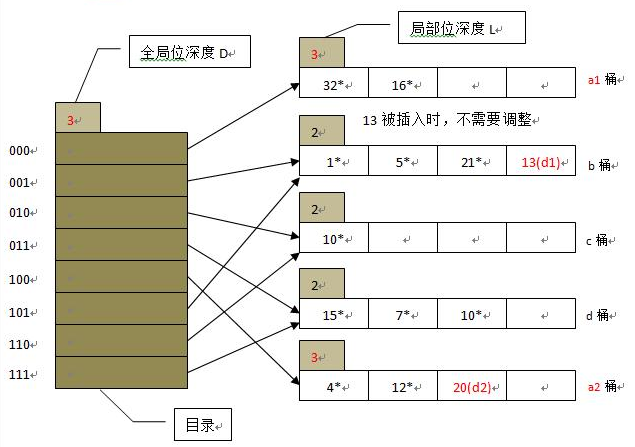
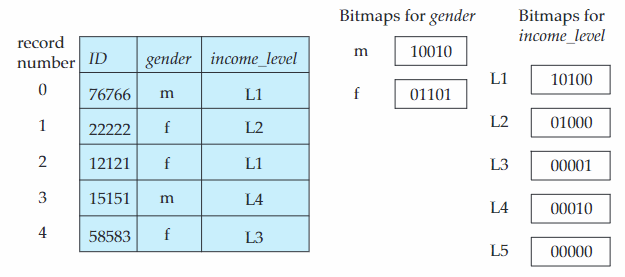
11 Indexing and Hashing

1. Basic Concepts  
   索引分为顺序索引(ordered indixes)和散列(has indices)索引，前者基于值得顺序；后者将值平均分布到若干bucket中，值所属的bucket由散列函数决定。  
   索引和散列的实现技术有多种，但没有哪一种是绝对最好的，每种方式有其最适合的场景，可通过这几个方面来进行评估：访问类型（能有效支持的访问类型，如特定值查找、范围查找）、访问时间、插入时间、删除时间、空间开销（索引额外占用的空间）。
2. 顺序索引  
   为了快速随机访问记录，可以使用顺序索引。用于在文件中查找记录的属性或属性集称为搜索码(search key)，每个索引结构与一个特定的搜索码关联，并按顺序存储搜索码的值，并将每个搜索码与对应的记录关联起来。  
   被所以的记录本身也可以按一定的顺序存储，如果记录按照某个搜索码指定的顺序排序，则该搜索码对应的索引称为聚集索引(clustering index)或主索引(primary index)，聚集索引的搜索码一般是主键，但不是必须这样。搜索码指定的顺序与记录的物理存储顺序不同的索引称为非聚集索引(noclustering index)或辅助索引(secondary index)。
   1. 稠密索引和稀疏索引  
      索引项由搜索码和指向具有该搜索码值的若干条记录的指针构成，指针包含block的标识和标识磁盘块内记录的块内偏移量。顺序索引又分为稠密索引(dense index)和稀疏索引(sparse index)。  
      稠密索引中，每个搜索码都有一个索引项。索引项包含搜索码值以及指向具有该搜索码值的第一条数据记录的指针。如果是稠密聚集索引，具有相同搜索码值的其余记录可以顺序地存储在第一条数据记录之后；对于非聚集索引，则需要没每条搜索码值建立索引。  
      稀疏所以中，只为搜索码的某些值建立索引项。只有索引是聚集索引时才可以使用稀疏索引，这样即使索引中没有要查找的搜索码，也可以根据索引确定搜索码的区间，进而找到要查找的记录。
   2. 多级索引  
      如果记录数很多，其索引也会变得很大。假定100条索引占用一个4K的block，那么100,000,000条记录的索引的体积会高达4G。只有将全部索引都装载到内存，才能最大化索引的速度，如果计算机内存小于4G，那么索引就无法完全放入内存中，即使内存够大，也还要供给其他应用程序使用。  
      在数据量很大、索引文件较大时可以使用多级索引，即在原先内层索引的基础上再建立若干外层索引，外层索引与内层索引之间的关系同前面索引与数据文件的关系一样。多级索引与树结构紧密相关，比如用于内存索引的二叉树。
   3. 索引的更新  
      在插入、删除数据的时候，对应的索引需要更新；而数据被修改时，其索引项不是直接修改，而是分为“删除就搜索码值，插入新搜索码值”这两步执行的，所以索引的更新只涉及插入和删除。
      1. 插入
         1. 稠密索引  
            如果插入数据的搜索码值在索引中不存在，则直接插入索引。  
            如果插入的搜索码值已经存在且指向多条数据，则在这条索引项下新增指向插入数据的指针。  
            如果插入的搜索码值已经存在但只指向第一条包含该搜索码值的记录，则仅将插入数据放到具有相同搜索码值的其他记录之后。
         2. 稀疏索引  
            假设稀疏索引为每个块保存了一个索引项。如果插入数据时，系统需要新创建一个块，则在索引中插入针对新块中第一条记录的搜索码的索引项。如果新插入的数据刷新了块中记录的最小搜索码值（按照搜索码的顺序存储），这个指向这个块的索引项需要被更新；其余情况不需要更新索引。
      2. 删除
         1. 稠密索引  
            如果待删除的记录的搜索码值是唯一的，则删除对应的索引。  
            如果待删除记录的搜索码值不唯一，且索引项指向所有具有该搜索码值的记录，则只删除索引项中指向待删除记录的指针。  
            如果待删除记录的搜索码值不唯一、索引项仅指向第一条具有该搜索码值的记录，而且第一条记录待删除，在删除这条记录的同时，将索引项指向下一条具有该搜索码值的记录。
         2. 稀疏索引  
            如果不存在包含被删除记录的索引项，则不做任何操作。  
            如果被删除的记录是具有该搜索码值的唯一记录，则索引项A指向下一个包含该搜索码值的记录，但如果下一个包含该搜索码值的记录已经有索引项，则删除索引项A。  
            如果被删除的记录并非是具有该搜索码值的唯一记录，则索引项指向下一条具有相同搜索码值的记录。

   6. 静态散列  
      顺序文件组织的缺点之一是必须通过访问索引或使用二分法搜索来定位数据，这需要较多的I/O操作。基于散列技术的文件组织方式则不需要访问索引结构，散列也提供了一种组织索引的方式。  
      在散列(hash)技术中，用桶(bucket)来表示能存储一条或多条记录的存储单元。如果K代表所有搜索码的集合，B代表所有bucket的集合，则散列函数h表示一个从K到B的映射函数。  
      插入搜索码为Ki的记录时，通过散列函数计算h(Ki)得出bucket的地址，如果这个bucket还有空间，就将数据插入。  
      查询Ki时，也是先通过h(Ki)得出bucket的地址，但bucket中往往有多条记录，这时就需要进一步根据搜索码在bucket内部搜索。  
      散列可以有两种用途，在散列文件组织中，通过散列函数直接定位记录所在的磁盘块；在散列索引组织中，把搜索码和指针组织成一个散列文件结构。
      1. 散列函数  
         合理地选择散列函数非常重要，否则可能导致记录被集中映射到少数几个bucket的情况。要求散列函数的分布特性是均匀、随机的，既每个bucket被分配到的记录数应该是相等的，而且分配结果与搜索码本身的顺序无关。  
         典型的散列函数是根据搜索码的二进制值进行计算的，比如可以计算二进制所有位的和，然后取模。一个结果良好设计的散列函数应不受记录数量的影响，而具有稳定的搜索效率。
      2. Bucket溢出的处理  
         如果记录被映射到一个Bucket时Bucket以及没有可用空间，就会发生溢出。溢出的原因可能是因为随着记录数的增长，没有足够的bucket；也可能是因为散列函数设计不合理或者存在太多相同的搜索码，导致记录被集中映射到某些bucket，而一个bucket能容纳的记录是有限的，这种情况称为桶偏斜(bucket skew)。  
         为了应对bucket溢出，可以在确定bucket数量时留一定的余量，但这会造成空间浪费；也可以使用溢出桶(overflow bucket)来接收溢出记录：一旦发生溢出，就新增一个溢出桶，以接收溢出记录，溢出桶与原始桶构成链表(overflow chaining)，于是查找数据时，要增加对是否存在溢出桶的探查，如果存在，则进一步在溢出桶中搜索。  
         这种静态散列的缺点在于必须在设计阶段确定好bucket的数量，随着记录的数的增加或收缩，bucket数无法跟随变化，会造成溢出或空间浪费。
      3. 散列索引  
         除了散列文件组织，还可以用散列的方式组织索引。将散列函数作用于搜索码以确定对应的桶，然后将此搜索码以及相应的指针存入。
   7. 动态散列  
      静态散列要求桶的数目始终固定，那么在确定桶数目和选择散列函数时，如果桶数目过小，随着数据量增加，性能会降低；如果留一定余量，又会带来空间的浪费；或者定期重组散列索引结构，但这是一项开销大且耗时的工作。为了应对这些问题，为此提出了几种动态散列(dynamic hashing)技术，可扩展动态散列(extendable hashing)是其一。
      1. 可扩展动态散列  
           
         用一个数组来存储桶指针的目录，数组的位数为2的D次方，桶的容量为2的L次方，D和L分别称为全局位深度和局部位深度。每次发生桶溢出时，溢出桶分裂，容量变为2的L+1次方，其它桶的容量保持不变，同时数据目录的深度变为D+1。扩展容量时，只是调整了局部的桶容量和目录的容量，性能开销比较小。  
         上图中，目录深度为2，目录项有4个。然后开始插入数据d1和d2，假定h(d1)=13、h(d2)=20，由于13=1101，且全局位深度为2，则根据后两位01确定应插入b桶，b桶有空间，可直接插入。20=10100，应插入a桶，但a桶以及满了，于是开始分裂，a桶的局部位深度变为3，容量扩展为8，如果扩展后的局部位深度超过了全局位深度，则全局位深度等于这个最大的局部位深度，于是全局位深度也随之变为3。  
           
         如上图所示，a桶分裂为a1、a2，目录变为三位，对原来a桶中的元素进行重组，由于目录位多了一位，要根据000、100来分别存储到a1、a2桶。虽然目录发生了翻倍，但未进行分裂的桶的局部深度仍然为2，所以会有多个目录项指向这些桶，比如001、101的后两位都是01，都指向b桶。  
         1. 对于查找操作，根据当前的全局位深度，通过目录直接定位到桶地址，随后在桶内部逐一查找。
         2. 对于删除操作，与查找操作类似，删除元素后，如果发现桶变为空，可与其兄弟桶进行合并，并使局部位深度减一。如果所有的局部位深度都小于全局位深度，则目录数组也进行收缩。  
            https://www.cnblogs.com/kegeyang/archive/2012/04/05/2432608.html
      3. 静态散列与动态散列对比  
         与静态散列相比，动态散列的主要优势在于其性能不会随着记录数增长而下降，另外还具有最小的空间占用。缺点在于它会额外增加一次查询定位，因为在查询bucket本身前，需要先查找目录来定位bucket。  
         另一种动态散列技术-线性散列(linear hashing)可以避免额外的查询定位，但可能这种方式需要更多的溢出桶，日后学习。
   8. 顺序索引与散列的适用场景  
      每种索引结构都有其优缺点。如果是select \* from a where b=c这样的定值查询，散列比顺序索引跟适合，顺序索引会随着记录数的增加而性能降低，散列则相对稳定。而对于where b>c and b<d这样的范围搜索，则顺序索引更适合，散列的随机特性使得无法定位搜索的bucket。所以散列只适合根据搜索码搜索且不是范围查询的场合。
   9. 位图索引  
      位图索引(bitmap indices)是一种专为多个键的简单查询而设计的。应用位图索引的前提是记录必须被按顺序编号，一般从0开始。给出编号n，必须能够很容易的找到对应的记录，如果记录被存放在连续的块，可以将编号n转换成块编号+块内偏移的表示以快速定位记录位置。
      1. 位图索引的结构  
           
         位图索引用一个位来对应一条记录，这便是记录需要被编号的原因。instructor\_info表如上图，性别的值有男、女两种，收入等级则划分为5级，既有5种值。在给性别属性建立位图索引时，就会分别为male和female建立，对于male位图来说，如果一条记录的性别为male，则位图上对应的位会置1，female、收入等级位图也采用相同的做法。  
         位图索引的优势体现在根据多个键的查询的时候，比如查询where gender=’f’ and income\_level=’L2’，只需将f的bitmap和L2的bitmap取交集即可。  
         此外，在进行数据分析时经常需要统计符合某些条件的记录的数量，使用bitmap也可以很方便地实现，只需统计交集中值为1的位的数目。  
         删除记录的时候会使数据序列产生间隙，但逐个移动数据消除间隙开销很大，所以引入一个新的存在位图(existence bitmap)，在间隙对应的位置1。新增的数据将被追加到尾部，这样不会影响已有记录的顺序。