

11 索引与散列

Author: 中山大学 17数据科学与计算机学院 YSY

<https://github.com/ysyisyourbrother>

11.1 基本概念

两种基本索引类型:

- **顺序索引**: 基于值的顺序排序
- **散列索引**: 基于将值平均分布到若干散列桶中, 一个值所属的散列桶是由一个函数决定的, 这个函数称为**散列函数**

对每种技术的评价基于以下因素:

- **访问类型**(access type): 能有效支持的访问类型。访问类型可以包括找到具有特定属性值的记录, 以及找到属性值落在某个特定范围内的记录。
- **访问时间**(access time): 在查询中使用该技术找到一个特定数据项或数据项集所需的时间。
- **插入时间**(insertion time): 插入一个新数据项所需的时间。该值包括找到插入这个新数据项的正确位置所需的时间, 以及更新索引结构所需的时间。
- **删除时间**(deletion time): 删除一个数据项所需的时间。该值包括找到待删除项所需的时间, 以及更新索引结构所需的时间。
- **空间开销**(space overhead): 索引结构所占用的额外存储空间。倘若存储索引结构的额外空间大小适度, 通常牺牲一定的空间代价来换取性能的提高是值得的。

用于在文件中查找记录的属性或属性集称为**搜索码**

11.2 顺序索引


如果文件按照某个搜索码指定的顺序排序, 那么该搜索码对应的索引称为**聚集索引**, 也称为**主索引**

搜索码制定的顺序与文件中记录的物理顺序不同的索引称为**非聚集索引**或**辅助索引**

索引顺序文件: 搜索码上有聚集索引的文件, 搜索码通俗的说就是搜索数据库内容时用的, 通常就是索引。

example:

10101	Srinivasan	Comp. Sci.	65000		
12121	Wu	Finance	90000		
15151	Mozart	Music	40000		
22222	Einstein	Physics	95000		
32343	El Said	History	60000		
33456	Gold	Physics	87000		
45565	Katz	Comp. Sci.	75000		
58583	Califieri	History	62000		
76543	Singh	Finance	80000		
76766	Crick	Biology	72000		
83821	Brandt	Comp. Sci.	92000		
98345	Kim	Elec. Eng.	80000		



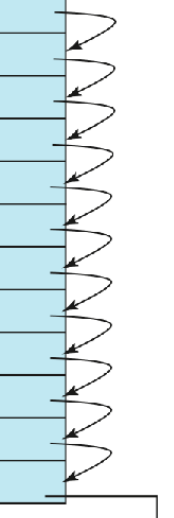
11.2.1 稠密索引和稀疏索引

索引项或索引记录由一个搜索码值和指向具有该搜索码的一条或多条记录的指针构成

其实就是外部的指针，可以快速的找到对应搜索码值的记录。

- **稠密索引**：在稠密索引中，文件中的每个搜索码值都有一个索引项

10101		→	10101	Srinivasan	Comp. Sci.	65000	
12121		→	12121	Wu	Finance	90000	
15151		→	15151	Mozart	Music	40000	
22222		→	22222	Einstein	Physics	95000	
32343		→	32343	El Said	History	60000	
33456		→	33456	Gold	Physics	87000	
45565		→	45565	Katz	Comp. Sci.	75000	
58583		→	58583	Califieri	History	62000	
76543		→	76543	Singh	Finance	80000	
76766		→	76766	Crick	Biology	72000	
83821		→	83821	Brandt	Comp. Sci.	92000	
98345		→	98345	Kim	Elec. Eng.	80000	



- **稀疏索引**：在稀疏索引中，只为搜索码的某些值建立索引项

使用的前提是：关系按照搜索码的顺序存储

查找的时候找到最大的小于等于待查找的码值的搜索码，然后从该位置开始，沿着文件中的指针查找。

Biology		76766	Crick	Biology	72000	
Comp. Sci.		10101	Srinivasan	Comp. Sci.	65000	
Elec. Eng.		45565	Katz	Comp. Sci.	75000	
Finance		83821	Brandt	Comp. Sci.	92000	
History		98345	Kim	Elec. Eng.	80000	
Music		12121	Wu	Finance	90000	
Physics		76543	Singh	Finance	80000	
		32343	El Said	History	60000	
		58583	Califieri	History	62000	
		15151	Mozart	Music	40000	
		22222	Einstein	Physics	95000	
		33465	Gold	Physics	87000	

稀疏索引占空间小，维护的成本也低

11.2.2 多级索引

具有两级或两级以上的索引称为**多级索引**

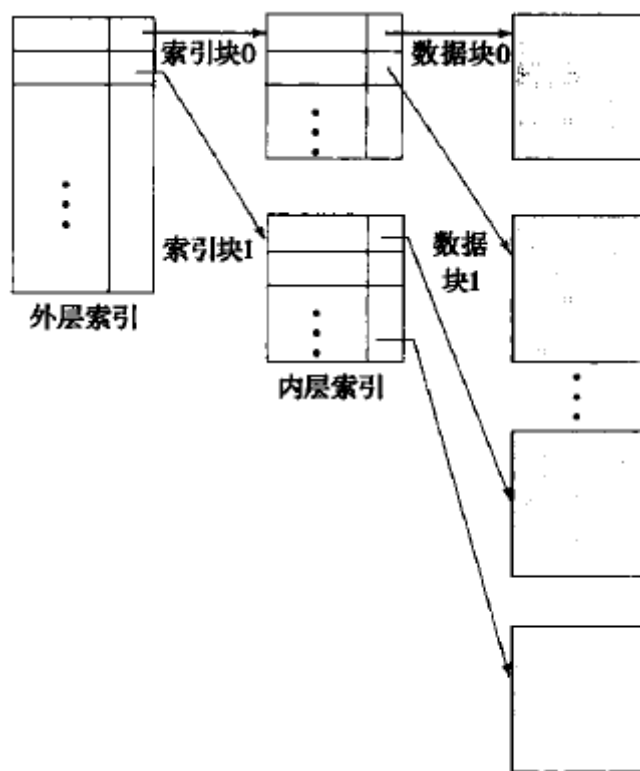


图 11-5 二级稀疏索引

11.2.3 索引的更新

每当文件有记录更新时，索引都需要更新

- **插入。**系统首先用出现在待插入记录中的搜索码值进行查找，并根据索引是稠密索引还是稀疏索引而进行下一个操作。
 - **稠密索引：**
 1. 如果该搜索码值不在索引中，系统就在索引中合适的位置插入具有该搜索码值的索引项。
 2. 否则进行如下操作：
 - a. 如果索引项存储的是指向具有相同搜索码值的所有记录的指针，系统就在索引项中增加一个指向新记录的指针。
 - b. 否则，索引项存储一个仅指向具有相同搜索码值的第一条记录的指针，系统把待插入的记录放到具有相同搜索码值的其他记录之后。
 - **稀疏索引：**我们假设索引为每个块保存一个索引项。如果系统创建一个新的块，它会将新块中出现的第一个搜索码值（按照搜索码的顺序）插入到索引中。另一方面，如果这条新插入的记录含有块中的最小搜索码值，那么系统就更新指向该块的索引项；否则，系统对索引不做任何改动。
- **删除。**为删除一条记录，系统首先查找要删除的记录，然后下一步的操作取决于索引是稠密索引还是稀疏索引。
 - **稠密索引：**
 1. 如果被删除的记录是具有这个特定搜索码值的唯一的一条记录，系统就从索引中删除相应的索引项。
 2. 否则采取如下操作：
 - a. 如果索引项存储的是指向所有具有相同搜索码值的记录的指针，系统就从索引项中删除指向被删除记录的指针。
 - b. 否则，索引项存储的是指向具有该搜索码值的第一条记录的指针。在这种情况下，如果被删除的记录是具有该搜索码值的第一条记录，系统就更新索引项，使其指向下一条记录。
 - **稀疏索引：**
 1. 如果索引不包含具有被删除记录搜索码值的索引项，则索引不必做任何修改。
 2. 否则系统采取如下操作：
 - a. 如果被删除的记录是具有该搜索码值的唯一记录，系统用下一个搜索码值（按搜索码顺序）的索引记录替换相应的索引记录。如果下一个搜索码值已经有一个索引项，则删除而不是替换该索引项。
 - b. 否则，如果该搜索码值的索引记录指向被删除的记录，系统就更新索引项，使其指向具有相同搜索码值的下一条记录。

11.2.4 辅助索引

辅助索引（非聚集索引）必须是稠密索引，对每个搜索码值都有一个索引项，因为它不是按顺序组织文件，不能线性查下去。

一般如果聚集索引的搜索码不是候选码（就是不能唯一指向一个记录），使用稀疏索引即可。

如果辅助索引的搜索码不是候选码，就是说这个搜索码可能对应多个记录，这个时候不能找第一个然后顺序找下去，因此需要每一条记录一个指针。不过在索引中每个搜索码值应该是唯一的，但如果每个搜索码可能有多个记录的时候，就很难表示。

我们可以用一种**间接指针层**的方法实现，指针不直接指向记录，而是指向一个桶，这个桶包含了这个候选码对应的所有记录的指针，比如下图的搜索码是salary

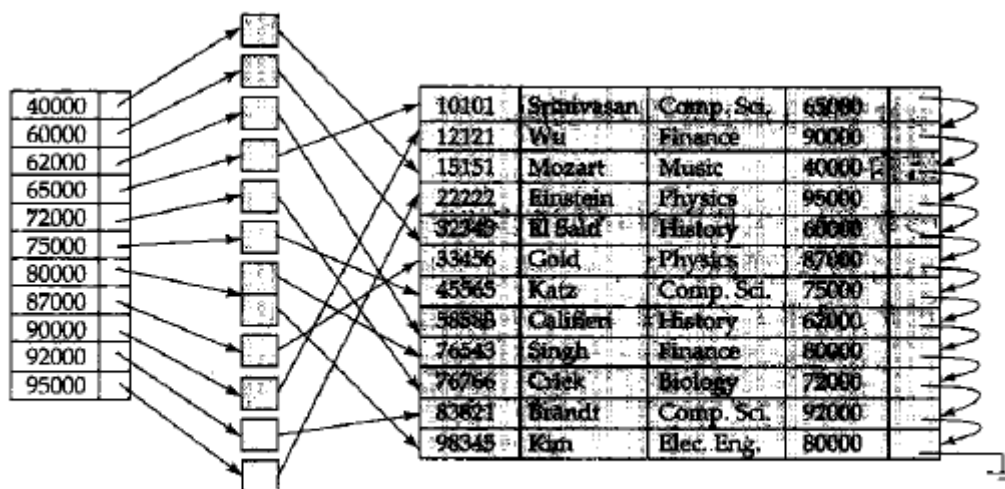


图 11-6 *instructor* 文件的辅助索引，基于非候选码 *salary*

11.2.5 多码上的索引

一个包含多个属性的搜索码被称为复合搜索码

11.3 B+树索引文件

读课本例子

11.6 静态散列

基于散列技术的文件组织使我们能够避免访问索引结构

K是搜索码的集合，B是桶的集合，散列函数目标是从K到B的一个映射。

桶：表示能存储一条或多条记录的一个存储单位

散列文件组织中，我们通过计算所需记录搜索码值上的一个函数直接获得包含该记录磁盘快地址

散列索引组织中，我们把搜索码以及与它们相关联的指针组织成一个散列文件结构

11.6.1 散列函数

我们希望搜索码值分配到桶中并且具有以下分布特性的散列函数：分布是均匀的、随机的

11.6.2 桶溢出处理

如果桶没有足够的空间，就会发生**桶溢出**，有以下几个可能原因：

- **桶不足**
- **偏斜**（某些桶分配到的记录比其它的桶多）

偏斜发生的原因：

- 多条记录可能具有相同的搜索码
- 所选的散列函数可能会造成搜索码的分布不均，多个搜索码映射到同一个桶

我们用**溢出桶**来处理溢出问题，这种链接列表的溢出处理称为**溢出链**

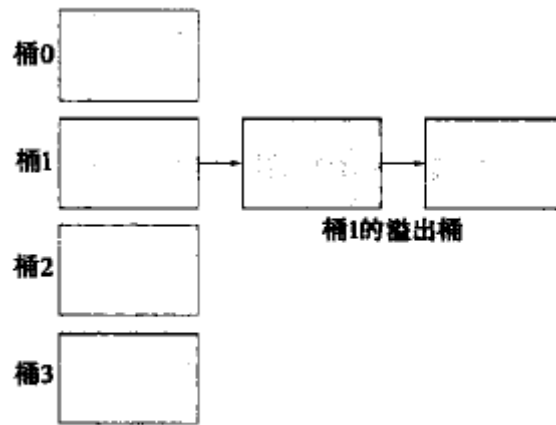


图 11-24 散列结构中的溢出链

上面这种形式的散列结构称为**闭地址**，下面这种方法称为**开地址**

桶集合是固定的，没有溢出链

当一个桶满了以后，系统将记录插入到其它桶中。一种策略是用下一个有空间的桶，这样可以用**线性探查法**找到

该方法在数据库系统不使用，因为删除操作太麻烦

11.6.3 散列索引

散列索引将搜索码及其相应的指针组织成散列文件结构，散列函数用来确定对应的桶，然后把搜索码以及指针放进桶里。这样查找的时候就找到桶然后在桶中顺序查找。

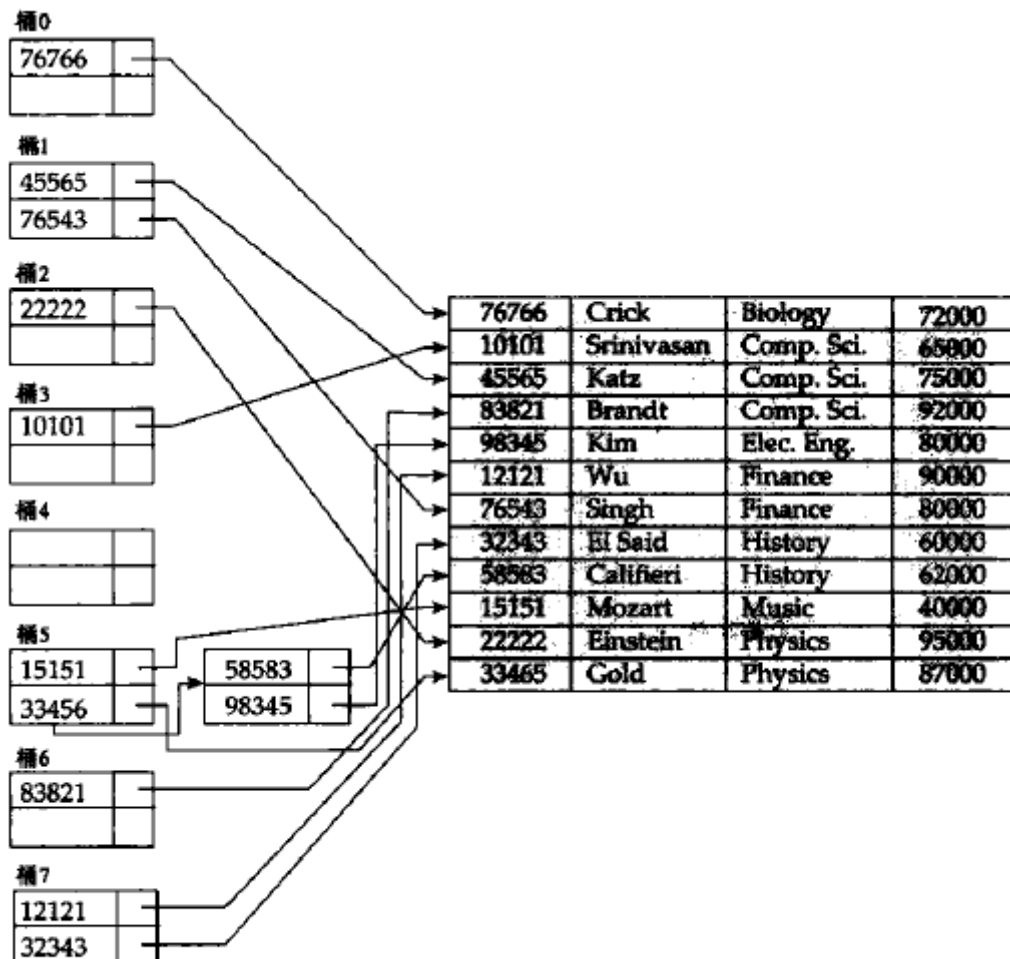


图 11-25 *instructor* 文件中在搜索码 ID 上的散列索引

11.7 动态散列

在静态散列技术中，桶数量是固定的。

动态散列技术允许散列函数动态改变，以适应数据库增大或缩小的需要

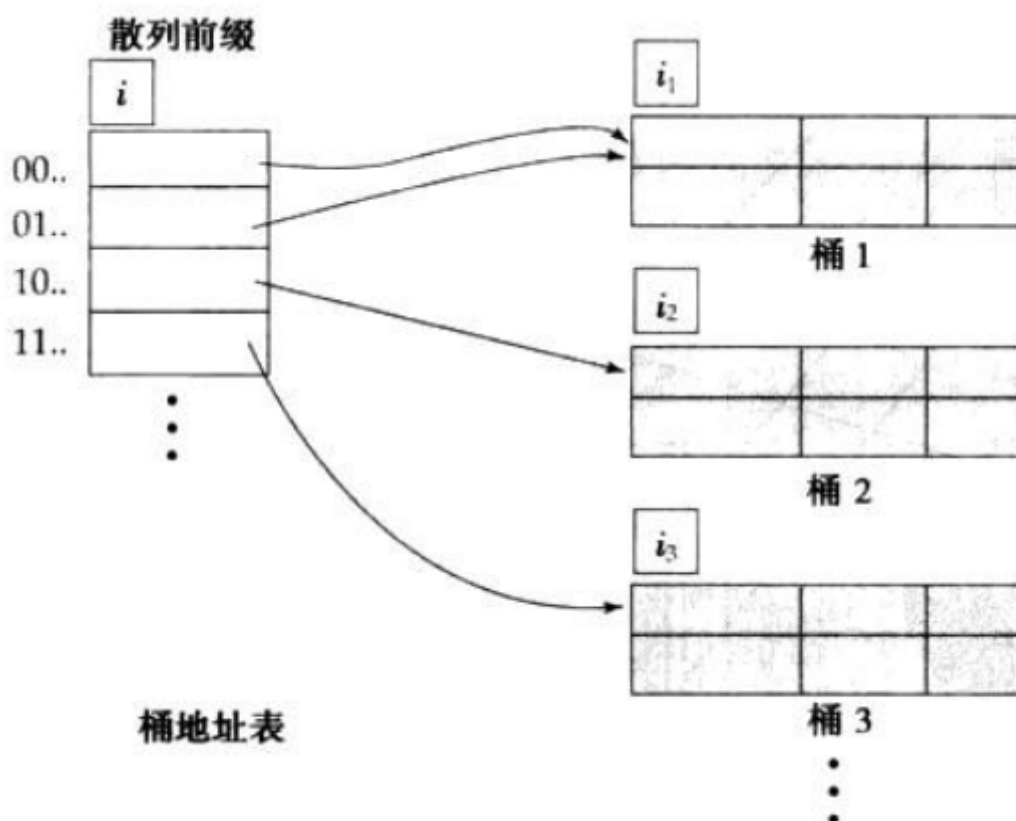
11.7.1 数据结构

当数据库增大或缩小时，可扩充散列可以通过桶的分裂或合并来适应数据库大小的变化。

1. 保持空间的使用效率
2. 每次重组只作用于一个桶，性能开销比较低。

选择具有均匀性和随机性的**散列函数h**，**产生的值范围是b位二进制数**。一个典型的b值是32

我们是在把记录插入文件时按需建桶的。开始时我们不使用散列值的全部b位，任意时刻我们使用的位数 i 满足 $0 \leq i \leq b$ 。这样的 i 个位用作附加的桶地址表中的偏移量。



找出桶地址表中的正确表项需要 i 位，但几个连续的表项可能指向同一个桶。所有这样的表项有一个共同的**散列前缀**，但这个前缀的长度可能小于 i 。因此我们给每一个桶加一个整数值 i_j ，**用来表明这个桶内所有表项共同的散列前缀长度**。

11.7.2 查询和更新

为了确定含有搜索码值 K_i 的记录，系统取得 $h(K)$ 的前 i 个高位，然后为这个位串查找对应的表项，再根据表中指针找到桶的位置。

插入：

要插入一条记录 K ，先定位到某一个桶 j 。如果桶里又剩余空间，直接插入。如果桶已经满了，系统要分裂这个桶并将该桶现有记录和新纪录一起进行重新分配。

为了分裂该桶，系统必须首先根据散列值确定是否需要增加所使用的位数：

- 如果 $i = i_j$ ，那么在桶地址表中只有一个表项指向桶 j 。所以系统需要增加桶地址表的大小，以容纳由于桶 j 分裂而产生的两个桶指针。为了达到这一目的，系统考虑多引入散列值中的一位。系统将 i 的值加 1，从而使桶地址表的大小加倍。这样，原表中每个表项都被两个表项替代，两个表项都包含和原始表项一样的指针。现在桶地址表中有两个表项指向桶 j 。这时，系统分配一个新的桶（桶 z ），并让第二个表项指向此新桶。系统将 i_j 和 i_z 置为 i 。接下来，桶 j 中的各条记录重新散列，根据前 i 位（记住 i 已加 1）来确定该记录是放在桶 j 中还是放到新创建的桶中。

系统现在再次尝试插入该新记录。通常这一尝试会成功。但是，如果桶 j 中原有的所有记录和新插入的记录具有相同的散列值前缀，该桶就必须再次分裂，这是因为桶 j 中的所有记录和新插入的记录被分配到同一个桶中。如果散列函数已经过仔细挑选，一次插入导致两次或两次以上分裂是不太可能的，除非大量的记录具有相同的搜索码。如果桶 j 中所有记录搜索码值相同，那么多少次分裂也不能解决问题。在这种情况下，采用溢出桶来存储记录，就像在静态散列中那样。

- 如果 $i > i_j$ ，那么在桶地址表中有多个表项指向桶 j 。因此，系统不需要扩大桶地址表就能分裂桶 j 。我们发现指向桶 j 的所有表项的索引前缀的最左 i_j 位相同。系统分配一个新桶（桶 z ），将 i_j 和 i_z 置为原 i_j 加 1 后得到的值。接下来系统需要调整桶地址表中原来指向桶 j 的表项。（注意，由于 i_j 有了新的值，并非所有表项的散列前缀的最左 i_j 位都相同。）系统让这些表项的前一半保持原样（指向桶 j ），而使后一半指向新创建的桶（桶 z ）。然后就像上一种情况那样，桶 j 中的各条记录被重新散列，分配到桶 j 或新桶 z 中。

此时，系统重新尝试插入记录。失败的可能性微乎其微，如果失败，则根据情况是 $i = i_j$ 还是 $i > i_j$ 继续做相应的处理。

第一种情况是出现桶分裂前，每个表项都对应一个桶，如果满了，就增加桶地址表，刚分裂完，如果不是被插入桶，桶的 $i_j = i - 1$ 如果是被插入桶， $i_j = i$ 。（见下图的举例说明）

这里存在一个问题是在分裂后重新散列的表项和前 i 位和要插入的还是一样，这样目标桶还是没有位置插入，因此只能放入溢出桶。

第二种情况是刚增加了同地址表后，还有很多满足 $i_j \leq i - 1$ 的桶没有分配新桶，此时可以直接分配新桶。

删除：

要删除一条搜索码值位 K 的记录，系统可以按前面的查找过程找到相应的桶。系统不仅要把搜索码从桶中删除，还要把记录从文件删除。如果桶也变成空的，那么桶也需要删除。

举例说明

搜索码位 dept_name，散列值有 32 位。假设一个桶只能容纳两条记录。假设一开始文件是空的

dept_name	$h(\text{dept_name})$
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

图 11-27 dept_name 的散列函数

插入记录(10101, Srinivasan, Comp. Sci., 65000)。桶地址表包含一个指向桶的指针，系统插入该条记录。接着，我们插入记录(12121, Wu, Finance, 90000)。系统也把这条记录放到对应结构中的一个桶里。

当试图插入下一条记录(15151, Mozart, Music, 40000)时,我们发现桶已经满了。由于 $i = i_0$, 因此需要增加所使用的散列值中的位数。现在我们使用 1 位, 允许有 $2^1 = 2$ 个桶。这一位数的增加使桶地址表的大小必须加倍, 增加到有两个表项。系统分裂桶, 在新桶中放置那些搜索码的散列值以 1 开始的记录, 而将其余的记录留在原来的桶中。图 11-29 给出了该结构在分裂后的状态。

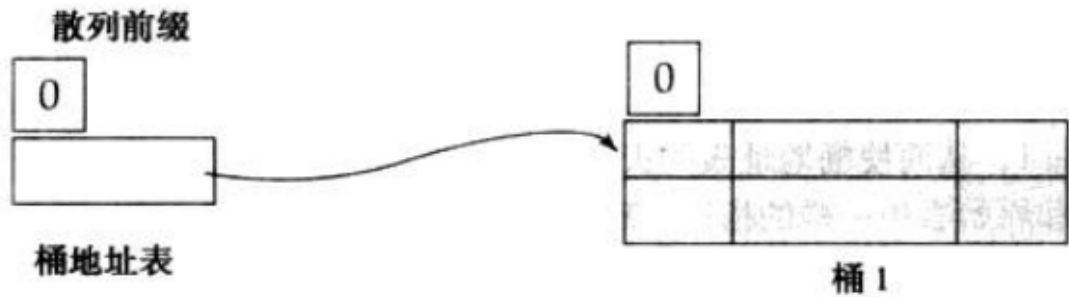


图 11-28 初始的可扩充散列结构

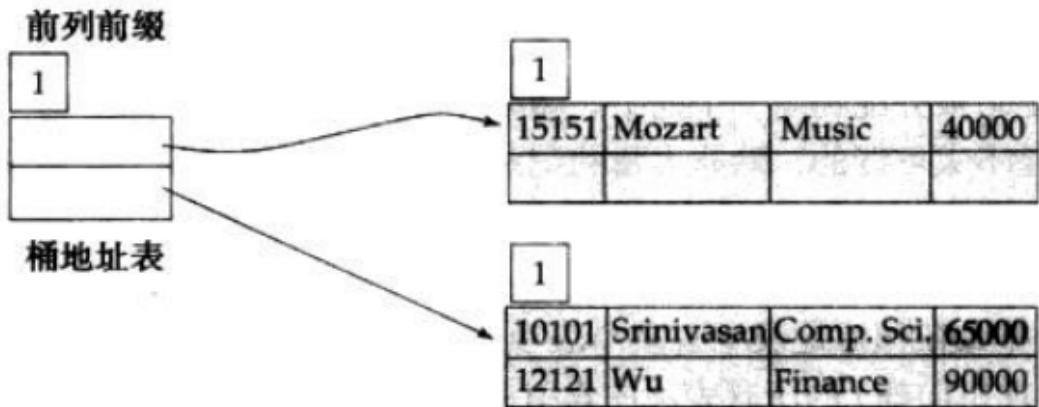


图 11-29 3 次插入操作后的散列结构

接下来, 插入(22222, Einstein, Physics, 95000)。由于 $h(\text{Physics})$ 的第一位是 1, 因此必须将该记录插入桶地址表中表项“1”指向的桶。我们又一次发现桶满了并且 $i = i_1$ 。我们将使用的散列值位数增加到 2。这一位数增加使桶地址表的大小必须加倍, 增加到有 4 个表项, 如图 11-30 所示。由于图 11-29 中散列前缀为 0 的桶并未分裂, 因此桶地址表中 00 和 01 表项都指向该桶。

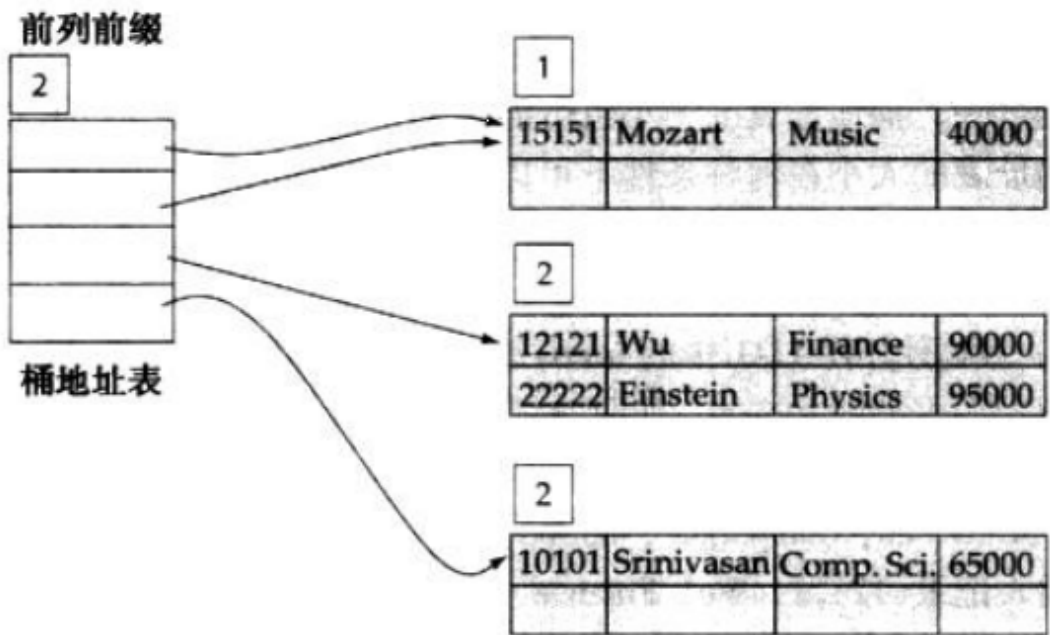


图 11-30 4 次插入操作后的散列结构

对于图 11-29 中散列前缀为 1 的桶(正在分裂的桶)中的每一条记录,系统检查其散列值中的前两位,决定在新结构的哪一个桶中存放它。

接下来,我们插入(32343, El Said, History, 60000),它进入 Comp. Sci. 所在的同一个桶。然后又插入(33456, Gold, Physics, 87000),这导致了桶溢出,引起位数增加和桶地址表的大小加倍(如图 11-31 所示)。

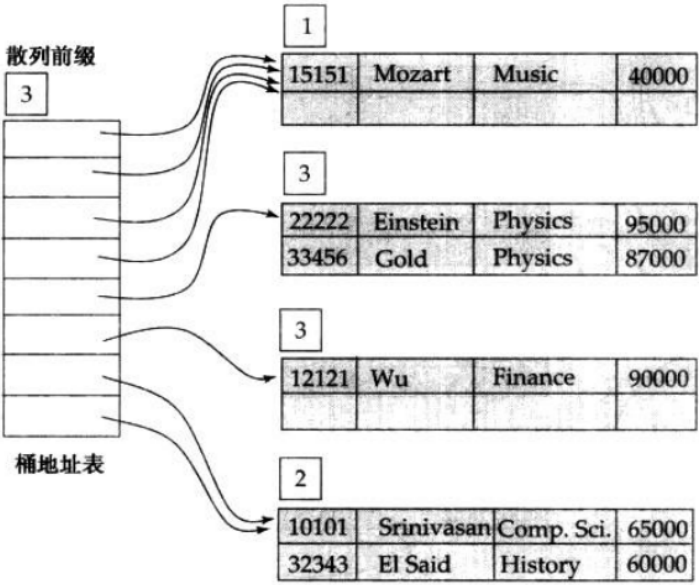


图 11-31 6 次插入操作后的散列结构

(45565, Katz, Comp. Sci., 75000)的插入引起另一次溢出。但是,这次溢出不必用增加位数来解决,因为该桶有两个指向它的指针(如图 11-32 所示)。

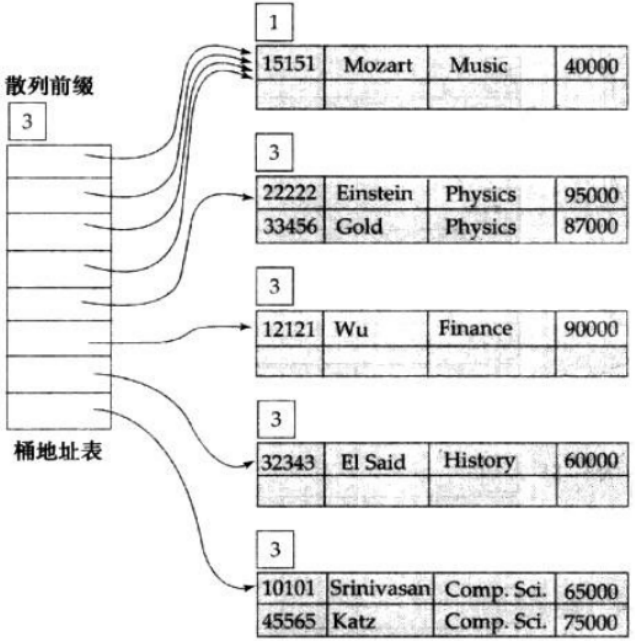


图 11-32 7 次插入操作后的散列结构

接下来插入“Califieri”、“Singh”、“Crick”的记录,它们不会带来桶溢出。但是,第三条 Comp. Sci. 记录(83821, Brandt, Comp. Sci., 92000)的插入会导致另外的溢出。这种溢出不能通过增加位数来解决,因为有三条记录具有相同的散列值。因此系统使用一个溢出桶,如 11-33 所示。我们继续按这种方法进行,直到插入图 11-1 中的所有 instructor 记录为止。产生的结构如图 11-34 所示。

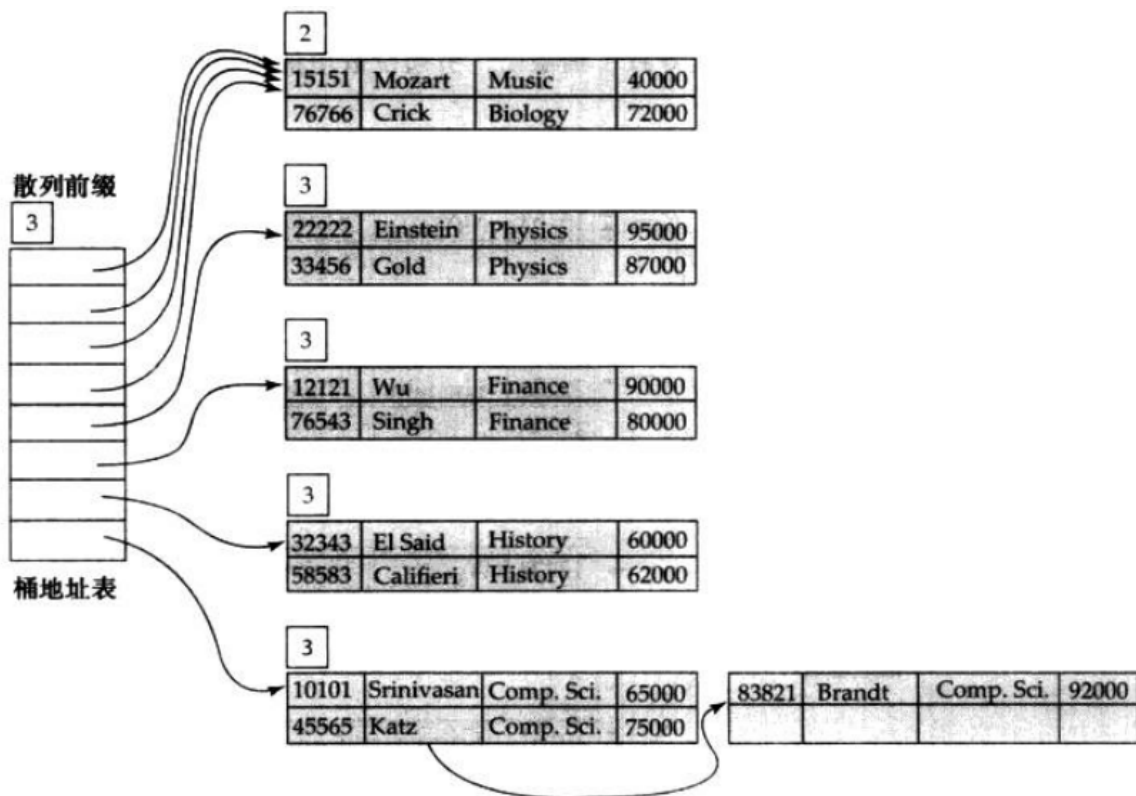


图 11-33 11 次插入操作后的散列结构

11.7.3 静态散列和动态散列的比较

动态散列优点：

- 可扩充散列的优点是其性能不随文件增长而降低
- 空间开销少，空间分配是动态。尽管桶地址表会带来额外的开销

动态散列缺点：

- 要先取桶地址表索引，有一点点额外的时间和空间开销

11.8 顺序索引和散列比较

大多数数据库支持B+树索引，并且有可能还额外支持某种形式的散列索引。

选择何种索引方式，数据库的设计者和实现者需要考虑这些问题：

1. 索引或散列组织的周期性重组代价是否可以接受
2. 插入和删除的相对频率如何
3. 是否愿意增加最坏情况下的访问时间为代价优化平均访问时间
4. 用户可能提供哪些类型的查询

如果查询大多为：

```
select A1, A2, ..., An
from r
where Ai = c;
```

对于这种查询，**散列**的方案更可取。顺序查找所需平均时间于关系r中的记录个数的对数成正比，而散列结构中平均查找时间是一个与数据库大小无关的常数。非散列索引的唯一优势就是，最坏情况下查找时间和记录个数的对数成正比，而散列的最坏情况（散列函数映射后聚集在一起）和关系r中的记录数成正比，不过这几乎不会发生。

如果查询大多为：

```
select  $A_1, A_2, \dots, A_n$ 
from  $r$ 
where  $A_1 \leq c_2$  and  $A_i \geq c_1$ 
```

考虑顺序索引的方法，一旦找到值 c_1 的桶，就顺着索引的指针链读取下一个桶，如此下去可以到达 C_2 。

如果用散列结构就不能这样做了，因为散列函数将值随机分散到各桶中去，我们因此必须要读取所有的桶。

通常设计者会使用顺序索引，除非他预先知道将来不会频繁使用范围查询。散列组织对于在查询执行过程中创建的临时文件来说特别有用，如果需要基于码值查找并且不执行范围查询。