第 6 章 存储技术与数据库物理设计

数据库物理结构是影响数据库系统功能和性能的重要因素，DBMS所采用的数据操作算法、查询优化处理方法和事务处理算法与数据库物理存储结构密切相关。本章首先介绍各种存储介质特性，然后介绍数据库文件的组织和结构形式，以及支持文件快速访问的常用索引技术，包括有序索引、B^+-树、散列文件等。最后介绍DBAS物理设计阶段的物理数据库设计内容和步骤。

6.1 物理存储介质

6.1.1 存储介质层次

如图 6.1 所示，计算机系统中的存储介质（Storage）是按照层次组织的，根据访问速度、容量、成本和可靠性可分成以下 6 类。

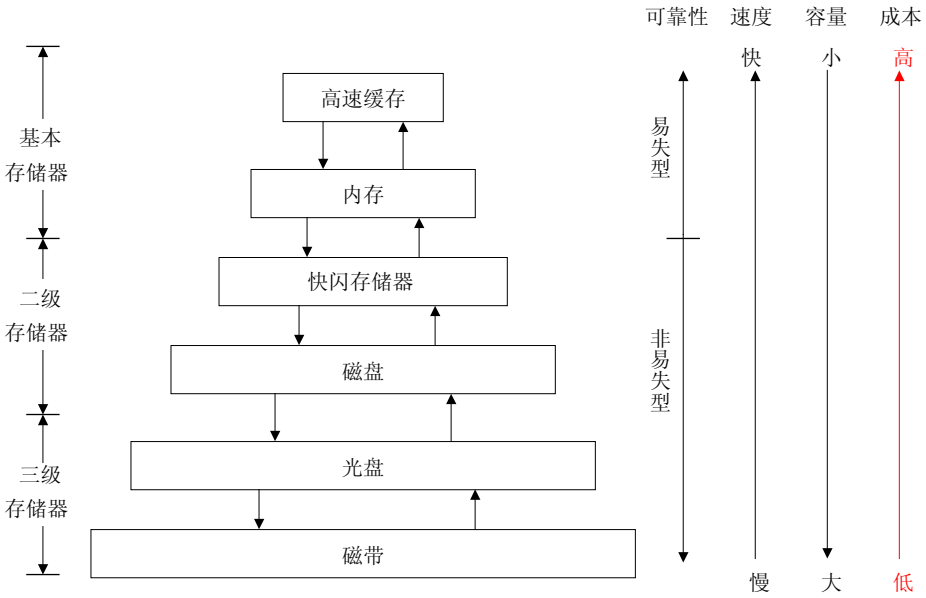


图 6.1 存储介质层次

1. 高速缓冲存储器（Cache）

高速缓冲存储器简称为高速缓存，也就是一般所说的 Cache。利用 cache 机制，可以提高 CPU 存取指令和数据的速度。

2. 内存储器（Main Memory）

内存储器简称为内存或内存，CPU 通过执行指令可以直接对内存中的数据进行修改。

3. 快擦写存储器（Flash Memory）

快擦写存储器又称为电可擦可编程只读存储器（即 EEPROM），简称为“快闪存”。快闪存在掉电后仍能保持数据不丢失。在快闪存中读一次数据的时间为 100ns，几乎接近内存速度，但写操作较慢，而且不能直接重写。如果要重写，必须先擦去整组快闪存储器的内容，然后再写数据。快闪存的缺点是只能支持有限次擦写，一般次数在 1-100 万次左右。目前快闪存已用于小型低成本数据库系统中。

4. 磁盘存储器（Magnetic-Disk Storage）

磁盘是目前最流行的外部存储器，能长时间地联机存储数据。磁盘属于直接访问存储器（Direct access memory），支持用户直接读磁盘中某一位置的数据。

5．光存储器（Optical Storage）

目前流行的光存储器是光盘。**光存储器**利用光学原理，通过激光器访问存储在光盘中的数据。光存储器包括光盘只读存储器（CD-ROM）、一写多读光盘（WORM）。WORM 允许写入数据一次，但不能擦除或重写，容量已达 GB 数量级。这种介质广泛用于数据的归档存储。

6. 磁带（Tape Storage）

磁带属于顺序存取存储器，只能从头至尾顺序地访问存储在磁带上的数据。磁带的容量很大，又称为海量存储器，但访问速度慢（以分钟计算），并且使用次数有限。磁带可用于存储数据库备份数据或归档数据。

对上述 6 类存储介质，可以从存储容量、访问/存取速度、成本（指单位存储容量的价格）、等方面来考察。并且从可靠性角度，按照系统掉电或关机后数据是否仍然能够保存在存储器中分为易失型存储器和非易失型存储器。

上述存储介质还可划分为基本存储器或内存储器（Primary storage）、辅助存储器（Secondary storage，也称为联机存储器）、第三级存储器（Tertiary storage，也称为脱机存储器）三个层次。

6.1.2 磁盘存储器

磁盘存储器是一种大容量、可直接存取的外部存储设备。直接存取是指可以随机到达磁盘上的任意位置存取其中的数据。磁盘有软磁盘和硬磁盘（简称为硬盘）之分。下面以图 6.2 所示、带有活动臂的硬盘存储器为例说明磁盘存储器工作原理。

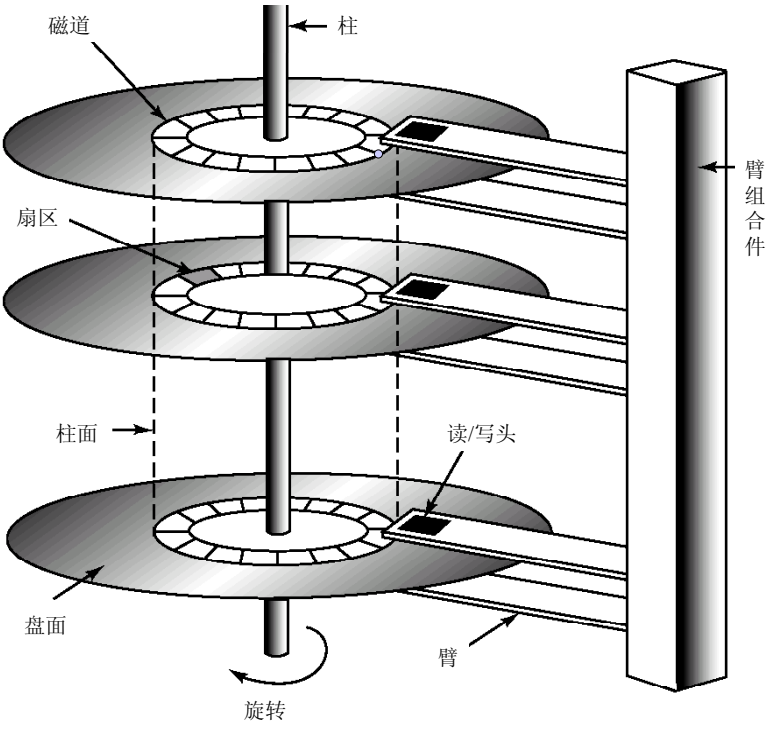


图 6.2 活动臂硬盘存储器

多个磁盘组装在一起，形成一个磁盘组。一个磁盘组可以具有数十个存储数据的磁盘表面。每个磁盘表面由多个磁道组成。数据存储在磁道上。每个磁道又分为扇区（也称为磁盘

块，block）。在磁盘组上，所有磁盘面上具有相同直径的同心圆集合称为一个圆柱。

磁盘存储器由磁盘和驱动器构成。磁头（也称读/写头）和磁臂是驱动器的重要组成部分。磁头负责读写各磁道上的数据，安装在磁臂上。每个盘面上有一个读/写头。读/写头可以随磁臂移动到磁盘面的任何一个磁道上，读写该磁道上的数据。

磁盘存储器的读写单位是磁盘块（或扇区），即一次能且仅能读写一个磁盘块。每个磁盘块可以存储很多字节的信息。内存与磁盘间交换信息以磁盘块为单位。磁盘存储器可以根据磁盘块地址直接读写任何一个磁盘块，磁盘块地址的形式是<柱号，盘面号，扇区号>。

磁盘存储器读写磁盘上某个磁盘块数据的过程如下：磁盘驱动器首先根据磁盘块地址驱动磁臂，将磁头定位到磁盘块地址指定的磁道，然后等待指定的磁盘块旋转到磁头下边，最后在内存和磁盘块之间传输数据。磁头定位到指定磁道的时间称为**寻找时间**。等待指定磁盘块旋转到磁头下面的时间称为**旋转延迟**。在内存和磁盘块之间传输数据的时间称为**传输时间**。读写一个扇区的总时间是寻找时间、旋转延迟和传输时间的总和。

磁盘定位和磁盘旋转是机械运动。所以，寻找时间和旋转延迟很大。为了节省读写时间，应该尽量一起读写多个邻接的磁盘块。这样，只花费一个磁盘块的寻找时间和旋转延迟就可以读写多个磁盘块。

一次磁盘读写需要毫秒级的时间。寻找时间和旋转延迟一般在 15 到 60 毫秒之间，数据传输时间大约在 1 到 2 毫秒时间之内。与 CPU 在内存处理数据的时间相比，磁盘读写时间是相当高的。因此，磁盘读写是数据库应用的瓶颈。数据库的物理存储结构、数据库操作算法和查询优化的研究都将最小化磁盘读写次数作为重要目标之一。

进行磁盘读写时，内存中必须具有与磁盘块容量相匹配的缓冲区，用于存储磁盘块。磁盘控制器可以将磁盘块读入内存缓冲区，再由 CPU 执行应用程序读取缓冲区中磁盘块进行相关处理；也可以由应用程序将需要输出的数据写入缓冲区中，再由磁盘控制器将缓冲区数据写入磁盘。可以一次读写一个磁盘块的数据，也可以一次读写多个邻接磁盘块中的数据。

当需要在内存和磁盘间传输多个磁盘块时，可以在内存中设置多个数据缓冲区。磁盘驱动器与 CPU 并行工作，当磁盘驱动器与一个缓冲区交换数据时，CPU 同时处理来自另一个缓冲区中的数据，以提高内存与磁盘间的数据传输速度。

6.1.3 独立磁盘冗余阵列

独立磁盘冗余阵列 （Redundant Array of Independent Disks，RAID）是一种多磁盘组织标准，是大型数据库系统经常采用的数据存储技术。RAID 利用一台磁盘阵列控制器统一管理和控制一组磁盘驱动器，组成一个速度快、可靠性高、性能价格比好的大容量外存储（磁盘）子系统。RAID 巧妙地解决了 CPU 速度快与磁盘设备速度慢之间的速度不匹配问题，其策略是：用一组较小容量的、独立的、可并行工作的磁盘驱动器组成阵列来代替单一的大容量磁盘，再结合冗余技术，数据在各个磁盘上能用多种方式组织和分布存储，多个独立的磁盘访问请求能被并行处理，数据分布的单个 I/O 请求也能并行地从多个磁盘驱动器同时存取数据，从而改进了 I/O 性能和系统可靠性。

RAID 规范由 RAID0 至 RAID7 共 8 级组成，但它们之间并不隐含层次关系，而是表明了不同的设计结构，具有的共同特点是（1） RAID 是一组物理磁盘驱动器，可以被操作系统作为单一逻辑磁盘驱动器。（2）数据被分布存储在多个物理驱动器阵列上。（3）使用冗余磁盘保存奇偶校验信息，以保证当磁盘出现错误时数据的恢复。多个磁头和驱动机构同时操作能达到较高 I/O 数据传输速率，但使用多台硬设备也增加了故障的概率，为了补偿可能造成的可靠性下降，RAID 利用存储的奇偶校验信息来恢复由于磁盘故障丢失的数据。

RAID0 将大块数据分割成数据条块（strip），交替间隔地分布存储在横跨阵列中的所有磁盘上。逻辑上连续的数据条块，在物理上可被循环地依次存储在横向相邻的磁盘驱动器上，

并通过阵列管理软件进行逻辑地址空间到物理地址空间的映射，支持按条块并行存取位于不同磁盘上的数据。但 RAID0 不支持第三个特性，未引入冗余校验，导致数据可靠性差，适用于数据存取速度要求高，但非要害数据的这类应用。

RAID1 与 RAID2 至 RAID5 的主要差别在于实现冗余校验的方法不同。RAID1 采用镜像技术，适用于做系统驱动器，存放关键的系统文件。RAID2 和 RAID3 采用了并行存取技术，分别引入海明（Hamming code）校验码和位交织（bit interleaved）奇偶校验码改进可靠性，适用于大数据量 I/O 请求，如图像处理应用。RAID4 和 RAID5 采用了独立存取技术，分别引入块交织（block interleaved）奇偶校验码和块交织分布（block interleaved distributed）奇偶校验码改进可靠性，适用于 I/O 请求频繁的事务处理。RAID6 和 RAID7 是增强型 RAID，设置了快速专用异步校验磁盘，具有独立异步数据访问通道。

6.2 文件组织

6.2.1 数据库物理组织

数据库逻辑设计为存放在数据库中的应用数据设计了逻辑组织结构。这种逻辑结构建立在某种具体数据结构模型（如关系模型、层次模型、网状模型、面向对象模型等）基础上，与数据库的具体物理实现无关，它定义了用户所看到的应用数据在数据库中的组织方式，随之也规定了用户访问数据库的方式。

以关系数据库系统和磁盘外设存储设备为例。数据库系统设计者根据关系模型规定的数据组织形式，将应用数据的逻辑结构描述为关系模式。因此，从 DB 用户角度，数据库是由一系列关系表组成的，每个关系表是关系元组的集合，关系表的结构由上述关系模式来定义。用户利用关系 DBMS 中的数据操纵语言 DML 所提供的数据访问操作来存取数据库中的数据，这些操作的类型、功能符合关系模型中的关系操作规范。

数据库中的数据是以文件形式存储在外设存储介质（如磁盘）上的。从操作系统角度，文件在逻辑上被组织成纪录的序列，也就是说，每个 DB 文件可以看作是逻辑纪录的集合。

一个文件在磁盘上占有一定的物理存储空间，文件中的每个逻辑纪录被映射存储到某个特定的磁盘块上。一个文件在物理上可以看作是由存放文件记录的一系列磁盘块组成的，称之为物理文件。

文件的逻辑纪录与磁盘块间的映射关系是由操作系统来管理的。当需要对一个文件逻辑纪录查询、插入、删除、修改时，首先需要根据这种映射关系找到该逻辑纪录所在的磁盘块，然后再进行相应的操作。

从数据库物理组织角度，需要解决如下几个问题：

（1）文件的组织。如何将数据库映射为操作系统文件，也就是如何将关系数据库中的关系表映射为数据库文件，以及将关系表中的元组映射到文件的逻辑纪录，如何管理这些数据库文件；数据库文件中逻辑纪录采用何种格式。

（2）文件的结构，也称为文件中记录的组织。如何在外设磁盘上安排、存放数据库文件的逻辑纪录，也就是如何将 DB 文件的逻辑纪录映射到物理文件中的磁盘块。

（3）文件的存取。对具有某种结构的 DB 文件，如何去查找、插入、删除和修改其中的逻辑纪录。文件的存取与文件的结构密切相关，每一种文件结构都有与之对应的文件存取方法。

（4）索引技术。当 DB 中的数据量很大、DB 文件中的文件纪录很多时，也就是关系数据库中关系表的数目和关系表中的元组数目非常多时，如何提高 DB 文件的存取速度。

下面讨论第 1 个问题，6.3 节考虑第 2 和第 3 个问题，6.4 、6.5 和 6.6 节研究第 4 个问题。

6.2.2 文件组织

1．数据库与文件的对应关系

在外存中，数据库以文件形式组织，文件由逻辑记录组成，纪录由多个域组成。一个关系数据库包括一张或多张关系表，关系表与文件的对应关系可以采用如下两种方式：

（1）每张关系表单独用一个文件来存储，DBMS 可以利用操作系统的文件管理功能来管理数据库文件。一些嵌入式 DBMS 和早期小型关系 DBMS（如 Debase 和 Foxbase）经常采用这种方法。

（2）SQL Server、DB2、Oracle、Sybase 等现代大中型 DBMS 在文件管理方面不直接依赖于 OS，而是由 OS 分配给 DBMS 一个大的 OS 文件（一块大的磁盘存储空间），DB 中所有关系表都存储在这个文件中。对文件的管理由 DBMS 直接负责。

SQL Server 2000 数据库系统支持下列三种类型的数据库文件：

（1）主数据文件。每个数据库有一个主数据文件，除用于存储关系表中的应用数据外，还用于跟踪数据库中所有其它文件。

（2）次数据库文件。一个数据库可以有零个或多个次数据文件，用于存放主数据文件中存放不下的数据或索引数据。

（3）日志文件。每个数据库至少有一个日志文件，它存储了可用于数据库恢复的事务日志信息。

关系表在逻辑上由一系列元组组成，元组由多个属性组成。每个元组可以用数据库文件中的一个逻辑记录来存储，纪录包括多个域。元组的每个属性对应于文件记录中的一个域。在 SQL Server2000 中，数据库文件中的逻辑纪录称为数据行。

在后面的叙述中，文件记录的域也称为文件记录的属性。文件记录的属性、主键、非主属性、候选健分别指文件所存储的关系模式的属性、主键、非主属性、候选健。

查找键（Search key）是文件记录中某个属性或属性的集合。用户可以根据文件纪录在查找键上的取值对记录进行查询。

例如，在关系模式“商品 Goods (GoodsID, GoodsClassID, GoodsName, ProductionDate, TotalStorage, Description)”对应的数据库文件中，可以将主属性 GoodsID 作为查找键，也可以由非主属性 GoodsName 和 ProductionDate 共同组成查找键{GoodsName，ProductionDate}。

2．文件记录格式

数据库文件通常采用两种逻辑纪录格式：定长记录格式和变长记录格式。

（1）定长记录。数据库文件中的所有记录具有相同、固定的长度。

例如，对于关系模式“商品 Goods (GoodsID, GoodsClassID, GoodsName, ProductionDate, TotalStorage, Description)”，可以设计一个文件 Goods_table，其记录格式如下：

```
type    GOODS = record
                    GoodsID :          char(8)
                    GoodsClassID :     char(8)
                    GoodsName :        char(50)
                    ProductionDate :   Datetime
                    TotalStorage :     Money
                    Description :      char(50)
                end
```

纪录中的每个域对应于关系模式中的一个属性，并且域的数据类型已知，每个域所占的字

节也已知。例如，如果 1 个字符用 1 个字节来表示，则 GoodsID 占用 8 个字节。由此，可以算出每个纪录所占的字节总长度。

（2）变长记录

在数据库系统中，有时需要文件中的记录是变长格式。例如，一个文件存储了多种不同记录类型的记录；文件记录本身是变长的；文件记录中某个字段可以重复出现等。变长记录可采用字节串表示形式，也可采用定长表示形式。

例如，上述定长记录的文件 Goods_table 也可以设计成如下变长记录格式：

```
type GOODS-list = record
                    GoodsClassID:    char (8)
                    Goods-inf : array [1.. ∞ ] of
                                  record
                                      GoodsID : char (8)
                                      GoodsName : char (50)
                                      ProductionDate : Datetime
                                      TotalStorage : Money
                                      Description : char (50)
                                  end
                  end
```

此处定义了（GoodsID, GoodsName, ProductionDate, TotalStorage, Description）作为数组 Goods-inf 的组成元素，元素的个数视实际情况而定。

实现变长记录的技术有多种，包括分槽式页结构、指针方法和保留空间等方法。限于篇幅，此处不再赘述。

6.3 文件结构与存取

一个数据库文件往往包含许多逻辑记录，如何在外设磁盘上安排、存放这些纪录，也就是如何将文件纪录映射到外设磁盘块，属于文件结构问题。

当文件记录存放在外设磁盘上时，一个纪录中的内容可以全部存放在一个磁盘块中，这属于非跨块存放。如果一个记录中的数据很多，也可以采用跨块存放方式，分布存放在多个磁盘块中。

文件结构主要有五种形式：堆文件（Heap file）、顺序文件（Sequential file）、聚集文件（Clustering file）、索引文件（Indexing 文件）和散列文件（Hashing file）。

对于某种结构的文件如何去定位查找、插入和删除其中的纪录，属于文件（记录）的存取方法问题。文件的存取与文件结构密切相关，不同结构的文件的存取方法是不一样的，并且存取效率也差别很大。

6.3.1 堆文件

堆文件也称为无序（记录）文件。在堆文件中，记录随机地存储在文件物理空间中，新插入的记录存储在文件的末尾。

堆文件常常用来存储那些将来使用、但目前尚不清楚如何使用的文件记录。为了实现对文件记录的有效存取，堆文件经常与附加的存取路径一起使用，如辅助索引。

堆文件既可用于定长记录文件，也可用于变长记录文件。记录存储方法可以采用跨块记录存储方法，也可以采用非跨块记录存储方法。

在文件中查找一个满足给定条件的记录时，首先从文件的第一个记录开始搜索，直到发现满足条件的记录为止。如果满足条件的记录不止一个，我们需要搜索整个文件。设文件占B个磁盘块，满足条件的记录只有一个，一个查找操作平均需要搜索 (B+1)/2 个磁盘块。因此，堆文件的查找操作效率比较低。

堆文件的插入操作十分简单。在堆文件的头存储它的最后一个磁盘块的地址。当插入一个新记录时，首先读文件头，找到最末磁盘块地址，将最末磁盘块读入内存缓冲区；然后，在缓冲区内将新插入的记录写入最末磁盘块的末尾；最后，将缓冲区中修改过的最末磁盘块写回磁盘文件。

堆文件的删除操作比较复杂，可采用三种方法来实现。

（1）第一种方法是首先找到被删除记录所在的磁盘块，然后将该磁盘块读到内存缓冲区。在缓冲区中删除记录，将缓冲区中修改后的磁盘块内容写回磁盘文件。这种方法将使文件中出现空闲的存储空间，需要周期地整理存储空间，避免存储空间的浪费。

（2）第二种方法是在每个记录的存储空间增加一个删除标志位。当删除一个记录时，将删除标志位置1。查找记录时跳过删除位置1的记录。这种方法也需要周期地整理存储空间。

（3）如果文件是定长记录文件，当删除一个记录时，将文件末尾记录移动到被删除记录的位置，从而避免重新整理存储空间。

6.3.2 顺序文件

顺序文件按照文件记录在查找键上的取值的大小顺序排列各个纪录，记录按照查找键值的升序或降序顺序地存储在文件中。

顺序文件的每个记录中有一个指针字段，根据查找键值的大小用指针将各个记录按序链接起来。可以很方便地按查找键值的大小，顺序读出文件中所有记录。

例如，图 6.3 给出了关系表 Goods_table(GoodsID, GoodsClassID, GoodsName, ProductionDate, TotalStorage, Description) 对应的顺序文件，记录按 ProductionDate 值升序排列。域/属性 ProductionDate 是文件的查找键。

文件初始建立时，应尽可能使记录的物理存储顺序与查找键值的顺序一致，以便访问数据时减少磁盘块访问的次数。

根据一定的查询条件对顺序文件进行查询时，如果查询条件定义在查找键上，可以使用二分查找或插值查找技术快速找到满足条件的记录。二分查找技术的基本思想是：每次从文件中位于居中位置的记录开始检查，看是否为欲找的目标记录，若不是则丢掉不包含该记录的文件的一半，从剩下的一半文件记录中重新开始检查，如此反复，直至找到目标记录或证明它不存在。

如果查询条件定义在非查找键上，对顺序文件的访问必须从文件头部开始依次扫描各个文件记录，直至找到目标纪录。访问方式与堆文件访问方式相同，顺序文件没有提供任何优越性。

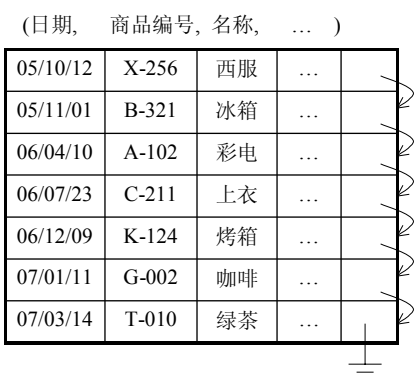


图 6.3 顺序文件

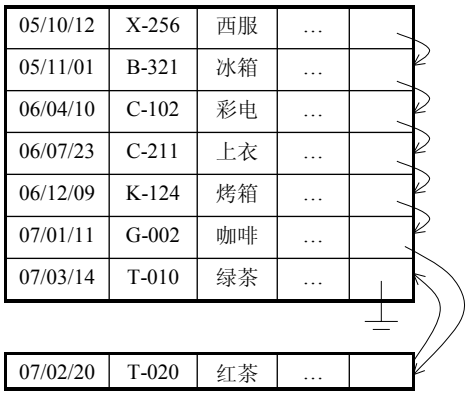


图 6.4 插入一个记录后的顺序文件

顺序文件上的插入和删除操作比较复杂，需要保持文件中记录的顺序，耗费的时间较大。

顺序文件插入操作包括定位和插入两步：

（1）定位。在指针链中，找到插入的位置，即插入记录应插在哪个记录的前面（这是查找键值的顺序，而不一定是物理顺序）。

（2）插入。在找到记录的物理块内，如果自由空间中有空闲记录，那么在该位置插入新记录；如果无空闲记录，那么就只能插入到溢出块中；重新调整纪录指针链关系，保证纪录顺序的正确性。

例如，在图 6.3 中的商品文件中插入一个新记录，可得到图 6.4 所示的顺序文件。

对顺序文件记录的删除操作可以直接通过修改指针实现。被删除的记录链接成一个自由空间，以便插入新记录时使用。例如，在图 6.3 中，删除第 3 个纪录“彩电”后，可将第 2 个记录“冰箱”的指针指向原来的第 4 个记录“上衣”。

在顺序文件初始建立时，可以保持纪录的物理顺序与查找键值顺序一致，但是，经过多次插入或删除操作后，很难继续维持这种一致的状态。此时需要对文件重新组织一次，使其物理顺序和查找键值的顺序一致，以提高查找速度。

6.3.3 聚集文件

1. 多表聚集

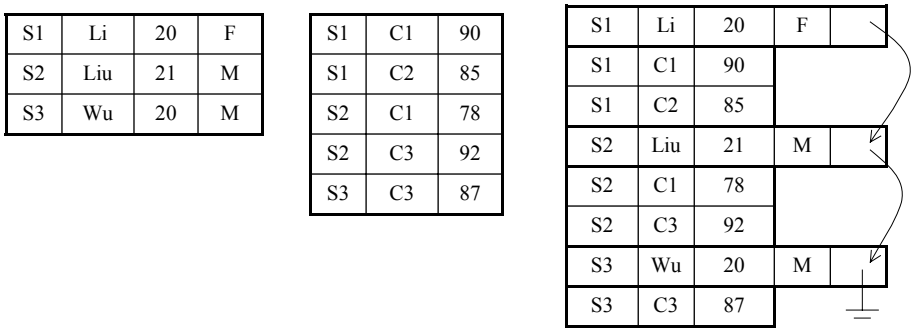
在关系数据库系统中，关系表中的数据存放在数据库文件中。如果一个文件只存放一个关系表中的数据，则该关系表对应的关系模式决定了文件的记录类型。此时，该文件的记录类型唯一，称之为单记录类型文件。单记录类型文件经常用在小型数据库系统和嵌入式数据库系统中。

聚集文件是一种具有多种记录类型文件，它存储了来自多个关系表的记录数据，每个关系表对应文件中的一种记录类型。

当数据库系统中的数据量非常大时，对数据库的查询需要多次访问外设磁盘上的数据库文件，严重影响到数据库系统查询响应时间等性能指标。特别是一些复杂查询，经常涉及到 DBMS 内部复杂的多表连接操作，需要同时访问多张关系表中的关联数据，引起大量的磁盘访问操作。因此，如何降低复杂多表操作时的磁盘访问次数是提高数据库系统复杂多表查询速度的关键，取决于如何有效地在外设磁盘上合理安排各个关系表中的记录数据。

在聚集文件中，经常将不同关系表中有关联关系的记录存储在同一磁盘块内，从而减少数据库多表查询操作时的磁盘块访问次数，提高系统 I/O 速度和查找处理速度。

数据库系统设计者可以根据用户在实际应用中对数据库的访问情况来合理地建立聚集文件。例如，可以将频繁发生的多表查询所涉及到的多个关系表存储在一个聚集文件中。对聚集文件的管理则由 DBMS 负责。



(a) 关系S　　(b) 关系SC　　(c) 聚集文件

图 6.5 聚集文件例子

例如，一个教学信息管理系统的数据库中有 2 张关系表：学生关系表 S(Student# , Sname, Age, Sex)和学生选课关系表 SC(Student# , Course#, Grade)。下述 SQL 语句查询学生的学号、姓名、选修的课程及其成绩：

```
SELECT S.Student# , Sname, Course#, Grade
FROM    S, SC
WHERE S.student# = SC.Student#
```

如图 6.5 所示。如果将关系表 S 和 SC 上分别组织成 2 个文件，此查询需要在关系表 S 和 SC 上做连接操作，分别读取 S 表中的元组和 SC 表中的元组，并比较元组在公共属性 student 上的取值是否相同。由于需要分别读取 2 个文件，当关系 S 和 SC 数据量很大时，需要访问许多磁盘块，连接操作速度是很慢的。

如果采用聚集文件，将关系 S 和 SC 的数据放在一个文件内，并且尽可能将每个学生的基本信息和他的成绩放在聚集文件中的相邻位置上，则上述查询操作可以在一个文件中读取学生的基本信息和成绩信息。并且对于同一个学生而言，存放学生信息的文件记录和存放学生成绩的文件记录安排在文件相邻位置上，这 2 个纪录有可能存放在同一个磁盘块或存放在位置相近的 2 个磁盘块中，因此可以用较少的磁盘访问操作将学生基本信息和成绩信息读到内存中。如果学号为 S1 的学生的基本信息和成绩信息放在同一磁盘块中，则聚集文件中的第 1、第 2、第 3 个记录可以在一次 I/O 操作中被同时读入内存缓冲区进行连接操作。

为了进一步提高聚集文件的查询速度，可以在文件中建立以查询学生信息为主的链表。

2. 单表聚集

单表聚集是为了提高某个属性（或属性组）的查询速度，将在这个或这些属性（称为聚集键）上有相同值的元组集中存放在一个物理块中，以提高按聚集键进行查询的效率。许多关系型 DBMS 都提供了此功能。

例如，假设对商品关系表 Goods_table(GoodsID, GoodsClassID, GoodsName, ProductionYear)在生产年份上建有索引, 现在要查询 2005 年生产的所有商品的全部信息。假设 2005 年生产的商品有 100 种，在极端情况下，这 100 种商品对应的元组分布在 100 个不同的物理块上，每访问一个物理块需要执行一次 I/O 操作。因此该查询即使不考虑访问索引的 I/O 次数，也要执行 100 次 I/O 操作。如果将生产年份相同的元组集中存放，则每读一个物理块可得到多个满足查询条件的元组，从而显著减少磁盘访问次数。

聚集以后，聚集键相同的元组集中在一起了，因而聚集键值不必在每个元组中重复存储，只要在一组中存一次就行了，因此可以节省一些存储空间。

不管是多表聚集还是单表聚集，聚集文件组织只能提高某些特定应用的性能，而且建立与维护聚集的开销是相当大的。对已有关系建立聚集，将导致关系中元组移动其物理存储位置，导致此关系上原有的索引无效，必须重建。当一个元组的聚集键改变时，该元组的存储

位置也要做相应移动。因此只有在用户应用满足下列条件时才考虑建立聚集，否则很可能会适得其反：

（1）通过聚集键进行访问或连接是该关系的主要应用，与聚集键无关的其它访问很少或者是次要的。尤其当 SQL 语句中包含有与聚集键有关的 ORDER BY、GROUPBY、UNION、DISTINCT 等子句或短语时，使用聚集特别有利，可以省去对结果集的排序操作。

（2）对应每个聚集键值的平均元组数既不太少，也不太多。太少了，聚集的效益不明显，甚至浪费块的空间；太多了，就要采用多个链接块，同样不利于提高性能。

（3）聚集键值相对稳定，以减少修改聚集键值所引起的维护开销。

6.3.4 索引文件

索引文件是一种利用索引技术（Indexing）支持快速文件访问的文件组织和存取方法。对于存储在外设磁盘上的数据文件，定义用于文件记录访问的查找键，建立查找键与文件记录物理存储地址间的映射关系，用索引项记载这种关系并组成索引文件。利用索引文件可以快速访问数据文件记录。

索引技术将在 6.4 节中介绍。索引文件组织的典型例子是B^+-树文件组织，具体参见 6.4.3 节。

6.3.5 散列文件

散列文件是一种利用散列函数（Hash 函数，也称为哈希函数）支持快速文件访问的文件组织和存取方法。用散列方法存储一个文件时，首先需要指定文件记录的一个（或一组）域作为 HASH 域，然后定义一个 HASH 域上的函数，称为**散列函数**。

假设文件的 HASH 域是 A，H 是定义在 A 的值域上的散列函数。对文件中的记录 r，r 在域 A 上的取值为 a，则记录 r 的存储地址由散列函数值 H (a)来确定，这个地址是 r 所在的磁盘块的物理地址。访问散列文件中某个记录时，首先使用散列函数找到该记录所在的磁盘块，然后将该磁盘块读入内存缓冲区，在缓冲区中找到记录后进行相应处理。

散列文件将在 6.5 节中介绍。

6.4 索引技术

6.4.1 基本概念

1. 索引技术

用户对数据库的查询访问请求经数据库管理系统作相应处理后，会转换为对数据库文件的访问。当数据库中数据量很大、数据库文件包含的记录数非常多时，对数据库文件的访问耗时费力。为提高数据库访问的查询响应时间，可以采用**索引技术（Indexing）。**

索引技术的思想与科技图书中术语索引的思想相似。一本科技图书中的术语索引可以帮助读者很快找到该术语在书中的具体解释信息。图书中术语索引表的每项包括：术语和该术语所在页号。当查阅一个术语时，先查阅术语索引，找到该术语所在的页号，然后到相应的页查阅术语的详细介绍。

索引技术是一种快速文件访问技术，它将一个文件的每个记录在某个或某些域（或称为属性）上的取值与该纪录的物理地址直接联系起来，提供了一种根据纪录域的取值快速访问

文件纪录的机制。索引技术在功能上类似于图书中的术语索引，纪录域的取值相当于图书术语索引表中的术语，记录的物理地址（一般为该纪录所在的磁盘块块号）对应于术语所在图书页号。

索引技术的关键是建立纪录域取值到记录的物理地址间的映射关系，这种映射关系称为索引（Index）。中文术语“索引”既可以指广义上的 Indexing（索引技术），也可以指狭义上的 Index（索引）本身。

2. 索引技术分类.

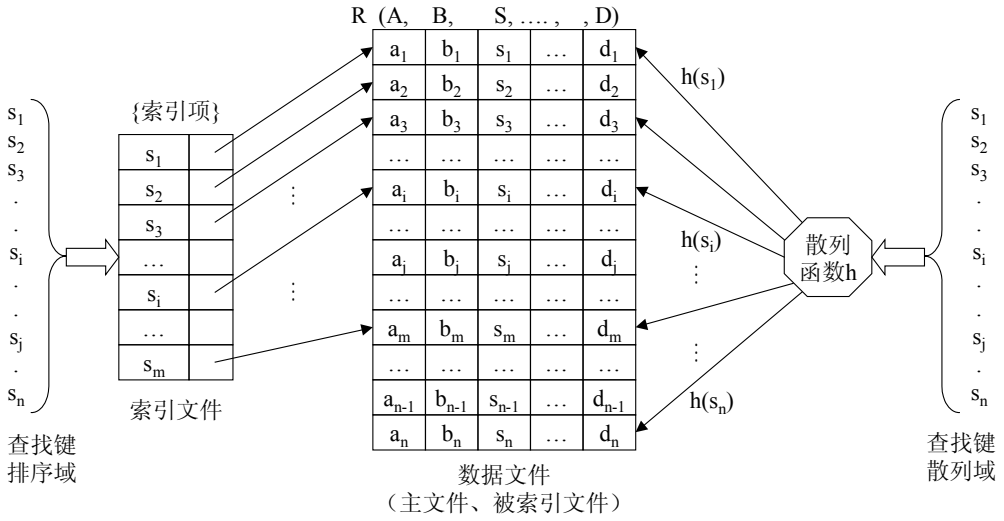


图 6.6 索引技术

参见图 6.6，根据索引 Index 的实现方式，索引技术可以分成 2 类。

（1）**有序索引**（Ordered index）技术，也称为索引文件机制。

有序索引技术利用索引文件（Index file）实现纪录域取值到记录物理地址间的映射关系。这里的记录域就是查找键。索引文件由索引纪录（Index record）组成，每个纪录中记载着一个索引项（Index entry），索引项纪录了某个特定的查找键值和具有该值的数据文件记录的物理地址。查找键也称为排序域。

当需要访问数据文件中某个数据纪录时，先根据纪录域取值查阅索引文件，找到对应的索引项；从索引项中找出数据记录在数据文件中的物理地址．然后根据这个地址存取数据记录。

在图 6.6 中，数据文件中存储了关系模式为R (A, B, S, …, D)的关系表的元组数据。其中，A为R的主键，因此也是数据文件的主键。B为R的非主属性，并且S作为查找键。当用户需要访问在查找键上的取值为s_i的特定数据纪录时，需要到索引文件中寻找s_i对应的索引项，再根据索引项中的地址指针（记录所在的磁盘块号）去访问纪录。

查找键可以是数据文件的主键或主属性，也可以是非主属性。如果是非主属性，则有可能存在 2 个数据文件记录，这 2 个记录虽然主键值不同，但在查找键上的取值完全相同。例如在图 6.6 中，对数据文件的第i个和第j个元组，$a_i \neq a_j$，但$s_i = s_j$，也就是说按照查找键通过索引去查找数据文件，有可能得到多个数据纪录。

对索引文件而言，它的索引项中出现的所有不同查找键值，有可能只是数据文件的所有记录中出现的所有不同查找键值的一个真子集。例如在图 6.6 中，在查找键S上，索引文件中的$\{s_1, s_2, \ldots, s_i, \ldots, s_m\} \subseteq$ 数据文件中的 $\{s_1, s_2, \ldots, s_i, \ldots, s_j, \ldots, s_m, \ldots, s_n\}$。也就是说，并非数据文件中所有的数据记录都一定在索引文件中有对应的索引项，这样的索引称为稀疏索引。

（2）散列技术，也称为哈希（Hash）索引机制。

散列技术利用一个散列函数（Hash function）实现纪录域取值到记录物理地址间的直接映射关系。这里的纪录域就是查找键，也称为散列函数的散列域或排序域。

当需要访问数据文件中查找键值为s_i的某个或某些目标纪录时，将s_i作为散列函数h的输入，计算得出的散列函数输出值$h(s_i)$就是目标记录在数据文件中的物理地址。

在一个文件中查找某个或某些特定文件记录时，需要给出记录应满足的查询条件。一般而言，这种查询条件形如“文件记录在它的某个或某些纪录域/属性上的取值满足…条件”。这些用于在数据文件中查找记录的属性或属性集合就是在 6.3.2 节中定义的文件查找键。

对于存储有关系表数据的数据库文件，该文件的查找键可以是关系表的主键或候选键，也可以是由关系表的其它非主属性组成。

3. 有序索引

文件的结构有堆文件、顺序文件、聚集文件、索引文件和散列文件等 5 种形式。数据库中的数据文件经常采用顺序文件结构，文件的数据记录按照某个特定的查找键值的升序或降序顺序地存储在文件中。本节讨论的数据文件均假设采用顺序文件结构。

当需要采用有序索引机制快速访问一个数据文件时，首先要为该数据文件建立一个索引文件，索引文件是索引记录或索引项的集合。

索引文件建立方法如下：首先选定数据文件中的某个或某些纪录域作为查找键，然后建立起**数据纪录**在查找键上的取值与该纪录的物理地址间的映射关系，组成**索引项**。所有索引项作为**索引记录**存储在索引文件中。索引文件根据某个特定的查找键值的升序或降序存储索引纪录，并且组织为顺序文件。

数据文件和索引文件是有序索引技术中的 2 个主体，数据文件也称为被索引文件（Indexed file）或主文件。

索引建立在查找键上。如果对一个数据文件需要从几个方面去查询文件记录，可以定义多个查找键，针对每个查找键建立相应的索引文件。因此，一个数据文件可以有多个查找键和多个索引文件。对一个特定的索引文件，可以根据这个索引文件的查找键去访问索引文件中的索引纪录，再根据索引记录中的数据记录物理地址去直接访问数据文件记录。

对 2.8 节应用案例中的关系模式“商品（商品编号，商品类别编号，商品名称，生产日期，单价，商品描述信息）”进行简化，选取部分属性，构造出关系模式为 Goods（类别，商品名称，商品编号, 单价）的关系表 goods。该关系表包括 9 个元组，按顺序文件方式存储，查找键为“商品类别”，主键为商品编号。文件包括 9 个记录，记录按照“商品类别”的汉语拼音的升序排列，例如“家电 jiadian”排在“酒类 jiulei”前面。各商品的“商品名称”中忽略了商品的具体品牌。

(类别， 商品名称，商品编号，单价)

服装	西服	X-002	700	
家电	洗衣机	X-100	1500	
家电	冰箱	B-301	3500	
家电	彩电	C-025	6000	
酒类	白酒	B-200	450	
书籍	教材	J-106	50	
鞋类	皮鞋	P-025	420	
鞋类	运动鞋	Y-158	660	
烟	纸烟	Z-002	120	

图 6.7 顺序文件 goods

本节后续部分将以 goods 文件为例，说明索引技术的相关概念和技术。

有序索引作为基于索引文件的索引技术，需要考虑的 2 个关键问题是（1）如何组织索引文件中的索引记录。（2）如何从索引文件出发，访问数据文件中的数据记录。

有序索引技术有多种多样的形式，根据索引（文件）本身的性质，可以从以下几方面对有序索引进行分类。

（1）聚集索引和非聚集索引。

一个数据文件上可以建立多个索引文件，数据文件和每个索引文件都根据各自的查找键有序地组织为顺序文件。

对数据文件和它的一个特定的索引文件，如果数据文件中数据记录的排列顺序与索引文件中索引项的排列顺序相一致，或者说，索引文件按照其查找键指定的顺序与数据文件中数据记录的排列顺序相一致，则该索引文件称为**聚集索引**（Clustering index）。否则，该索引文件称为**非聚集索引**（Nonclustering index）。

（2）主索引和辅索引。

在数据文件的包含主键的属性集上建立的索引称为主索引（Primary index）。在数据文件的非主属性上建立的索引称为辅索引（Secondary index）。

（3） 稠密索引和稀疏索引。

如果数据文件中的每个查找键值在索引文件中都对应一个索引纪录，则该索引文件称为稠密索引（Dense index）。如果只是一部分查找键的值有对应的索引纪录，则该索引文件称为稀疏索引（Sparse index）。

（4）单层索引和多层索引。

单层索引也称为线性索引，其特点是索引项根据键值在索引文件中顺序排列，组织成一维线性结构。多层索引则将索引文件中的索引项组织成树形结构，以提高索引文件的访问效率。

有序索引还包括一些特殊的索引技术，如位图（Bitmap）索引技术和网格（grid）索引技术。

6.4.2 聚集索引

聚集索引是其查找键指定的顺序与数据文件中数据记录的排列顺序相一致的索引文件。

以图 6.7 所示的顺序文件 goods 顺序文件为例介绍聚集索引的三种实现方法：稠密索引、稀疏索引和多层索引。

1. 稠密索引和稀疏索引

稠密索引对数据文件中的每个查找键值都建立一个索引纪录，而稀疏索引只对部分查找键值建立了索引纪录。

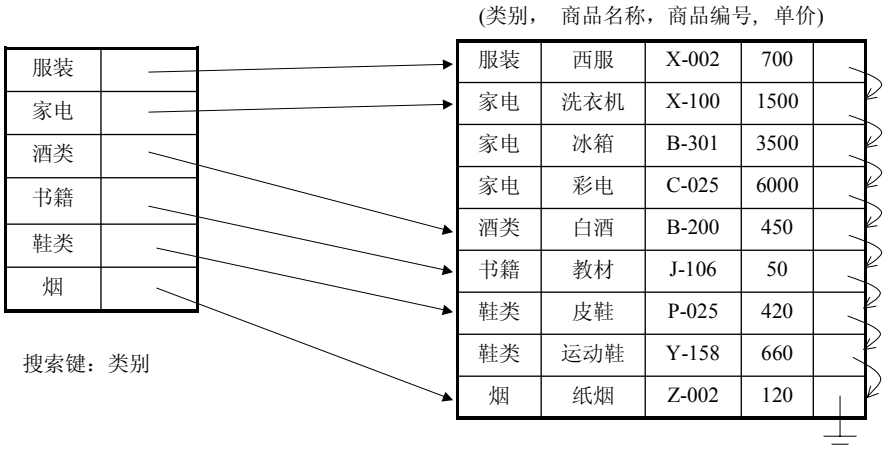


图 6.8 聚集、稠密索引

在图 6.8 中，数据文件 goods 的查找键为“类别”，索引文件同样按照查找键“类别”组织索引项，因此属于聚集索引。对数据文件中查找键“类别”的 6 个取值“服装，家电，酒类，书籍，鞋类，烟”，索引文件中有 6 个索引项与之对应。因此，该索引文件为稠密索引。

基于稠密索引的数据文件访问效率高。例如，例如查找“家电”记录时，首先在索引中查找“家电”的索引记录。如能找到，则沿着索引记录中的指针到达“家电”的第 1 个数据记录，然后再沿着数据记录中的地址指针顺序访问后续 2 个“家电”记录。

图 6.9 中的索引文件只对数据文件中查找键“类别”的 3 个取值“服装，酒类，鞋类”建立了索引项，因此属于稀疏索引。

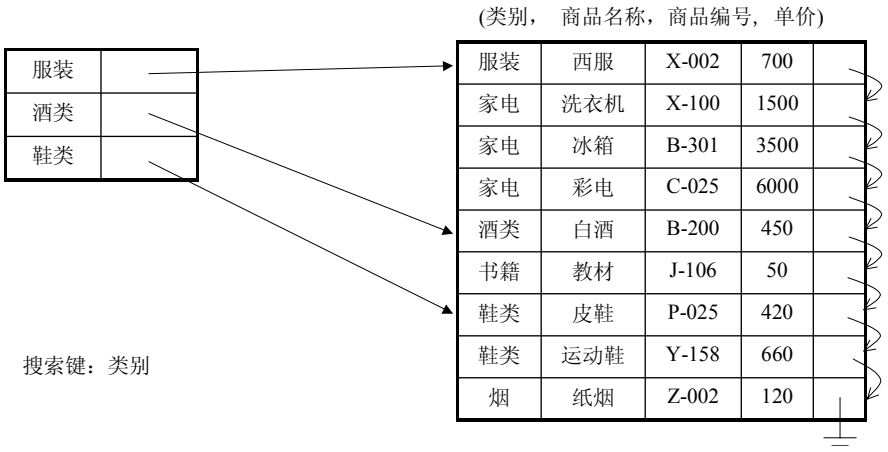


图 6.9 聚集、稀疏索引

根据稀疏索引查找“家电”记录时，先在索引中查找“家电”数据记录所在的范围。根据类别名称汉语拼音的顺序，“家电”在“服装”和“酒类”之间，因此沿着“服装”索引记录中的指针到达主文件中的“服装”数据记录，然后沿数据纪录的指针链，后向顺序查找，依次找到“家电”的 3 个数据纪录。

相比之下，在带稠密索引的主文件中，查找速度较快；而带稀疏索引的文件中查找较慢，但稀疏索引的空间较小，因此插入、删除操作时指针的维护量相对要少些。

此处去掉一段！！！

如果数据文件按某个查找键有序地排列组织其数据纪录，在该查找键上再建立一个索引文件，不管索引本身是稠密还是稀疏的，该索引文件中查找键值的顺序和索引项的排列顺序与数据文件中数据记录顺序一定是一致的，该索引文件一定是聚集索引。这是因为数据文件和索引文件采用了相同的查找键，并且都组织成顺序文件。

图 6.8 和图 6.9 中的 2 个索引文件使用的查找键与数据文件 goods 用于排序的查找键都是“类别”，索引纪录的排列顺序和数据文件记录的排列顺序是一致的（都是按照类别名称汉语拼音升序排列），因此这 2 个索引都是聚集索引。

如果一个数据文件按照某个查找键组织为顺序文件，同时又对数据文件建有聚集索引，则该数据文件称为**索引顺序文件**（Index-sequential file）。对索引顺序文件而言，既可以对数据纪录进行顺序访问（Sequential access），又可以根据数据文件上的聚集索引对数据记录进行直接访问（Direct access）。

例如，图 6.8 和图 6.9 中的数据文件 goods 就是索引顺序文件。

2. 多层索引

图 6.8、6.9 均是单层索引。单层索引中，索引项组织成线性结构，每个索引项直接指向数据文件中的数据记录。

当数据文件很大时，即使采用稀疏索引，建成的索引文件也会很大，导致对索引文件本身的查询效率非常低。

例如，1 个数据文件有 100,000 个记录，每个磁盘块可存储 10 个记录，需要 10,000 个

磁盘块。若以块为基本单位建立索引文件，则共有 10,000 项索引记录。假设 1 个磁盘块可存储 100 个索引记录，那么需要 100 个磁盘块来存储全部索引记录。

索引文件较小时，可以常驻内存，对索引文件的访问还是较快的。如果索引文件很大，则只能以顺序文件形式存储在磁盘上。数据文件访问过程中需要读取某个索引项时，必须将该项所在的磁盘块调入内存。假设索引文件占据b个磁盘块，如果采用顺序查找，则最多要读b块。如果采用二分查找法，读的块数是「$\log_2 b$」。例如上面提到的 100 个索引块，二分查找的次数是 7 次。设读 1 块的时间是 30ms，则读 7 块的时间就要 210ms。

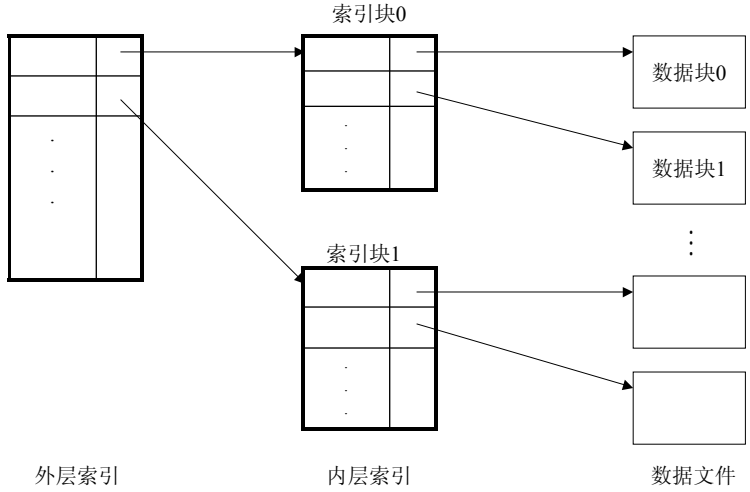


图 6.10 二层稀疏索引

为解决这个问题，可以对索引文件中的索引项本身再建立一级稀疏索引，组成 2 层索引结构，用于快速查找索引项，如图 6.10 所示。根据查找键查询文件记录时，首先在外层索引中使用二分法查找，找到 1 个外层索引项，该索引项的查找键值小于或等于给出的查找键值；然后沿着外层索引项中的指针到达内层索引块；在内层索引块可用顺序查找或二分查找找到相应的内层索引项，这个索引项中的指针指向了需要访问的数据记录。

进一步地，可以建立多层树型索引结构来快速定位大数据量文件中的数据记录。多层索引的典型例子是 6.4.3 节中的B^+-树索引。它将索引文件中的索引项组织成特殊的B^+-树结构，树中叶节点直接指向数据文件的记录。

3. 索引更新操作

数据文件中数据记录的插入、删除可能会引起索引文件的修改。下面以单层索引为例。

（1）插入操作

- ✧ 首先，用插入记录的查找键值找到插入位置。
- ✧ 如果是稠密索引并且查找键值在索引文件中未出现过，则为插入记录构造新的索引项，将索引项插入索引文件中。
- ✧ 如果是稀疏索引，并且 1 个索引记录指向 1 个磁盘块，当磁盘块能放得下新记录时，不必修改索引；如果要加入新的磁盘块，则为插入记录构造新的索引项，并将索引项插入索引文件中。

（2）删除操作

- ✧ 首先在数据文件中找到被删记录。
- ✧ 如果符合查找键值的记录在文件中只有一个，则在数据文件中删除记录后，接着修改索引文件。
- ✧ 修改索引文件时，对稠密索引，从索引中删除被删记录相应的索引记录；对于稀疏索引，如果被删记录的查找键值在索引文件中出现，那么用数据文件中被删记录的下一个数据记录的查找键值 A 替换，如果 A 已在索引文件中出现，被删记录的相应索引记录也应

从索引文件中删除。

4. 非聚集索引

在一个数据文件上可以建立 1 个聚集索引和多个非聚集索引，非聚集提供了按照多个查找键访问数据文件的方法。非聚集索引一定是稠密索引，不可能是稀疏索引。

图 6.11 中，索引文件中索引项按照“商品编号”排序，数据文件中的数据纪录则按照“商品类别” 排序，因此该索引文件属于非聚集索引。

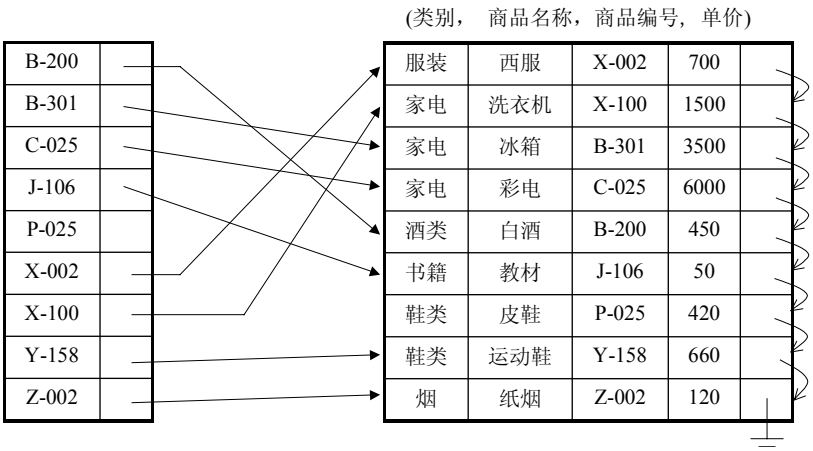


图 6.11 非聚集、主索引

在采用非聚集索引的数据文件中，具有相同查找键值的记录可能分布在数据文件各处，查找时无法利用数据纪录中按数据文件查找键值建立的指针链，因而查找速度较慢。

在聚集索引中可以采取顺序查找方法，减少了读磁盘块的次数。但在非聚集索引中，由于同一个查找键值的数据记录分布在文件各处，因此无法根据非聚集索引查找键顺序扫描文件，每读一个数据记录几乎都要执行一次读磁盘块到内存的操作。

数据记录插入或删除时，需要修改非聚集索引，修改方法与聚集索引修改类似。

5. 主索引和辅索引

主索引以数据文件的主键作为索引文件的查找键、建立起的索引文件。在图 6.11 中，索引文件建立在主键“商品编号”上，因此是主索引，也是稠密索引。

在图 6.8、6.9 中，由于 2 个索引文件是建立在非主属性“类别”上，“类别”并不是数据文件“goods”的主键，因此这 2 个索引是辅助索引。

在实际关系数据库系统中，关系表经常按照主键的顺序将元组排列存储在数据库文件中，并且以主键作为查找键建立起索引文件。该索引文件既是主索引，也是聚集索引。

6.4.3 B^+-树索引技术

前面提到的索引顺序文件的主要缺点是随着文件数据量的增加，无论是顺序查找或索引查找，其性能都会恶化。虽然可以通过重组文件来改善，但频繁的重组增加了系统的负担。为了改善索引结构的性能，可以采用多层树索引。

B-树和B^+-树是 2 类典型的树索引，具有高效、易变、平衡和独立于硬件的优点，在数据库系统中得到广泛应用，成为最重要的索引结构动态文件。动态文件是指文件的记录在使用中会发生变化，如被插入和删除。本节介绍B^+-树索引。

1. B^+-树结构

B^+-树具有平衡树（Balanced Tree）的特点，数据库技术中平衡树的形式定义如下：

一棵 *m* 阶平衡树或者为空，或者满足以下条件：

（1）每个节点至多有 m 棵子树；
（2）根节点或为叶节点，或至少有两棵子树；
（3）每个非叶节点至少有⌈m/2⌉棵子树；
（4）从根节点到叶节点的每条路径的长度相同，即叶节点在同一层次上。

一棵m阶B^+-树是平衡树，其组织方式为：
（1）每个节点中至多有m-1 个查找键值$K_1, K_2, \ldots, K_{m-1}$, m个指针$P_1, P_2, \ldots, P_m$。

P_1	K_1	P_2	…	P_{m-1}	K_{m-1}	P_m

图 6.12　B^+-树的节点结构

（2）叶节点的组织方式

每个叶节点是<指针P_i，查找键值K_i>的集合（$1 \le i \le m$-1），指针指向数据文件中的数据记录，即指针P_i指向查找键值为K_i的数据记录。称<指针P_i，查找键值K_i>为**索引项**。

如果查找键恰好是数据文件的主键，则叶节点中的指针直接指向数据文件中的记录；如果查找键不是数据文件的主键，并且查找键值的顺序与数据文件的排列顺序不一致，那么叶节点中的指针指向一个桶，桶中存放的指针指向具有该查找键值的数据记录。

每个叶节点中至少应有⌈$(m-1)/2$⌉个查找键，最多有m - 1 个查找键，并且查找键值不允许重复。如果B^+-树索引是稠密索引，那么每个查找键值必须出现在某个叶节点中。

叶节点中最后一个指针P_m指向下一个叶节点（按查找键值顺序）。根据指针P_m，可以方便地对数据文件进行顺序访问。

对图 6.7 的数据文件goods，图 6.13 给出了其 3 阶B^+-树的叶节点结构的例子。

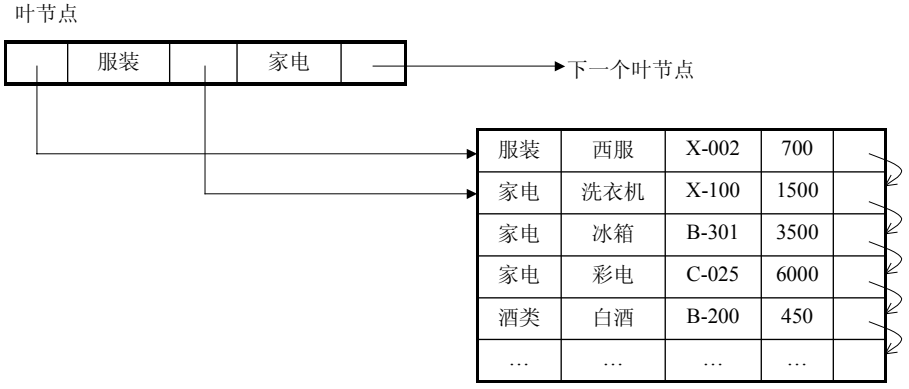


图 6.13　3 阶B^+-树的叶节点, $m = 3$

（3）非叶节点的组织方式

B^+-树中的非叶节点组成叶节点上的一个多级稀疏索引。每个非叶节点是索引项<指针P_i，查找键值K_i>的集合。非叶节点（不包括根节点）中至少有⌈$m/2$⌉个指针，至多有m个指针，指针的数目定义为节点的“扇出端数”(Fanout)。

如果非叶节点有n个指针，对于第i个指针P_i（$i = 2, 3, \ldots, n-1$），P_i指向的子树中所有查找键值均小于K_i，但大于或等于K_{i-1}。指针P_n指向的子树中所有查找键值均大于或等于K_{n-1}，指针P_1指向的子树中所有查找键值均小于K_1。

只有根节点中的查找键值数目可以小于⌈$m/2$⌉，其它节点中查找键值数目至少应有⌈$(m-1)/2$⌉个。

对图 6.7 的数据文件goods，图 6.14 和图 6.15 分别给出了完整的 3 阶B^+-树索引和 5 阶B^+-树索引。图中空指针和指向数据记录的指针未画出。

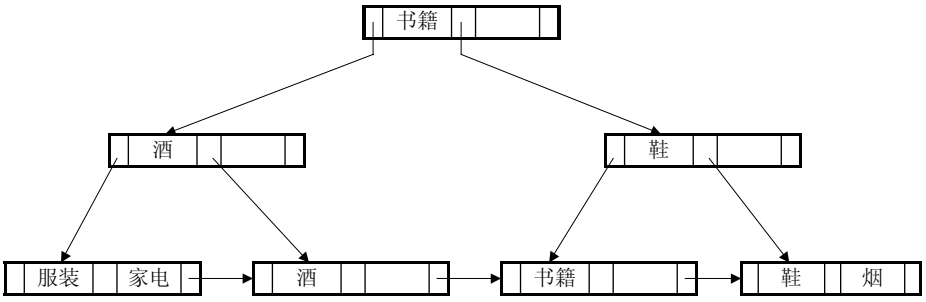


图 6.14　3 阶B^+-树索引，$m = 3$

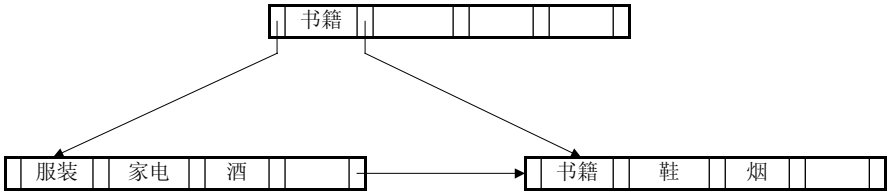


图 6.15　5 阶B^+-树索引, $m = 5$

2. B^+-树的查询

当用户查询查找键值为K的所有记录时，首先在根节点中找大于K的最小查找键值（设为K_i），然后沿着K_i左边的指针P_i到达第二层的节点。如果根节点中有n个指针，并且$K > K_{n-1}$，则沿着指针P_n到达第二层的节点。

在第二层节点中，用类似的方法找到一个指针，进入第三层的节点……，一直到进入B^+-树的叶节点，找到一个直接指向数据文件记录或桶的指针，桶中存放指向数据文件记录的指针，根据该指针访问所需数据记录。

如果数据文件中查找键值有W个不同的值，则对于m阶B^+-树而言，从根节点到叶节点的路径长度不超过$\lceil \log_{\lceil m/2 \rceil} W \rceil$。

下面讨论B^+-树索引查询中，查询次数与文件存储块数的关系。假设在B^+-树索引中，每块存储一个节点，占 4096 字节。查找键的长度为 12 字节，指针占 8 个字节，那么每块大约可存储 200 个查找键值和指针，m约为 200。如果查找键的长度为 32 字节，指针仍为 8 字节，那么每块大约可存储 100 个查找键值和指针，即m约为 100。

m=100 时，如果文件中查找键有 100 万个值，一次查询需读索引块的数目为$\lceil \log_{50} (1\,000\,000) \rceil = 4$。如果$B^+$-树索引的根节点常驻内存，那么查找时只需再读三个索引块即可。

B^+-树的结构与内存中普遍使用的二叉排序树的主要区别在于节点的大小以及树的高度。在二叉排序树中，每个节点很小，只有一个键值和两个指针。而B^+-树中，每个节点很大，可以是磁盘上的一个块，包含更多查找键值和指针。二叉排序树显得瘦而高，而B^+-树显得胖而矮。在平衡的二叉排序树中，如果查找键的值有 100 万个，那么查找的节点数约为$\log_2 (W) = 20$；如果文件采用二叉排序树方法，需要读 20 个物理块。而在同样情况下，B^+-树索引只要读 4 块即可。因此，在外存中不用二叉排序树结构，而是使用B^+-树结构。

3. B^+-树的更新

B^+-树索引文件中的插入、删除操作比查找复杂得多。在插入数据记录时，有可能导致叶节点分裂，并引起上层节点的分裂和B^+-树层数的增加。在删除数据记录时，则有可能出现相反现象。下面就是否出现分裂与合并情况分别讨论。

（1）不引起索引节点分裂的插入操作

假设插入记录的查找键值为K_0。首先从根节点出发，使用B^+-树查找方法在叶节点中找到某个依赖于K_0的查找键值K_i。如果插入记录的查找键值K_0已出现在叶节点中，则在数据文

件中直接插入记录即可，不必修改索引；如果K_0在叶节点中不存在，那么在叶节点中K_i之前插入K_0值（此时假设叶节点中存在空闲空间），并将K_i及K_i后的值往后移动，使叶节点中查找键值仍然保持排序。然后将新记录插入到数据文件中。

（2）不引起索引节点合并的删除操作

首先使用B^+-树查找方法在数据文件中找到被删记录，将其删除。如果数据文件中还存在具有被删记录查找键值的其它记录，不必修改索引；否则应该从叶节点中删除该查找键值和相应的指针。此时假定叶节点中查找键值的数目仍然不小于$\lceil (m-1)/2 \rceil$。

（3）引起索引节点分裂的插入操作

如果插入一个数据记录时，需要在叶节点中插入其查找键值，并且叶节点中已放满查找键值，即叶节点中不同查找键值的数目达到了B^+-树中每个树节点允许的上限m-1，那么叶节点应分裂成两个。

例如在图 6.14 所示的 3 阶B^+-树文件中，插入 1 个查找键值为“工具”的数据记录，那么树中左边第一个叶节点就要分裂成两个叶节点，如图 6.16 所示。

图 6.16 插入“工具”后，叶节点的分裂

1 个叶节点分裂时，要将它所存储的 m 个查找键值分放在两个叶节点中。一般，前$\lceil m/2 \rceil$个查找键值放在原来的叶节点中，而余下的查找键值放在新的叶节点中。

叶节点分裂后，必须在其父节点中插入新节点中的最小查找键值，如图 6.17 中的“家电”。如果父节点中已放满了查找键值，那么父节点也需要分裂成两个节点，再在上一层节点中加入一个新的查找键值。有可能往上直至根节点也要分裂，从而产生新的根节点，即B^+-树高度增加了一层。

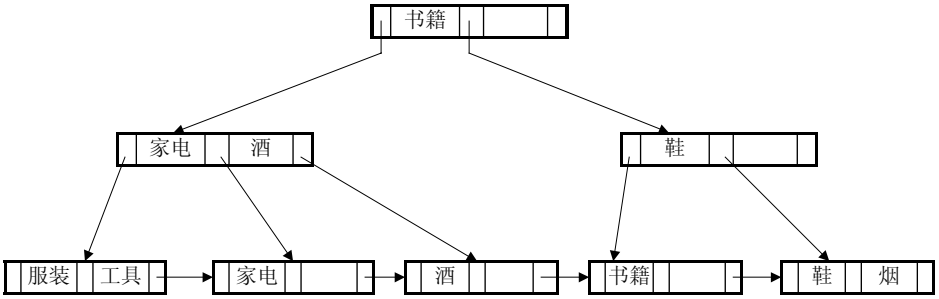


图 6.17 在图 6.14 中插入“工具”后的B^+-树

（4）引起索引节点合并的删除操作

如果删除 1 个数据记录时，需要在叶节点中删除该记录的查找键值，并且该查找键值删除后叶节点中查找键值数目小于$\lceil (m\text{-}1)/2 \rceil$，那么这个叶节点有可能被从树中删掉。

例如，如果在图 6.17 中删除“家电”数据记录，会导致图中第 2 个叶节点中删除<家电，指针>后成为空叶节点，因此该叶节点被删除。随之引起在第 2 层的左边父节点中删除指向“家电”叶节点的指针和指针左边的查找键值“家电”，即删除<家电，指针>。修改后的索引如图 6.18 所示。

在图 6.18 中，删除数据文件中“书籍”所对应的数据记录会导致B^+-树中第 3 个叶节点在删除查找键“书籍”和指针后成为空叶节点，该叶节点从而被删除掉。进而引发在该叶节点的父节点（第 2 层右边节点）中删除指向该叶节点的指针，使得该父节点中只剩下一个<鞋，指针>，这不符合B^+-树对非叶节点的要求。此时由于其相邻的同层左兄弟节点（第 2

层左边节点）中还有空闲空间，因此这 2 个第 2 层节点可合并成一个节点。进一步地，导致了根节点中也缺少一个指针，不符合B^+-树要求，因此可将根节点直接删去，如图 6.19 所示。此时B^+-树的高度降低了一层

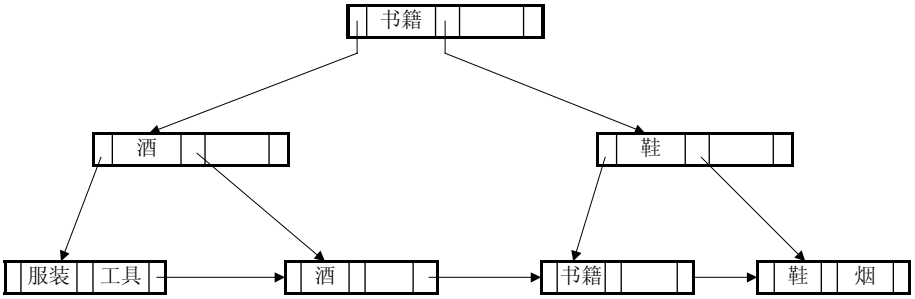


图 6.18 在图 6.17 中删除“家电”后的B^+-树

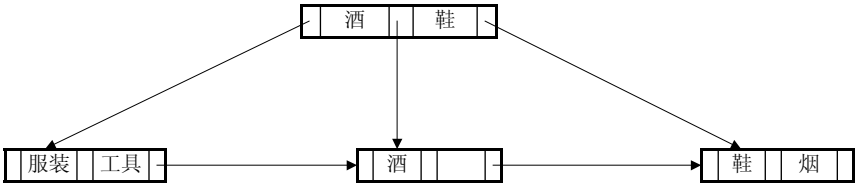


图 6.19 在图 6.18 中删除“书籍”后的B^+-树

B^+-树中的删除操作可能引发节点合并。如果节点合并后无法满足B^+-树的规范要求，只能重新调整树的结构。

例如，考虑在图 6.17 中从叶节点中删除查找键值“书籍”。“书籍”所在的叶节点在删除了查找键值“书籍”后成为空叶节点，随之被从树中删除掉。该叶节点的被删除使得它的父节点（第 2 层中的右边节点）需要删除指向“书籍”叶节点的指针，从而导致该父节点只包含一个指针，不符合B^+-树的规范要求。而且，此时该父节点的左兄弟节点（第 2 层左边节点）中已放满了查找键值（2 个查找键值“家电”和“酒”，达到m-1=2），该父节点无法与其左兄弟节点（第 2 层中的左边节点）合并。只能重新调整树结构：该父节点将它的左兄弟节点中的<酒，指针>移到自己名下，同时父节点和它的左兄弟节点的共同父节点（这里是根节点）的查找键值应修改为“酒”。如图 6.20 所示。

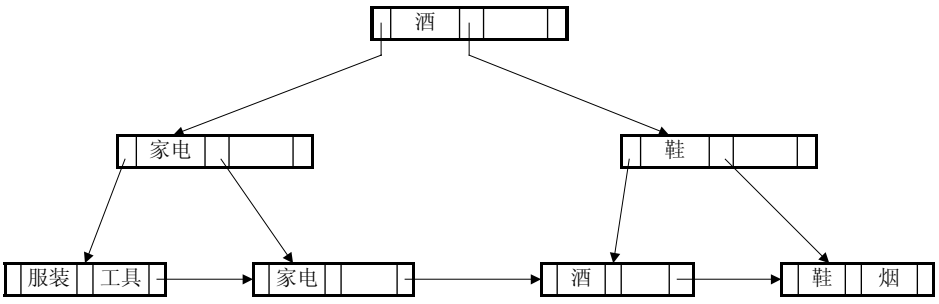


图 6.20 在图 6.17 中删除“书籍”后的B^+-树

B^+-树索引的插入、删除比较复杂，但这些操作的执行时间与B^+-树高度成正比。前面已提到，对包括 100 万个记录的数据文件，其B^+-树索引只有 4 层，因此树操作的时间代价并不大。

B^+-树具有较小的搜索代价，较好地解决了数据记录的插入、删除和未用空间回收等存储组织问题。B^+-树可在操作中动态地进行维护，通过压缩树节点中索引项的办法来降低树的高度，减少树搜索时读磁盘块次数，实现机制独立于具体的物理存储设备。由于具有这些

良好特性，B^+-树广泛应用于数据库系统。

4. B^+-树文件组织

前述B^+-树索引将索引文件和数据文件截然分开。B^+-树中的所有叶节点处在同一层次上，并且利用叶节点中的指针指向数据文件中的数据记录，或指向一个桶，再由桶内指针指向数据记录。在这样的结构中，B^+-树不仅是一个索引，还是一个文件中记录组织者。也就是说，B^+-树提供了一种文件中记录的组织方式，形成一种特定的文件结构。

可进一步改进B^+-树索引文件组织。B^+-树叶节点不是存储指向数据记录的指针，而是直接存储数据记录本身，那么这种结构称为B^+-树文件组织。图 6.21 给出了一个具体例子。

访问数据文件中的数据记录时，根据查找键值沿树中节点到达叶节点后，可以直接找到数据记录。不必再向传统B^+-树那样，再沿着叶节点中的地址指针去寻找数据记录，文件的性能得到进一步提高。虽然数据记录长度要比指针大很多，也就是每个叶节点可存储的记录数目要小于非叶节点中的指针数目，但是仍然要求每个叶节点中至少有一半空间是装着记录的。

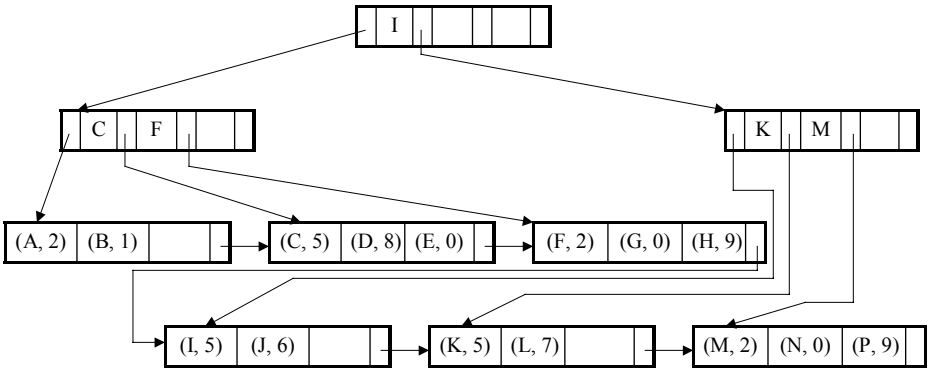


图 6.21 B^+-树文件组织

B^+ 树文件组织的插入、删除操作与B+-树索引记录的插入和删除操作是一样的，也有可能会引起节点的分裂或合并，层数的增加或减少。

5. B^+ -树性能分析

设数据文件中的记录数为有N。数据文件被组织成 $m=2d-1$和 $n=2e-1$的B^+-树，其中m是非叶节点中不同查找键值K_i的个数，n是叶节点中不同查找键值K_i的个数。显然，树中的叶节点个数不会超过 N/n，叶节点的父节点不会超过 $N/(m\,n)$，叶节点的父节点的父节点不会超过 $N/(m^2\,n).\cdots$，可这样一直推算到树根。各层次与树中节点数关系如表6.1所示。

表 6.1 B^+-树的层次与树中节点数的关系

层号	非叶节点数	叶节点数	数据记录（主键值）数
1	0	1	n
2	1	m	$m\,n$
3	$m+1$	m^2	$m^2\,n$
4	m^2+m+1	m^3	$m^3\,n$
…	…	…	…
i	$m^{i-2}+m^{i-1}+\ldots+m+1$	m^{i-1}	$m^{i-1}\,n$

显然有：　　$N=m^{i-1}\,n$，　$i=\log_m(N/n)+1$

当每个节点均装满时，有：

$$i=\log_{2d-1}(N/(2e-1))+1$$

当每个节点只装到其下限值时，有：

$$i = \log_d(N/e) + 1$$

所以B^+-树的层次取值范围为：

$$\log_{2d-1}(N/(2e-1)) + 1 \leq i \leq \log_d(N/e) + 1$$

当 N=20,000，取 $d = e = 100$，则有 $2 < i < 3$，也就是说，此种情况下树的高度最大 3，搜索代价较小。

以下去掉原文中的一段！！！

6. B-树索引

数据库系统中另外一种平衡树索引是 B-树索引。

B-树索引的结构、性质、数据访问机制类似于B^+-树索引，主要区别在于B-树中查找键值可以出现在任何节点上，但每个查找键值只能出现一次，而B^+-树中 1 个查找键值可以出现在多个树节点中。因此，B-树中的节点数目要比B^+-树少。

在B-树中，如果 1 个查找键值出现在某个非叶节点，则这个非叶节点应附加上一个指向数据记录的指针；如果查找键值出现在叶节点，则叶节点中的指针直接指向数据记录。同B^+-树一样，叶节点指针也可指向一个桶，桶内存放指向数据记录的指针。

在B-树中，查询 1 个特定查找键所需的树搜索次数取决于查找键值在树中所处位置，有时未到达叶节点，就已找到了键值。查找键值越接近叶节点，所需搜索次数越多。与B^+-树一样，B-树的查询时间复杂性仍然是对数级的。

与B^+-树相比，B-树的体积比B^+-树小，树搜索速度快，但B-树的删除操作相对复杂一些。目前大多数数据库系统使用B^+-树索引结构，如Oracle数据库。但也有一些数据库系统使用的是B-树索引结构，典型代表是SQL Server2000 数据库。SQL Server2000 采用B-树组织数据文件的聚集索引和非聚集索引信息。对聚集索引，B-树叶节点中直接存储关系表数据，属于B-树索引文件组织。

6.5 散列技术

当数据库文件组织成顺序文件时，为了提高文件记录存取速度，可以在文件上建立有序索引，如B^+-树索引，但索引文件本身所占存储空间和索引文件搜索的时间代价仍然很大。散列（Hashing）提供了一种不必通过索引文件就可以快速访问数据库文件的方法。利用散列技术，可以有效组织数据库文件，构造数据文件索引，支持数据库文件快速访问。本节介绍数据库中使用的散列技术

6.5.1 散列文件组织

1. 散列概念

散列是一种快速查找技术，它利用定义在文件记录上的查找键（也称为散列域），通过计算一个散列函数，以散列函数值作为记录的物理地址，实现对文件记录直接快速访问。

用散列方法组织存储一个文件时，首先需要指定文件记录的一个（或一组）域作为查找键，查找键也称为散列域。然后定义一个查找键上的函数，称为散列函数。散列函数的输入为文件记录的查找键值，函数的输出为记录的物理地址，如记录所在的磁盘块地址。

在数据库技术中，一般使用桶（Bucket）作为基本的存储单位。一个桶可以存放多个文件记录。桶可以是磁盘块，也可以是比磁盘块大的空间。因此，文件记录的物理地址可以是

记录所在的桶，散列函数的输出可以是桶号。以下均假设文件记录采用桶号作为其物理地址。

散列函数可形式化地定义为函数映射 h : K → B, 其中 K 是文件记录中出现的所有查找键值的集合，B 是所有磁盘块地址或桶地址的集合。

文件可以组织为散列文件。首先定义文件记录的查找键，将文件的物理存储空间划分为一系列的桶。然后利用散列函数求出每个记录应在的桶号，根据桶号将每个文件记录安排在一个桶中。

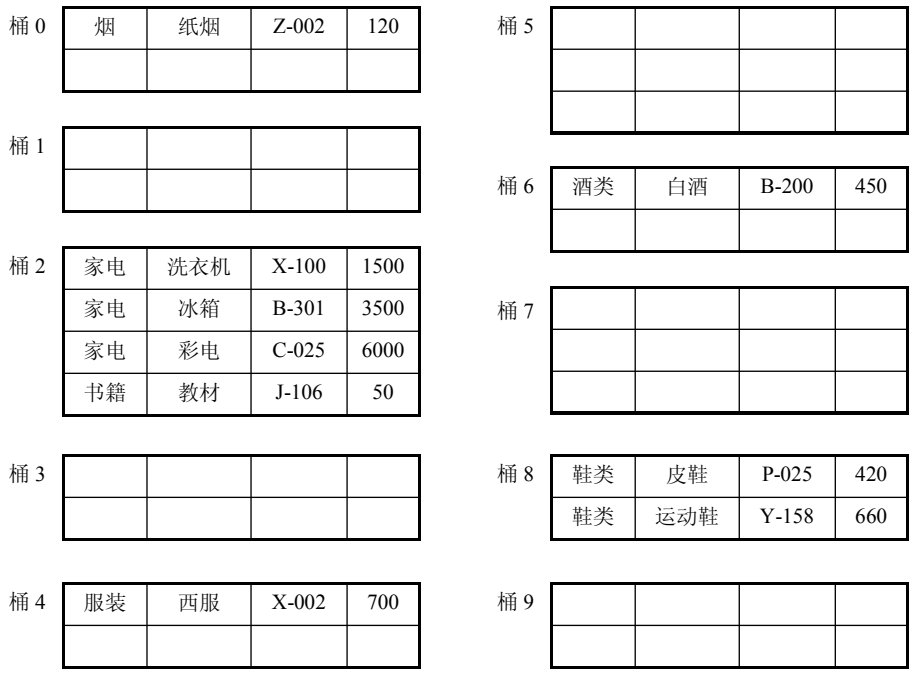


图 6.22 散列文件组织

例如，对图 6.7 中的文件 goods，可以重新组织为图 6.22 中的散列文件。假设查找键是“类别”，文件物理存储空间分为 10 个桶。散列函数设计为：将记录的“类别”属性值中的每个汉字按一定规律转换成一个整数，然后求出这些整数的和并除以 10，得到余数作为该记录的桶号，然后将记录存入相应的桶。例如“家电”转换后的整数和为 42，除以桶数 10,得到余数 2，作为三个“家电”纪录的桶号。“书籍”纪录也被存储到桶号为 2 的桶内。

假设散列文件的查找键是 A，h 是定义在 A 的值域上的散列函数。根据散列函数和查找键，可以方便地在散列文件中进行记录的查找、插入和删除。

当要查找在查找键A上的值等于K_i的文件记录时，首先计算散列函数，找到该记录的的桶地址h (K_i)；由于散列函数有可能是多对一的映射关系，具有不同查找键值的记录通过散列函数可能映射到同一个桶号，因此桶h (K_i) 内有可能有多个记录，记录的查找键值并不一定都是K_i，因此需要进一步在桶内检查各记录的查找键值是否等于K_i，从中找出需要访问的目标纪录。例如，在图 6.22 中，如果查找“书籍”记录，就需要检查桶 2 中的 4 个记录的查找键值。

如果需要插入查找键值为K_i的记录，首先计算该纪录应当在的桶的桶号h (K_i)。如果桶h (K_i)有空闲空间，则直接将该记录插入桶中。如果桶内空间不够，需要采用相应的桶溢出处理机制。

删除记录操作非常方便，先用查找方法将需要删除的记录找到，然后直接从桶内删去即可。

2. 散列函数

散列方法需要有一个好的散列函数。由于设计散列函数时不可能事先精确知道要存储哪些记录查找键值，因此要求散列函数在将查找键值转换成存储地址（桶号）时，尽可能均匀地将查找键值分布到各个桶中，具体地应满足下面两个条件：

（1）地址的分布是均匀的。将所有可能的查找键值转换成桶号以后，要求落在每个桶内的查找键值数目大致相同。

（2）地址的分布是随机的。不管查找键值在实际文件中的分布如何，散列函数值不受查找键值各种顺序的影响，例如字母顺序、长度顺序等，散列函数应该表现为随机的。

散列函数应仔细设计。好的散列函数可以保证文件各桶内的查找时间基本一致，并且查找的平均时间是最小的。设计不好的散列函数则会造成各个桶内的查找时间有长有短。

散列函数的构造方法有：直接定址法、数字分析法、平方取中法、折叠法和质数除余法。其中，质数除余法的原理是将记录的查找键值除以一个质数，以求得的余数作为桶号。图6.22 散列文件就是采用此方法构造出来的。

散列函数在数据结构教材中已有详细介绍，这里不再详述。

3. 桶溢出的处理

散列文件组织将文件的物理空间划分为一系列桶，每个桶的空间大小是固定的，可容纳的文件纪录数目是固定的。因此，如果某个桶内已装满记录，又有新的记录要插入到该桶，就会产生**桶溢出**（Bucket overflow），也称为散列碰撞。产生桶溢出的 2 个主要原因是：

（1）文件初始设计时，为文件记录预留存储空间不足，预留的桶数偏少。

（2）散列函数的“均匀分布性”不好，造成某些桶存满了记录，其它桶内却有较多空闲空间。

设计散列函数时，应根据文件在实际使用过程中最大可能的文件大小决定文件的物理空间大小，一般物理空间至少应有 20%的余量。然后再设计合适的桶数目和桶大小，应尽可能留有一些空闲桶，以利减少桶溢出的机会。

可以用装填因子（也称为负载因子）α 来衡量一个散列文件的物理空间被文件记录占满的程度。α 等于文件中全部记录实际占用的物理空间除以分配给文件的物理空间总量。一般 α 取值范围［0.6, 0.8］。如果 α >0.8，容易产生桶溢出；而 α <0.6，存储空间浪费又太多。

由于散列函数是事先设计的，不可能完全符合文件的实际使用情况，即使散列函数认真设计、存储空间留有一定余量，桶溢出现象也难以避免。因此，需要相应的桶溢出处理机制，下面介绍几种常见的桶溢出处理方法。

（1）溢出桶拉链法（溢出链法）

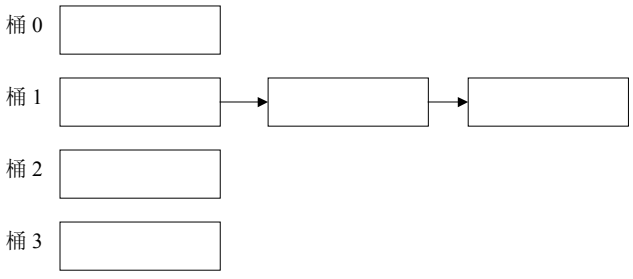


图 6.23 散列结构的溢出链

如果某个桶 b（称为基本桶）已装满记录，又有新记录等待插入桶 b，可以由系统提供一个溢出桶，并用指针链将该桶接在桶 b 的后面。如果溢出桶也装满了，则用类似方法在其后面再链接一个溢出桶。这种方法也封闭散列法（Closed hashing）。图 6.23 给出了该方法示意图。

文件记录的查找不仅涉及基本桶，可能也需要到基本桶后面链接的溢出桶中去查找。

（2）开放式散列法（Open hashing）

本方法中桶集合是固定的，只有基本桶，没有溢出桶。如果有一个桶 *b* 已经装满了记录，但又需要插入新纪录，则在桶集中挑选一个有空闲空间的桶，装入新记录。根据桶的选择方法不同，可分为下面两种方式：

- 线性选择法：在桶 *b* 的下面（以循环的次序）顺序选择一个有空闲空间的桶，将新记录装进去。
- 再散列选择法：采用二次散列方法，跳跃式地选择一个有空闲空间的桶，装入新记录。

开放式散列法可用于构造程序设计语言的编译器和汇编器中的符号表。由于开放式散列法的删除操作比较复杂，因此数据库系统中一般只使用封闭散列法。

散列方法的缺点在于：为了避免桶溢出等问题，必须选择合适的散列函数，设计一个与文件实际使用情况相匹配的好的散列函数比较复杂。而且散列函数也不像索引文件那样可以随着文件中数据纪录的数量变化动态地进行调整。

6.5.2 散列索引

散列方法不仅可以用在文件组织上，也可以用于创建索引结构。索引是形如<查找键值，指针>的索引项的集合，将查找键值与指针组织成散列文件结构，得到的索引文件称为**散列索引**。

散列索引构造方法如下：

（1）为数据文件中每个查找键值K_i建立一个索引记录<K_i ，P_i>；

（2）将这些索引记录组织成散列文件结构：散列文件物理空间由一系列桶组成，利用散列函数根据查找键值将每个索引记录<K_i ，P_i>分布到各个桶中。

下面举个具体例子。对图 6.7 中的文件 goods，可以建立图 6.24 所示的散列索引。散列索引的查找键为“商品编号”，由于该属性为文件的候选键，因此每个数据记录在“商品编号”上的取值均不同，共有 9 个形如<商品编号，指针>的索引项。这些索引项通过散列函数安排在散列文件中，文件由 5 个桶组成，每个桶可容纳 2 个索引项。由于第 2、3、4 个记录的散列值都相同，3 个记录所对应的 3 个索引项被安排在了第 1 号桶及其溢出桶中。

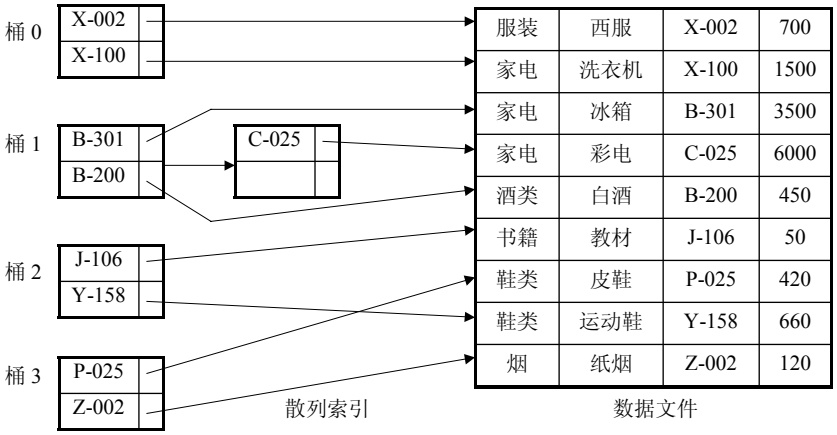


图 6.24 在查找键“商品编号”上的散列索引

在这个例子中，“商品编号”是一个候选键，每个“商品编号”值只与一个数据记录联系。一般地，查找键值可以与多个数据记录相联系，例如以“类别”作为散列索引的查找键时，第 2、3 和 4 个记录有相同的健值。此时，散列索引项中的指针可以指向一个数据记录，在数据文件中再沿着该数据纪录的指针链找到其它具有相同查找键值的记录；散列索引项中的指针也可以指向一个桶，桶内存放多个指向具有相同查找键值的数据记录的指针。这些技

术都是非聚集索引中使用过的。

严格地讲，散列索引是一种非聚集索引结构，不属于聚集索引结构。但散列文件组织提供由索引直接找数据记录的方法，因此可以认为散列文件组织提供了一个虚拟的聚集散列索引。

6.5.3 动态散列

前面提到的散列技术属于静态散列。静态散列的缺点是：

（1）只能有效支持定义在查找键上的具有相等比较的数据操作（如查询、插入、删除）。如果一个数据操作的操作条件不是建立在查找键上或不是相等比较，则数据操作的处理时间与堆文件相同。

（2）对于散列文件组织而言，当散列函数确定以后，文件所有的桶地址及桶空间都确定了。文件记录较少时，存在存储空间浪费。当文件记录超过一定数量后，文件中的溢出桶会比较多，沿溢出桶链的顺序访问会影响文件记录的访问效率。

（3）在实际使用过程中，当数据库数据量发生变化时，原有的散列函数有可能不再适用，因此需要作进一步调整。但散列函数的调整会影响到原有文件的整体组织结构，代价较大。

动态散列技术允许文件的桶空间根据需要随时申请或释放，而且文件空间维护的代价也不大，可在一定程度上克服静态散列的后 2 个缺点。

下面介绍一种动态散列方法－可扩充散列结构。

静态散列中有一种“成倍扩充法”机制。这种方法在需要扩充存储空间时，将桶数扩大一倍，从原来 m 个桶扩大为 $2m$ 个桶，每个桶内的记录数随之减少为原来的一半。当文件数据量已经很大时，将文件空间再扩大一倍，管理代价太大，空间利用率也低。

可扩充散列结构是对成倍扩充法的改进，当数据库文件增大或缩小时，可以通过桶的分裂或合并来适应数据库大小的变化，空间利用率高。重新组织文件时，每次只作用在一个桶上，所带来的性能开销较低。

1. 可扩充散列文件组织

可扩充散列文件组织需要一个均匀性和随机性都比较好的散列函数h，并且要求这个散列函数产生的函数值〔简称为散列值〕较大，是一个由b个二进制位组成的整数。如果$b = 32$，则有 2^{32} （约 4×10^9）个散列值，可以对应 2^{32}个桶。但每个散列值并不是静态地对应一个桶，而是可以根据实际需要不断申请或释放桶。

文件初始创建使用时，不是根据全部 b 位值计算桶地址，而是根据这个 b 位的前 i 位（高位）($0 \leqslant i \leqslant b$) 计算桶地址。随着文件数据量的增长，用于地址计算的位数 i 也随之增加。这 i 位的值组成一个桶号。所有的桶地址放在一张“桶地址表”中，桶地址的位置就是桶号。

图 6.25 是可扩充散列结构的示意图。在桶地址表上方出现的i表示当前情况是取散列值的前i位作为桶地址的位置。在桶地址表中，有时可能某几个相邻的项中的指针指向同一个桶。每个桶的上方出现的整数i_0、i_1, 、i_2、 …, 表示这些桶是根据散列值的前i_0、i_1, 、i_2、…位作为桶地址表中的地址寻找来的。桶地址表和桶的上方出现的i 、i_0 、i_1, 、i_2、…这些值称为散列前缀，并且$i_j \leqslant i$ ，i_j是第j个桶的散列前缀。指向第j个桶的指针数目是 $2^{(i-k)}$，其中 $k = i_j$。

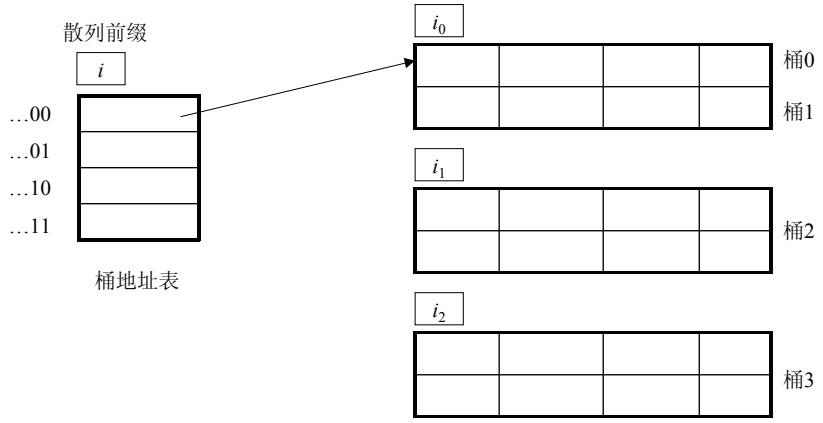


图 6.25 可扩充散列文件组织

2．可扩充散列文件的操作

可扩充散列文件的操作主要有查找、插入、删除三类操作。

（1）查找操作

设散列函数为h (K) 。若查找键值为K_1，桶地址表的散列前缀为i ，那么首先求出h (K_1) 的前i位值m，然后沿着桶地址表位置m处的指针去某一个桶中查找文件记录。

（2）插入操作

当在文件中插入一个查找键值为K_1的记录时，首先用查找操作找到应插入的桶，例如第j个桶。如果桶内有空闲空间，直接插入即可。如果桶已装满记录，那么必须分裂桶，重新分布记录，并插入新记录。分裂桶的过程分成两种情况考虑：

- $i = i_j$, 也就是指向第j个桶的指针只有一个。这时应在桶地址表中增加项数，以保证第j个桶分裂后，在桶地址表中有位置存放指向新桶的指针。可采用增加散列前缀的方法，也就是将i的值增 1。例如，原来i为 4，对应桶地址表中的第 16 项，若i增为 5，则对应了桶地址表中的第 32 项。也就是每一项分裂成相邻的两项，此时桶地址采用了成倍扩充法，但存储数据的桶空间并没有扩大。桶地址表的空间比存储数据的桶空间小得多。

  此时指向第 j 个桶的指针变为两个。再申请一个桶空间（第 z 个桶），将第二个指针值置为指向桶 z，再将第 j 和第 z 个桶的散列前缀置为新的 i 值。接着，将原来第 j 个桶中的记录根据散列值的前 i 位（已变为增加了 1 后的新值）决定是仍留在第 j 个桶还是移到第 z 个桶，并相应地进行分配。然后，将新记录插入到第 j 或第 z 个桶。

  如果分裂以后，原来第 i 个桶的记录仍全部留下，而新记录还要插入到该桶，那么只能用上述方法将桶地址表再次扩大。但这种可能性极小，因为单记录的插入不太可能造成桶地址表扩大两次。如果有的查找键值的记录非常多，那么此时也可考虑使用静态散列中的溢出桶方法，以免桶地址表过于庞大。
- $i > i_j$。此时，指向第j个桶的指针至少两个或两个以上，那么桶地址表不必扩大，只要分裂第j个桶即可。申请一个桶空间（桶z）。将i_j的值增 1，置第j个桶的指针和第z个桶的散列前缀都为新的i_j值。然后在桶地址表中原来指向第j个桶的指针中分出后一半指针，填上桶z的地址，而前一半指针仍指向桶j。再将第j个桶中的记录按散列前缀确定的位数所表示的值重新分布在桶j或桶z中，将新记录插入到桶j或桶z中。

  在这两种情况下需要重新分布记录，但只是对第 j 个桶的操作，与其它桶无关。

(3)删除操作

删除查找键值为K_1的记录时，先用查找方法找到被删记录，然后将记录从所在的桶内删除。如果删除后，桶变为空，这个桶也随之被删除，从而引起桶的合并操作。有时还可能

引起桶地址表的收缩（成倍地收缩）。

下面举一个例子。对图 6.7 中的数据文件 goods，按照动态散列文件结构重新组织，查找键选为“类别”，散列函数将查找键值转换为 32 位散列值，如图 6.26 所示。

服装	西服	X-002	700	
家电	洗衣机	X-100	1500	
家电	冰箱	B-301	3500	
家电	彩电	C-025	6000	
酒类	白酒	B-200	450	
书籍	教材	J-106	50	
鞋类	皮鞋	P-025	420	
鞋类	运动鞋	Y-158	660	
烟	纸烟	Z-002	120	

类别	h (类别)
服装	0010 1101 ……
家电	1010 0011 ……
酒类	1100 0111 ……
书籍	0011 0101 ……
鞋类	1111 0001 ……
烟	1101 1000 ……

散列值只列出高位部分

图 6.26 文件 goods 及其 32 位散列值

开始时文件为空，如图 6.27 (a) 所示。下面将文件 goods 中的 9 个记录逐个插入到可扩充散列文件中去。假设每个桶里最多只能放两个记录。

首先插入（服装，西服，X-002，700）记录，此时散列前缀为 0，桶地址表中只有一项，新记录被插入到桶 0；接着插入goods中的第 2 个记录（家电，洗衣机，X-100，1500），放入桶 0；再插入第 3 个新纪录（家电，冰箱，B-301，3500）。由于桶 0 已满，须分裂，散类前缀由 0 增为 1，即需要使用 2^1=2 个桶。此时，申请一个桶空间，取纪录的散列值“1010 0011 …”的第 1 位作为新的桶号，再重新分配第 1 和第 2 个纪录，插入新纪录。结果如图 6.27 (b) 所示。

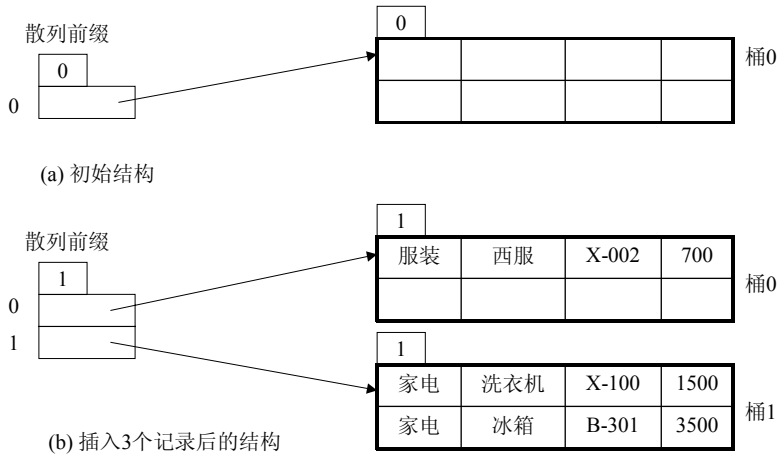


图 6.27 初始文件和插入 3 个记录后的文件

继续插入第 4 个记录（家电，彩电，C-025，6000）。此时桶 1 已满，而本记录的查找键值“类别”是重复的键值，因此不再进行桶分裂，而是申请 1 个溢出桶，在其中插入新纪录，如图 6.28 所示。

考虑插入（酒类，白酒，B-200，450）。据本记录的散列值“1100 0111…”的第 1 位，记录应放在桶 1，但桶 1 已满，因此需要分裂桶 1，桶地址表的散列前缀由 1 增为 2，原有的桶编号也随之修改。桶地址表中第 0 项和第 1 项的指针指向同一个桶，即桶 0、桶 1 合用

一个桶，原来的桶 1 分列成 2 个桶：桶 2 和桶 3。原来桶 1 中的纪录应重新分布，然后将新纪录插入到桶 3 中。

下面插入纪录（书籍，教材，J-106，50）。注意到此时桶地址表的散列前缀已变为 2，因此根据本记录散列前缀值“0011 0101…”的前 2 位，记录应插入桶 0。由于桶 0 当前未满，因此（书籍，教材，J-106，50）被放入桶 0。

再考虑插入纪录（鞋类，皮鞋，P-025，420）。根据本记录散列前缀值“1111 0001…”的前 2 位，记录被插入桶 3；接着插入（鞋类，运动鞋，Y-158，660）。由于本记录查找键值“鞋类”已重复，因此申请 1 个溢出桶，将纪录插入该桶中。

插入文件 goods 的前 8 个记录后的文件组织结构如图 6.29 所示。

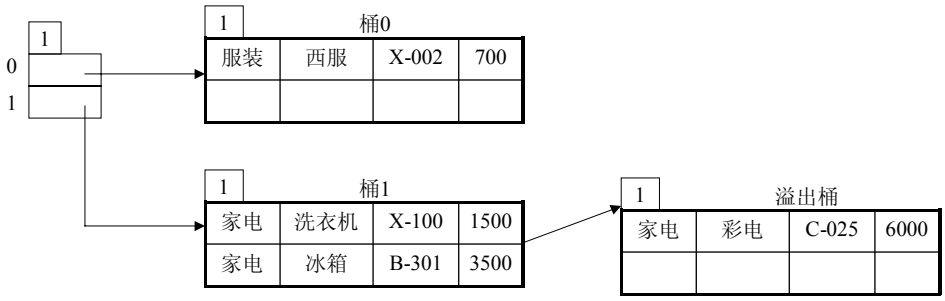


图 6.28 插入 4 个纪录后的散列文件

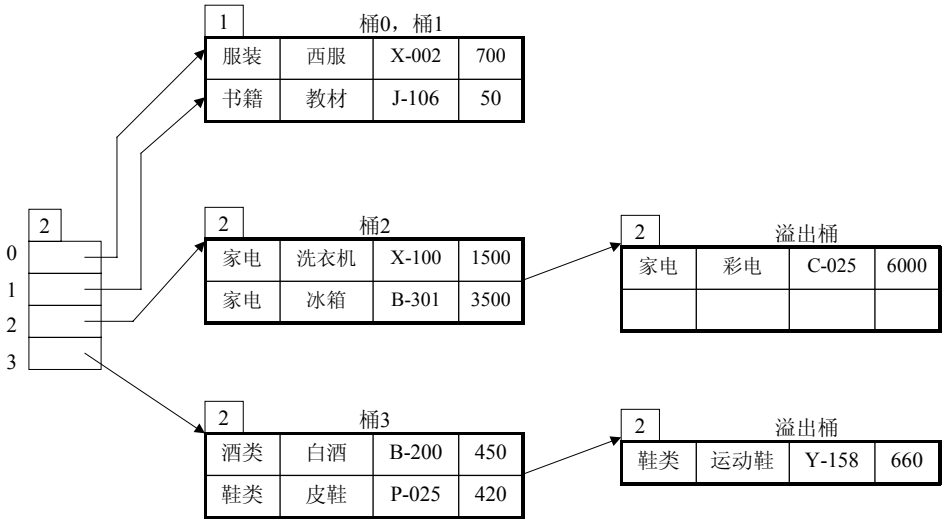


图 6.29 插入 8 个纪录后的散列文件

最后考虑插入第 9 个记录（烟，纸烟，Z-002，120）。据本记录的散列值“1101 1000…”的前 2 位，记录应放在桶 3，但桶 3 已满，因此需要分裂桶 3，桶地址表的散列前缀也由 2 增为 3，原有的桶编号也再次随之修改。桶地址表中的第 0、第 1、第 2 和第 3 项的指针指向同一个桶，原来的桶 3 分裂成桶 6 和桶 7，原桶 3 中的纪录应重新分布，2 个“鞋类”记录根据它们自己的散列值“1111 0001…”的前 3 位移到桶 7 中。新纪录则被插入到第 6 个桶中。

图 6.30 给出了全部插入操作完成后的、包括 9 个记录的基于散列的 goods 文件组织。

可扩充散列技术中，由于桶的编号在桶分裂或合并后不断变化，因此需要桶地址表来存储桶地址，以利查找。与静态散列相比，可扩充散列技术多了一个桶地址表，但这个间接访问对文件的访问性能影响较小。

与有序索引或其它散列技术相比，可扩充散列技术有两个优点：

（1）文件数据量增加时，仍然保持原有的操作和查询性能。

（2）空间开销非常小。额外的开销是桶地址表，由于每个散列值只需一个指针，因此桶地址表只占少量空间。另外，存储记录的桶和溢出桶空间都可以动态地申请和释放。

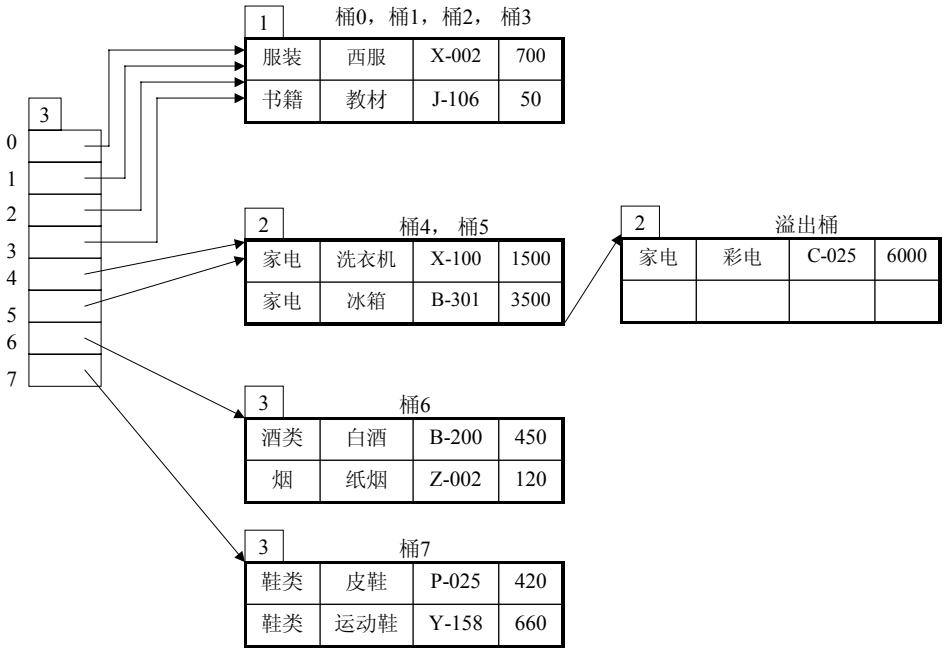


图 6.30 完整的基于散列的 goods 文件组织

6.5.5 有序索引和散列的比较

文件可以采用有序索引组织方案，例如，可以用索引顺序组织或B^+-树组织将记录文件组织成顺序文件。另外，也可以用散列来组织文件。还可以组织成堆文件，其中的记录不以任何特定方式排序。

数据库系统通过DBMS为用户和DBS设计者提供了多种可选方案。各种方案在一定条件下各有其优点。一般地，大多数DBS支持B^+-树索引，并且有可能支持某些形式的散列文件组织或散列索引。

DBS 用户在使用数据库过程中可能使用的查询类型是 DBS 设计者选择有序索引或散列技术时需要考虑的一个重要因素。

如果大多数查询是如下形式的等值查询，

select $A_1, A_2, \ldots, A_n$

from r

where $A_i = c$

，散列方案更可取。因为有序索引的查找所需时间与关系r中A_i值的个数的对数成正比，而在散列结构中，平均查找时间是一个与数据库大小无关的常数。

有序索引适合于如下形式的范围查询，

select $A_1, A_2, \ldots, A_n$

from r

where $A_i \leqslant c_2$ and $A_i \geqslant c_1$

，也就是要找出A_i的值在c_1和c_2之间的所有记录。假设文件已经按照属性A_i组织成索引顺序文件，用有序索引处理该查询时，首先根据索引查找值c_1对应的桶，找到后顺着索引中的指

针链顺序读取下一个桶，如此继续下去直到到达c_2。

反之，如果不用有序索引而用散列结构，可以首先查找c_1并确定其对应的桶，但却无法确定下一个需要检查的桶。一个好的散列函数会将A_i的值随机地分散到各桶中，因此，散列文件无法将各个桶按A_i的值顺序串接在在一起。查找到c_1对应的桶后，还需要根据 (c_1, c_2) 中的其它值搜索相应的桶，在桶中查找满足条件的所有纪录。

如果 DBS 设计者预先知道将来的 DBS 用户主要采用基于键值的等值查找，并且范围查询的使用频率很低，那么可以采用散列索引。否则，一般情况下有序索引使用的更多一些。

6.6 多健访问

前面提到的有序索引和散列技术只使用一个查找键组织文件和处理查询。但在实际应用中，经常涉及到多条件查询。例如对图 6.7 的商品 goods 文件，“查找价格大于 1000 元的家电类商品”，可用如下的 Select 语句实现：

```
select   名称
from     goods
where    类别=家电  and  单价=1000
```

假设 goods 文件分别在“类别”和“单价”上建有索引，处理这个查询有三种方法：

（1）使用“类别”索引将“家电”记录全部找到，检查每个找到的记录的单价是否等于 1000。

（2）使用“单价”索引，将“单价=1000”的记录全部找到，再检查每个找到的记录的类别是否为“家电”。

（3）使用“类别”索引，求出指向“家电”的记录指针；使用“单价”索引，求出指向“单价=1000” 的记录指针；然后求出这两个指针集合的交集，根据交集内的指找到所需记录。

这三种方法的缺点在于：如果“家电”类记录很多，“单价>1000”记录也很多，但同时满足 2 个条件的纪录却很少，那么都要扫描大量的指针，而最后却只求出少量纪录。

为解决此问题，可直接以<类别，单价>作为查找键，在goods文件上建立索引。这种索引与前面几种索引技术没有本质差别，只是查找键的属性个数多一些而已。查找键之间的比较按字典顺序进行，例如 $(a_1, b_1) < (a_2, b_2)$ 是在 $a_1 < a_2$ 或 $a_1=a_2$且$b_1<b_2$ 的情况下才成立。但这种多属性组成的有序索引处理如下非等值查询时有一些不足：

```
select   名称
from     goods
where    类别=家电  and  单价< 1000
```

，因为非等值“单价< 1000”查询涉及的纪录较多，很可能分布在不同的磁盘上，上述查询需要较多的 I/O 操作。

网格文件（Grid files）、位图索引（Bitmap index）和分区散列（Partitioned hashing）是三种数据库系统中常用的多属性索引技术，可有效解决多健访问问题。限于篇幅，不再赘述。

6.7 数据字典

6.7.1 元数据与数据字典

数据库管理系统 DBMS 中有一个称为数据字典（Data dictionary）的小型数据库，其中

存储了数据库数据对象的各类描述信息和数据库管理系统所需的控制信息，这两类信息就是所谓的数据库元数据（Metadata）。数据字典也称为系统目录（System. catalog）。

数据对象的描述信息包括数据的外模式、概念模式、内模式以及它们之间的映像的描述。数据库管理系统需要的控制信息包括查询优化、安全性检查、用户权限验证、事务处理、报告生成、约束验证、数据定义和操纵语言编译等系统程序模块所需要的信息。

6.7.2 数据字典内容

数据字典中的元数据为 DBMS 功能的正常实现提供了必需的基础数据。以关系数据库系统为例，关系 DBMS 的数据字典存储的信息主要包括：

（1）关系模式信息，如关系名、属性名、属性数据类型、键属性等。

（2）与视图描述有关的信息，如视图的定义信息。

（3）关系的存储结构和存取方法信息（如索引结构）。

（4）完整性约束信息，如主键、外键、空值/非空值、属性取值范围约束等。

（5）安全性有关的信息，如各类用户对数据库的操作权限和授权规则。

（6）数据库运行统计信息，如每个基本关系中元组的数目和各个属性的平均访问次数、索引的层次数等。这些信息由 DBMS 经常更新，反映了数据库的使用状况。

在关系 DBMS 中，数据字典被组织成关系表。DBMS 可以通过关系操作对数据字典中的元数据进行查询、修改和维护。而用户一般只能执行查询操作，例如可用 SQL 语句查询某个数据字典表的内容，但不能进行修改。

将数据字典作为普通关系处理的优点在于：

（1）可以利用 DBMS 的各种机制统一管理数据字典，避免设计新的数据字典管理机制。

（2）DBMS 和被授权的用户可以随时方便地利用系统提供的查询语言存取数据字典中的元数据。

6.7.3 数据字典的使用

数据字典是数据库管理系统的重要组成部分。数据字典中的信息既可供 DBMS 的子系统使用，也可供数据库设计者、数据库管理员 DBA 和一般用户使用。例如，DBA 可利用它来监视 DBMS 的使用情况，并协助用户完成有关管理工作，一般用户可通过数据字典查阅某个字段出现在哪个文件中等信息。

DBS 运行时，DBMS 各个子系统需要频繁使用数据字典，具体如下。

（1）数据定义语言 DDL 编译模块

DDL 编译模块编译处理由 DDL 编写的数据库三级模式定义和模式间映射，将这些模式描述信息存储到数据字典。

（2）数据操纵语言 DML 预处理模块

DML 预处理模块接收用 DML 书写的各种数据库查询和维护语句，进行语法和有效性检查。有效性检查主要是搜索数据字典，核对操纵语句中引用的数据库模式是否已经定义，检查操纵语句的有效性。

例如，关系数据库系统的数据操纵语言预处理模块接收到一个查询语句后，首先进行语法检查，然后在数据字典中查找语句中引用的关系和属性的信息，确定语句是否有效，如关系名和属性名是否在数据字典中存在，检查语句的语义正确性。DML 预处理模块还要根据数据字典中有关安全性的信息，检查语句的合法性

（3）数据操纵语言编译模块

DML 编译模块根据存储在数据字典中的概念模式与内模式文件结构之间的映射信息，将用 DML 编写的高级数据库查询或更新等语句变换为低级文件存取命令。这些信息

包括文件结构说明信息、文件记录域与概念模式的属性之间的对应、记录域的开始与终止字节位置等。

此外，当数据操纵语句的操作对象是外模式或视图时，DML 编译模块可以根据数据字典中有关外模式或视图的描述信息，进行外模式到概念模式的转换。

（4）查询优化模块

查询优化模块根据数据字典中的关系存取路径和关系操作实现等信息，确定如何产生一个优化的查询执行计划。例如，查询优化模块通过查询数据字典，了解关系的大小、物理存储结构、是否建立了索引或散列等信息，进而确定怎样执行一个定义在关系上的选择或连接操作和这些操作的执行顺序。

（5）授权和安全性验证模块

DBA 可以通过一些特权命令管理和更新数据字典中有关授权和安全性的信息。当用户对数据库进行查询或其它操作时，授权和安全性验证模块将存取数据字典，验证该用户是否有相应的操作权限。

6.7.4 数据字典系统

鉴于数据字典的重要性，许多大型数据库系统将数据字典从 DBMS 中分离出来，组成独立的数据字典系统，使之成为一个比 DBMS 更高级的用户与系统之间的接口，对元数据的访问均通过数据字典系统。

作为管理元数据的软件子系统，数据字典系统应具有以下三类基本功能：

（1）系统维护。实现对元数据的增添、删除和修改。

（2）控制。对元数据访问进行控制，保证元数据的完整性、安全性和一致性。

（3）信息提供。向各类用户，包括 DBMS 软件模块、数据库应用系统用户、应用程序员、系统分析员和数据库管理员等，提供所需的元数据信息。

一个理想的数据字典系统是整个数据库环境的一个完整部分，它在数据库应用系统的生命周期内起着重要作用，是一个能够在数据库设计、实现、运行和维护阶段控制和管理有关数据信息的重要工具。

6.8 数据库物理设计

6.8.1 设计步骤和内容

数据库应用系统物理设计包括数据库物理结构设计、数据库事务详细设计和应用程序详细设计。关于数据库事务详细设计和应用程序详细设计的具体要求和内容将在第 7 章中论述。本节集中讨论数据库物理结构设计。

数据库物理结构设计又称为物理数据库设计或数据库物理设计，是在具体的硬件环境、操作系统和 DBMS 约束条件下，根据逻辑数据库设计结果，设计合适的数据库物理组织结构。其目标是设计存储空间占用少、数据访问效率高和维护代价低的数据库物理模式。

根据 1.3 节所述，数据库设计是根据数据模型所规定的数据组织与管理规范，从视图级、逻辑级和物理级三个层次设计数据库的组织结构。两个比较有名的物理数据模型是一体化模型（Unifying model）和框架存储器模型（Frame memory model）。但与 E-R 模型和关系模型相比，数据库物理设计却很少直接使用物理数据模型，其原因在于：物理数据库设计着眼于数据库底层的物理存储与存取，与 DBS 所依赖的硬件环境、操作系统和 DBMS 密切相关。开发一个实际 DBAS 时，硬件平台、操作系统和 DBMS 必须根据需要事先选定，无需用户

自己设计开发。一旦选定硬件平台、操作系统和 DBMS，数据库的数据存储和存取方式等可用物理模式也就随之确定了。例如，硬件平台采用何种 RAID 存储设备、操作系统提供了什么样的文件管理机制、DBMS 可以支持哪几类索引机制等都属物理模式范畴。数据库物理设计所要做的是从这些可用物理模式中，根据实际情况为应用数据库选择、配置相应的物理数据组织机制。实际上，选定的硬件平台、操作系统和 DBMS 可以看作物理模型的具体实现。

此外，需要说明的 2 点是（1）鉴于目前绝大部分数据库系统都是关系数据库系统，下面所讲述的数据库物理设计内容都是面向关系数据库系统的。（2）数据库物理设计高度依赖于具体的 DBMS 和操作系统，本节从一般原理性角度讲述数据库物理设计的主要步骤和内容。不同 DBMS 和操作系统在具体的数据组织机制各有不同，因此设计一个实际数据库时，需要参考具体 DBMS 和操作系统的相关技术资料。

数据库物理设计主要包括以下步骤，每个设计步骤中包括若干设计内容。

（1）数据库逻辑模式转换。

将 DB 逻辑设计阶段得到的逻辑模式转换为 DBAS 所选定的具体 DBMS 所支持的基本关系表，设计关系表的完整性约束，定义相应的用户视图。

（2）DB 文件组织和存取方法设计。

根据 DBMS 和操作系统所支持的文件组织、文件结构和文件存取模式，为数据库文件设计合理的数据存储结构和与之对应的数据存取方法；特别是根据实际情况，为 DB 文件设计索引字段和索引类型，以提高文件访问速度。

（3）数据分布设计。

根据数据类型、作用和使用频率的不同，将数据库中应用数据、索引、日志、备份数据等不同类型数据合理安排在磁盘、磁带等不同存储介质中；针对分布式数据库系统，根据实际需要，对应用数据设计合理的数据副本，通过水平划分或垂直划分分割成不同的数据片断，然后分布式存储在各局部数据库中，以提高系统的数据访问效率和数据可靠性；设计关系模式中派生属性对应数据的存储方式，合理调整关系模式的规范化程度，以便在数据存储代价和数据访问效率之间作合理的权衡折衷。

（4）安全模式设计。

利用用户名/口令等机制设计 DBS 的系统安全方案；通过数据库系统视图机制和授权机制为用户分配对数据库对象的访问权限。

（5）确定系统配置

根据应用环境和上述物理设计结果，合理设置和调整 DBMS 和操作系统的存储分配参数，提供系统软硬件平台的初始配置信息。

（6）物理模式评估。

对物理数据库设计结果从时间、空间、维护代价等方面进行评估，从多种可行方案中选择合理的数据库物理结构。

上述设计结果应形成数据库物理设计文档，供下一阶段的 DBAS 开发步骤（即数据库实现与部署）使用。在本步骤中，设计人员可以根据设计出的数据库物理模式，形成数据库建库 SQL 语句脚本，提供给数据库系统实现人员。例如，数据库物理设计人员可以用分析设计工具 PowerDesign 生成面向 Oracle、Sybase、DB2、SQL Server、SQL Anywhere 等 30 多种具体 DBMS 的物理模型，并生成数据库对象的 SQL 语句建库脚本。

6.8.2 数据库逻辑模式转换

数据库逻辑设计创建了数据库的逻辑结构，其中有数据库的关系模式、关系模式上的完整性约束和关系表业务规则。关系模式定义了关系名称、属性名称和类型、属性域数据类型

等，完整性约束给出了主键约束、值域约束、空值约束、参照完整性约束等面向模型的语义约束，而业务规则则明确了面向具体应用的语义约束。这些结构信息都是独立于 DBAS 所采用的具体目标 DBMS 平台的。

物理数据库设计首先需要根据上述数据库逻辑结构信息，设计目标 DBMS 平台可支持的关系表（这里称为**基本表**）的模式信息，这些模式信息代表了所要开发的具体目标数据库的结构。这个过程称为数据库逻辑模式转换，主要包括以下设计内容。

1．实现目标数据库基本表和视图

这一步工作是采用目标 DBMS 所支持的建表方法，设计基本表及其面向模型的完整性约束。

不同的关系 DBMS 提供的建立数据库方法各有差别。例如，SQL Server2000 采用的 T-SQL 语言在语法结构、所支持的数据类型等方面有不同于标准 SQL 语言的自身特点。因此，在 SQL Server2000 平台下采用 Create Table 语句定义基本表时必须遵循 T-SQL 语法规则。

例如，对于 2.3 节定义的“商品”关系模式，可以给出以下 T-SQL 建表语句。

```
CREATE TABLE Table_Goods
(
 GoodsID              varchar(8)       NOT NULL ,
 GoodsClassID         varchar(8)       NOT NULL ,
 GoodsName            varchar(50)      NOT NULL ,
 ProductionDate       datetime         NOT NULL ,
 SaleUnitPrice        money            NOT NULL ,
TotalStorage          decimal(10,3)    NOT NULL ,
 Description          varchar(80)      NULL        ,
 FOREIGN KEY (GoodsClassID) REFERENCES Table_GoodsClass (GoodsClassID),
 PRIMARY KEY (GoodsID)
)
GO
```

在 SQL Server2000 平台下，还可以利用企业管理器交互式、图形化地创建基本表。

如果目标 DBMS 不完全符合 SQL 标准，不直接支持关系模式定义的某些数据特征，必须采取某种折衷处理，以保证关系模式的各种特征能正确映射到基本表中。例如，Microsoft Access 97 不完全符合 SQL92 标准，建表语句中没有明确的 FOREIGN KEY 子句，无法直接实现“商品”基本表 Table_Goods 与“商品类别”基本表 Table_GoodsClass 间的参照完整性约束。但是 Access 提供了创建 2 个表间的关系的方法，利用这种方法可以间接地定义“商品”基本表的外健。

对于在数据库逻辑设计阶段定义的面向各类 DBS 用户的用户视图，此时可以利用目标 DBMS 提供的视图定义机制，如 Create View 语句，创建目标数据库中的用户视图。

2．设计基本表业务规则

上述 CREATE TABLE Table_Goods 例子中，利用 Create Table 语句只定义了主键、外键、值域、空值等较为简单的、面向模型的完整性约束。在数据库逻辑设计阶段，已经面向各关系模式设计了面向应用的完整性约束，也就是关系表业务规则。进一步地，在数据库物理设计阶段，可以依据这些关系表业务规则，利用目标 DBMS 提供的 check 子句、断言、触发器等三种完整性控制机制，设计基本表应遵守的面向应用的完整性约束，也就是基本表业务规则。下面举例说明。

假设对“顾客”基本表 Table_Customer(customerID, cardID, name, sex, age, identitycard, address, postcode, tel)，有如下具体约束（1）性别只能是“男”或“女”（2）年龄范围在 5

到 85 岁之间。可以定义下述 SQL 语言 check 子句并加入到建表语句 CREATE TABLE Table_Customer 中。

constraints *sex-age-test*
check (*sex* **in** (‘男’，’女’))
check (*age* **between** [5, 85])

根据商场商品销售实际情况，商场销售出去的商品一定是商场库存中已有的商品。这就要求：每张销售单据明细表 Table_SaleBillDetail 中出现的“商品编号”GoodsID 一定已经出现在库存表 Table_Stock 中。可以用下面的 SQL 断言语句定义这 2 张基本表之间的完整性约束关系：

create assertion *GoodsID-constraints* **check**
(**not exists** ((**select** *GoodsID*
from *Table_SaleBillDetail*)
except
(**select** *GoodsID*
from *Table_Stock*)
))

当用户修改定义有 check 子句约束或断言约束的基本表数据时，DBMS 会自动检查约束关系是否得到保证。如果约束被违反，系统会自动报错。

某些复杂的业务规则不仅需要检查完整性约束是否得到满足，还要求约束违反时自动采取补救措施，恢复数据库的正确性。这类业务规则可用触发器来实现。

触发器是一种基于 event-condition-action model 的完整性控制机制。对于一个定义在某个或某些关系表上的触发器，当发生某些事件时（如修改、插入、删除关系表数据），如果满足一定条件（一般是通过完整性检查发现完整性约束被违反），触发器会执行相应的补救动作使数据库恢复到一致正确的状态。

SQL Server2000 数据库中, 触发器是一种特殊类型的存储过程，当表中数据被修改时，SQL Server 自动执行触发器，使用触发器可以实施更加复杂的数据完整性约束。由于声明参照完整性约束不能参照其它数据库中的对象，对于跨数据库的参照完整性约束只能通过触发器实现。触发器不允许带参数，也不能被直接调用，而只能由系统自动触发调用。

6.8.3 DB 文件组织和存取路径设计

1. 分析事务的数据访问特性

一个好的数据库物理模式应当满足存储空间占用少、数据访问效率高和维护代价低的要求。为了进行有效的数据库文件组织和存取路径设计，必须分析和理解数据库事务的数据访问特性。事务分析可以按照如下步骤进行：

（1）使用事务-基本表交叉引用矩阵, 分析系统内（部分重要的）数据库事务对各个基本表的访问情况，确定事务访问了哪些基本表、对这些基本表执行了何种操作，并进一步分析各操作涉及到的基本表属性。

图 6.31 给出了 1 个事务-基本表交叉引用矩阵的例子。其中事务 1 对基本表 1 和基本表 3 执行了插入和更新操作，对基本表 2 执行读操作。

	基本表 1				基本表 2				基本表 3			
	I	R	U	D	I	R	U	D	I	R	U	D
事务 1	*		*			*			*		*	
事务 2		*		*	*			*		*		
事务 3	*					*	*		*			
事务 4			*		*			*		*		*

注： I—插入 R—查询 U—更新 D—删除

图 6.31 事务-基本表交叉引用矩阵

（2）估计各事务的执行频率，即单位时间内事务的执行次数。在此基础上分析事务中的每个数据访问操作对各个基本表中的相关属性的操作频率。

例如事务 1 每天约运行 100 次，对基本表 1 的更新操作的频率为每天约 40 次，更新操作作用于基本表 1 中的属性 A。

（3）对每张基本表，汇总所有作用于该表上的各事务的操作频率信息，得到如下数据访问估计信息：该表是否被频繁访问、该表中哪些属性列的访问频率较高和作用于这些属性上的操作类型和查询条件类型（等值查询、范围查询）。

根据事务数据访问特性分析结果，可以对基本表设计更为有效的文件组织和索引方式。

2. 确定数据库文件组织结构

确定数据库文件组织结构是指为各种数据库对象选择合适的文件组织方式，它包括如下几方面内容：1）将数据库、基本表、视图等数据对象映射为操作系统文件；2）将基本表的元组映射为文件纪录，并确定纪录格式；3）在外设磁盘数据块上安排组织文件纪录。

一般来讲，数据库中的每个基本表可以存储在 1 个或多个文件中，1 个文件也可以存放多个基本表数据。视图的数据不单独存储，而是存放在视图所依赖的基本表文件中；基本表的元组与文件记录一一对应，元组的属性与记录域相对应。但记录既包含元组中的应用数据，也可以包含物理存储格式等描述信息。文件记录可以是定长或变长记录；文件可以组织为堆文件、顺序文件、聚集文件、散列文件和索引文件等形式。文件纪录存储在磁盘数据块中，一个数据块可以存放多个记录。

每种 DBMS 平台都提供一种或若干种数据库文件组织机制。当目标 DBMS 确定之后，数据库设计者所要做的工作就是根据应用系统实际使用情况，从目标 DBMS 提供的机制中，为数据库和基本表选择一种合适的文件组织结构。

以 SQL Server 2000 数据库系统为例，说明 DBMS 平台提供的文件组织机制：

（1）如 6.2.2 所述，在 SQL Server 2000 中，一个包含多个基本表的数据库对应的数据库文件有：一个主数据文件、零个或多个次数据文件、一个或多个日志文件。主数据文件主要用于存储用户数据，次数据文件除了存储用户数据外，还可以存储索引数据。

（2）数据库文件划分为多个大小为 8KB 的逻辑页，每个逻辑页又划分为多个槽，每个槽存放一个文件记录，每个文件记录对应一个基本表中的数据行（也就是基本表元组）。根据数据行是定长或变长，逻辑页中槽可以是固定长度，也可以是可变长度。逻辑页中除了存放基本表数据行外，还包括位数组或槽目录等存储格式描述信息。

（3）SQL Server 2000 采用 2 种文件组织方式：堆文件组织和 B-树文件组织。SQL Server2000 以**盘区**为单位，为数据库文件分配磁盘物理存储空间。一个盘区的大小为 64KB，占用 8 个连续页，可存放多个槽/记录。盘区相当于前面所说的磁盘数据块。对堆文件和 B-树文件，SQL Server2000 利用索引分配映射 IAM 记录了文件的每个逻辑页与该页所在物理盘区间的

对应关系，确定了每个页中的文件纪录/数据行的物理存储地址，实现了文件逻辑地址到物理地址的映射。

当基于 SQL Server 2000 平台设计物理数据库时，首先应当考虑为目标数据库设计合适的数据库文件。除了一个必需的主数据文件外，还可以设置多个次数据文件或日志文件，并指定各文件的磁盘存储位置；对上述三种数据库文件，还可以在 Create Database 语句中指定每个文件的大小、文件增长方式，从而规定了目标数据库的体积；确定了数据库文件之后，接着将基本表映射到主数据文件或次数据文件中，这可以利用 Create Table 语句来实现。

例如，可以利用 Create Database 语句分别在 3 个磁盘驱动器上创建 3 个次数据文件，并将这 3 个文件指派到同一文件组 filegroup 中。然后利用 Create Table 语句在 filegroup 上创建一个基本表 table-A，table-A 的数据被分布到 3 个次数据文件中，也就是分布到了 3 个物理磁盘上。对 table-A 的访问可以在 3 个磁盘驱动器上高效并行执行。

从文件组织角度，如果 SQL Server 数据库中的一个基本表的数据量很少，并且数据插入、删除、更新等操作非常频繁，该基本表不妨采用堆文件组织方式。因为堆文件无需建立索引，维护代价非常低。虽然堆文件只支持效率较低的文件顺序访问，但在文件数据量很少时，定位文件纪录的时间非常短。反之，对一个大数据量基本表，可以组织为聚集或非聚集 B-树文件，以提高文件查询效率。

下面给出一些为物理数据库选择合适的文件组织结构的原则：

（1）堆文件适合于下述情况：1）需要向基本表批量加载数据时。例如，新创建的表选择堆作为初始文件组织方式，向表中成批插入记录后，再重新确定调整文件组织方式，如改为 B^+-树文件组织；2）基本表的数据量很小。此时，采用顺序访问定位任何文件记录的时间都非常短，而且堆文件的维护代价很低；3）每当访问表时都要查询表中的每条记录，而且查询顺序可以任意。例如，检索“顾客”关系表中所有顾客的地址。

如果用户只需要访问表中的部分特定记录，堆文件是不合适的。

（2）顺序文件组织支持基于查找键的顺序访问，也支持快速的二分查找或插值查找。如果用户的查询条件定义在查找键上，则顺序文件是合适的文件组织方式。

（3）如果文件检索是基于散列域值的等值匹配，特别是如果访问顺序是随机的，散列文件组织是合适的。例如，对图 6.22 中的 goods 散列文件，根据“类别”属性进行等值查找。

但散列文件组织不适合于下述情况：1）基于散列域值的非精确匹配（模式匹配、范围查询）进行文件检索时。例如，在 goods 散列文件中，查找商品“类别”以“家”字开头的商品；2）基于非散列域进行检索时。例如，在 goods 散列文件中按商品“名称”进行查找；3）如果表中散列域的值经常被更新，DBMS 有可能需要利用散列函数为文件记录重新分配新的地址，导致调整纪录的物理存放位置，进而影响系统性能。

（4）B-树和 B^+-树文件都属于索引文件组织，适合于用户查询属于基于查找键的等值匹配、范围匹配、模式匹配和部分键匹配等应用场合。B-树和 B^+-树索引是动态的，可以随着数据文件的内容变化不断调整。当数据文件内容更新时，B-树和 B^+-树文件的性能不会恶化，是数据库系统中使用非常广泛的文件组织方式。

（5）聚集文件组织包括多表聚集和单表聚集 2 种形式，可以显著改善某些特定应用的查询效率。但这种机制有利有弊，有一定的适用场合，具体参见 6.3.3 节。

3. 设计存取路径

对数据库中的基本表等数据对象建立了合适的文件组织结构后，还需要为各个数据库文件设计存取路径。存取路径设计包括 2 个内容：（1）为数据库文件设计合理的物理存储位置。（2）为数据库文件设计索引机制，以提高文件存取速度。

设计数据库文件物理存储位置属于数据分布，将在 6.8.4 节中叙述。另外，上一小节

给出了一个在SQL Server2000 数据库中利用Create Database和Create Table语句创建数据库文件和关系表、并分布到不同磁盘上的例子。因此，下面集中讨论索引设计问题。

索引设计主要考虑在是否需要为基本表建立索引、选择表中哪些属性建立索引、建立什么样的索引。通过分析事务数据访问特性得到数据访问估计信息可以作为索引设计的一个重要依据。下面给出一些索引设计原则。

索引可以提高文件存取速度，改善大数据量文件的访问性能。但索引是由 DBMS 管理的，索引的建立、维护需要一定的时间和空间开销，对数据库中数据的插入、删除或修改可能引起索引的重新调整。因此，索引加快了数据查询速度，却减慢了数据更新速度。索引文件本身也会占用一定的存储空间。因此，必须权衡索引带来的益处和索引本身的实现维护代价，决定合适的索引机制。如果维护索引的代价超出加快查询所获得的好处，建立索引得不偿失。

可以根据下述原则决定是否为一个基本表建立索引：

（1）对于经常需要进行查询、连接、统计操作并且数据量大的基本表可以考虑建立索引；而对于经常执行插入、删除、修改操作或小数据量的基本表应尽量避免建立索引。

（2）一个基本表上可以建立多个索引，如 1 个聚集索引和多个非聚集索引。多个索引为用户提供了根据多个查找键快速访问文件的手段。但是索引越多，对表内数据更新时为维护索引所需的开销就越大。因此，在一个仅仅用于查询的基本表上建立多个索引是可取的。对于一个更新频繁的表则应少建索引。

（3）另外，索引可根据需要动态地创建或删除，以提高文件访问性能、适应不同操作要求。例如，对表进行大批量数据插入和更新时，可以先删除索引，以保证插入、删除、修改操作的系统响应时间，因为在数据插入或更新过程中维护索引需要花费很大代价。数据插入或更新完成后，再重建索引，以加快查询和统计等操作的系统响应时间。

可以根据下述原则决定一个基本表的哪些属性上应该建立索引：

（1）表的存取经常通过主键进行，因此可以在主键上建立索引。

（2）在 WHERE 查询子句中引用率较高的属性建立索引。

（3）如果属性参与了连接操作，则在该属性上建立索引后，文件纪录按序存放，有助于提高连接操作的执行效率。

（4）在 Order By 子句中出现属性上建立索引。

（5）在某一范围内频繁搜索的属性和按照排序顺序频繁检索的属性上建立索引。

（6）如果在 WHERE 子句中包含有一个关系的多个属性，可以考虑在这多个属性上建立多属性索引。对多属性索引，如果希望进行范围查询，应仔细安排属性在查找键中的次序。

相反，在具有下列特点的属性上最好不要建立索引：

（1）可为空值（Null）的属性、被函数引用的属性。

（2）使用了 GROUP BY 限定词的属性、查询中使用了限定词 DISTINCT 的属性。

（3）很少或从来不在查询中引用的属性。

（4）只有两个（如性别：男 / 女）或少数几个值的属性。

6.5.5 节讨论了有序索引和散列索引的各自特点。如果数据库文件需要频繁执行精确匹配查询（如等值查询），可以建立散列索引。而B^+-树等有序索引更适合于应用于范围查询的文件。

6.8.4 数据分布设计

1. 不同类型数据的物理分布

数据库中不仅有组织为基本表的应用数据，还有索引、日志、备份数据等其它类型数据。各种数据在系统中的作用不同，使用的频率也不一样，应当根据实际使用情况存放在合适的

物理介质上，以优化系统 I/O 性能。

数据库数据备份、日志文件备份数据用于故障恢复，使用频率低，而且数据量很大，可以组织为堆文件，存储在磁带中，对文件记录的访问采用顺序存取方式（Sequential access）。而应用数据、索引和日志则使用频繁，要求的响应时间短，必须放在支持直接存取（Direct access）的磁盘存储介质上。

当系统采用 RAID 等多磁盘存储系统时，可以将基本表和建立在表上的索引分别放在不同的磁盘上。访问基本表时，存放数据和存放索引的磁盘驱动器并行工作，可以得到较快的文件读写速度；类似道理，日志文件与数据库对象（表、索引等）也可以分别存放在不同磁盘上以改进系统 I/O 性能。

2. 应用数据的划分与分布

实际数据库中有可能存在一些非常大的基本表。这些表的属性列很多，表中的元组/数据行也非常多。为了改善对这类大基本表的访问性能，可以将基本表划分为若干分区，各分区数据分别存储在不同位置的磁盘上，并可以采用不同的物理组织方式。对基本表的划分可以依据不同原则：

（1）根据数据的使用特征划分。例如，一个基本表中有部分数据使用很频繁，主要是查询操作，并且要求响应速度快。可以将基本表分为频繁使用分区和非频繁使用分区，分别存储在 2 个磁盘上。频繁使用分区采用B^+-树索引以提高数据访问速度，由于数据访问主要是查询操作，因此对B^+-树索引的维护代价也很低。非频繁使用分区由于涉及数据更新操作，故采用常规线性索引，以降低索引维护代价。

从提高数据访问速度角度，可以将一个大的基本表划分为多个分区，每个分区作为 1 个数据文件分别存储在不同的磁盘上。对基本表的访问可以通过对分布在各磁盘上的数据文件的并行访问来实现。

（2）根据时间、地点划分。例如，对商品关系表 Table_Goods (GoodsID, GoodsClassID, GoodsName, ProductionDate, SaleUnitPrice, TotalStorage, Description) 可以按照商品的生产年份分成不同分区，生产年份相同的商品属于同一分区；或者按照商品生产厂商所在省份进行分区，产自于同一省份的商品归到同一分区中。

（3）DDBS 中的数据划分。分布式数据库系统 DDBS 提供了一种分布式的数据管理机制，具有成本低、数据可靠性高等优点。DDBS 采用水平划分或垂直划分的方法将基本表分为若干分区，根据需要为分区设计合适的副本，然后将各分区及其副本分布存储在系统中不同的站点上，不同的分区可以有公共数据。数据划分和分布的原则是数据应靠近其使用者，以降低查询过程中的数据传输代价，提高数据访问响应速度，并且通过副本冗余机制提高数据的可靠性。

水平划分是将一张基本表划分为多张具有相同属性、结构完全相同的子表，子表包含的元组是基本表中元组的子集。各子表可以有公共元组，也可以分别包括完全不同的元组。对商品关系表 Table_Goods 按照商品的生产年份进行划分就属于水平划分。

垂直划分是将一张基本表分解为多张子表，每张表包含的属性列较少。子表与原来的基本表的关系模式是不一样的。例如，基本表 Table_Goods 可以垂直分解为 2 张子表：

Table1_Goods (GoodsID, GoodsClassID, GoodsName, ProductionDate, SaleUnitPrice)

Table2_Goods (GoodsID, TotalStorage, Description)

3. 派生属性数据分布

基本表中的派生属性（derived attribute）是指该属性的取值可根据表中其它属性的取值唯一确定。例如，对前述“顾客”基本表增加属性“出生年月”，得到 Table_Customer(customerID, cardID, name, sex, age, birthdate, identitycard, address, postcode, tel)。age 为派生属性，它的取值可以根据属性 birthdate 的值推导出来。对带有派生属性的基本表可采用 2 种实现方式。

第一种方式将派生属性作为基本表内单独一列，称为派生列。图 6.32 (a) 中，派生列 age 占用了一定的存储空间，并且每当年份发生变化时，系统需重新根据当前年份和 birthdate 值更新 age 值。这种方式允许用户用 select 语句直接查询顾客的 age 属性。

第二种方式是派生属性不出现在基本表中，如图 6.32 (b)所示。这种方式节省了基本表的存储空间，并且当年份发生变化时无需不断更新派生属性 age 的值。但当用户查询某个顾客的 age 属性时必须自己构造如下的查询语句。

select *age* **as** *current-time – birthdate*
from *Table_Customer*
where *name* =John

在实际应用中，当派生属性与它所依赖的属性间的计算关系较为复杂，或当用户经常需要对派生属性进行查询，为提高查询效率，可以采用第一种方法，将派生属性的数据直接存储在关系表中，并且利用触发器等机制随时更新派生属性的值。

(customerID, name, sex, … birthdate, age)

customerID	name	sex	…	birthdate	age
1072	John	male	…	1973/02/11	35
1546	Mary	female	…	2000/12/04	7
2039	Jones	male	…	1982/06/10	25

(a)

customerID	name	sex	…	birthdate
1072	John	male	…	1973/02/11
1546	Mary	female	…	2000/12/04
2039	Jones	male	…	1982/06/10

(b)

图 6.32 派生属性数据分布

4. 关系模式的去规范化处理

在数据库逻辑设计阶段，由 E-R 图转换得到的关系模式必须经过规范化处理，以避免关系数据库使用过程中出现插入异常、删除异常、更新异常和数据冗余等问题，影响数据库的正确性。原则上，经过规范化处理后，静态关系至少要达到第一范式，动态关系至少要达到第三范式。

关系模式规范化过程实质上是将 1 个大的关系模式分解为一系列子模式，使得子模式可以满足更高一级的范式要求。例如，在商品零售经营管理系统中，有如下一个关于“商场”的关系模式：

Shop(shopID, street, city, province, zipCode, mgrStaff)，定义在该模式上的函数依赖 F={ShopID→street city province zipCode mgrStaffNo, zipcode→city province}。

模式Shop满足 2NF但不满足 3NF，因为函数依赖shopID→zipCode和zipcode→city province导致了非主属性city和province对主键shopID的传递函数依赖。根据 3NF分解算法，可以将Shop分解为 2 个 3NF子模式：

Shop (shopID, street, zipCode, mgrStaff)， Zipcode(zipcode, city, province)

通常情况下，当用户查询商场地址时，希望同时知道商场所在的 street、city 和 province。因此用户对商场地址的查询将涉及到 2 张表 Shop 和 Zipcode，例如可采用下面的查询语句：

select *street, city*, *province*
from *Shop, Zipcode*
where (*Shop.zipcode* = *Zipcode.zipcode*) **AND** (*mgrStaff*=wangjun)

该查询语句需要执行 Shop 和 Zipcode 表上的连接操作，查询执行代价较大。

如果在实际应用中，用户需要经常性地查询商场地址，并且 2NF 的 Shop 模式在使用过程中可以基本避免各种数据库访问异常，或虽然会导致访问异常，但处理异常的代价较低，那么对 2NF 的 Shop 可以不做规范化分解，直接作为一张 2NF 基本表存储在物理数据库中。这样，用户查询商场地址的操作只涉及一张关系表，避免了复杂费时的连接操作，大大降低了查询代价。

在物理数据库设计阶段，可以对考虑数据库中某些 3NF、BCNF 模式是否可以降低其规范化程度，例如降为 2NF 范式，以提高查询效率。这称为关系模式的去规范化处理。但不满足 3NF 的动态关系模式又可能导致数据库访问异常。因此，设计数据库系统的基本表时，需要在模式规范化与查询效率之间仔细折衷权衡。

去规范化的实质是通过降低关系模式的规范化程度，在关系表中引入受控冗余数据，提高数据查询速度。例如，在 2NF的关系表Shop(shopID, street, city, province, zipCode, mgrStaff)中，对处于同一省、城市的所有商店，它们在city,和province 2 个属性上的数据完全一样，并重复出现在关系表中，这种冗余数据有利于对关系表中商店地址的查询。

6.8.5 安全模式设计

1. 系统安全设计

数据库安全设计的目的是通过设计合理的安全控制机制，保证（1）拒绝非法用户对数据库的访问（2）允许合法用户在其访问权限范围之内访问数据库，阻止非法越界访问，也就是只允许合法用户访问其应该访问的那部分数据。

针对上述 2 个目的，数据库安全模式设计包括系统安全设计和数据安全设计。系统安全设计是指为数据库合法用户分配用户名和口令，使其能够正常登录数据库系统访问所需数据。借助用户名和口令，系统可以拒绝非法用户对数据库系统的访问和由此带来的可能的数据破坏。

随着信息安全技术的发展，系统安全保障手段也日渐丰富。除了用户名和口令这种常规控制方式之外，还可以采用基于 CA 认证的系统安全控制机制。此外，数据库的系统安全控制手段必须与数据库系统的整体安全级别（如 B 级、C 级）相匹配。

2. 数据安全设计

数据安全设计是指为目标数据库中的基本表和数据视图设计访问规则。

DBAS 系统需求分析已经明确了各类用户需要访问哪些数据和对这些数据的访问操作类型，数据库逻辑设计据此为不同用户设计了相应的独立于具体 DBMS 数据视图。物理数据库设计通过数据库逻辑模式转换，进一步将这些数据视图转换为目标数据库中可实现的数据视图。

一般来讲，一个目标数据库包括众多基本表，每个数据库用户并不需要访问数据库中全部数据，只是根据自己的需求访问其中的一部分，这部分数据组成了用户的数据视图。将用户访问范围限制在其用户视图上可以防止由合法用户访问造成的信息泄密。此外从数据独立性角度，用户最好不要直接访问基本表，而是间接地通过视图去访问基本表中的数据。一旦基本表结构发生变化（例如，基本表中增加了 1 个属性），由 DBA 或系统设计者通过重新调整视图与基本表间的对应关系，使得面向用户的视图仍然维持不变，从而保证了用户访问不受数据库结构变化的影响。

例如，对于商品库存表 Table_Stock（商品编号，库存量，库存描述信息，采购成本，利润），假设普通销售人员只允许访问前三项信息。可以定义数据视图 View_Stock（商品编号，库存量，库存描述信息），销售人员通过该视图获得他所需要的信息。当 Table_Stock 中增加新属性“税费”时，只要视图 View_Stock 结构保持不变，销售人员对数据库的访问

就不会受影响。

权限（Privilege）是允许用户对一给定的数据库对象（数据库、基本表、视图、存储过程、日志等）可执行的操作，如查询、插入、删除、修改、创建、备份等。设计基本表和数据视图访问规则是指根据用户需求，采用目标 DBMS 提供的数据访问授权机制，为用户分配对数据库对象的合法访问权限。

最常用的访问规则设计方法是使用 SQL 语言中的授权语句 Grant 和 Revoke。利用角色（Role）机制可以方便快捷地进行权限控制。此外，许多商用 DBMS 还提供了图形化、交互式的权限设计手段，如 SQL Server 中的企业管理器。

6.8.6 确定系统配置

1. 存储分配参数配置

数据库应用系统中的 DBMS 和操作系统产品提供了一些与存储分配有关的配置参数，供设计人员和 DBA 对数据库进行配置和优化。不同的 DBMS 和操作系统的存储分配参数各不相同，但一般包括：数据库配置参数，如同时使用数据库的用户数、同时打开数据库对象数、数据库大小；磁盘块使用参数；内存缓冲区参数，如缓冲区个数和大小；时间片大小；装填因子；锁的数目等。

这些配置参数都有一些建议缺省值，可以作为一般情况下系统的初始参数配置。但这些缺省值不一定适合每一种应用环境，数据库物理设计时，需要重新对这些变量赋值以改善系统的性能。

合理的参数设置可以提高数据库的存储和查询效率，不合理的参数不仅会严重影响数据库的存储和查询效率，甚至可能导致系统运行时的数据库崩溃。存储分配参数的种类和合理的参数取值与具体的 DBMS 操作系统平台密切相关，依赖于具体的应用场景。数据库系统设计人员需要参照 DBMS 和操作系统的技术资料，如用户手册、性能调优手册等，结合 DBAS 实际应用情况，为数据库系统设置合理的存储分配参数。详细的参数设置已超出本书范围。

物理数据库设计时设计人员对系统配置参数的设置和调整只是初步的。在系统运行时如果发现系统性能没有达到预期要求，而且原因与某些配置参数设置不当有关，那么需要 DBA 根据系统实际运行情况调整相关参数，以期改进系统性能。这属于数据库系统运行维护工作中的参数调整范畴。

2. 实例

在 Oracle 数据库中，最基本的数据单位是存储记录，存储记录既包含关系元组也包含物理存储格式描述信息。磁盘数据块是磁盘读写的基本单位，每个数据块可以存储多个记录，由一个或多个连续的磁盘数据块组成的存储空间称为数据域。与关系、索引、聚集等数据库对象相对应的存储结构称为数据段，由一个或多个数据域构成。数据段存储在数据库文件中，一个数据库文件对应一个操作系统文件。一个或多个数据库文件形成一个数据库分区。数据库分区是物理数据库的逻辑划分。

ORACLE 数据库的每个关系、索引或聚集都存储在一个数据段中。关系、索引和聚集的物理存储结构设计主要是数据段的设计，包括数据段存储空间的设计、构成数据段的数据域设计和数据域中的磁盘数据块设计。

数据域与数据段的设计主要是确定下列 4 个参数：数据段初始数据域容量 INITIAL-EXETENT、第二数据域容量 NEXT_EXETENT、数据段最小数据域数目 MINEXTENTS、从第三个数据域开始的每个数据域增长百分比 PCTINCREASE。

磁盘数据块的设计主要是确定 2 个参数：数据块内为存储记录的扩展而保留的空间百分比 PCTFREE、数据块中数据量不得低于的百分比 PCTUSED。如果某数据块的自由空间的百分比低于 PCTFREE，不允许再向此块中加入新记录；如果某数据块所存储的数据的百分

比低于 PCTUSED，允许向此块中加入新记录。可以根据以下原则决定这 2 个参数的取值。

（1）选择低 PCTFREE 可以使数据块中容纳较多的纪录，一个关系可以存储在较少的数据块中从而改善全关系扫描性能。对于静态关系可以选用较低的 PCTFREE 值。但低 PCTFREE 又会增加关系元组跨块存放的可能性，降低系统 I/O 效率；高 PCTFREE 减少了关系元组跨块存放概率，适合于经常扩展纪录大小的关系。但高 PCTFREE 又会降低全关系扫描性能并占用较多存储空间，因为关系占用的数据块较多。

（2）低 PCTUSED 值可减少关系元组跨块存放概率，但可能浪费存储空间。高 PCTUSED 值节省了存储空间，增加了跨块记录出现的可能性，降低了系统 I/O 效率。

选择 PCTFREE 和 PCTUSED 的值时要充分考虑上述因素，保证高系统效率和高存储空间利用率。由于存储空间的费用越来越低，在无法同时兼顾系统效率和存储利用率时，应该以提高系统效率为主。

下面再以 SQL Server2000 系统为例，说明操作系统参数配置。

假设 SQL Server2000 运行在 Windows NT 操作系统之上，需要在操作系统中配置系统虚拟内存的大小。如果虚拟内存配置过低，会影响系统中并发执行的进程数目；虚拟内存配置过大而实际物理内存并不大，又会引起进程执行过程中频繁的缺页中断，影响系统效率。可考虑将虚拟内存大小设置为系统实际物理内存的 1.5 倍。

6.8.7 内模式评估

前述各个步骤的设计结果定义了目标数据库的物理结构，组成数据库内模式（或物理模式）。在设计过程中，通过对时间效率、空间效率、维护代价和各种用户要求权衡考虑，可能会产生多个设计备选方案，因此数据库设计人员必须对这些方案进行仔细评估，从中选择一个较优的方案作为数据库的物理结构。

评价物理数据库的方法完全依赖于所选用的 DBMS，主要是从定量估算各种方案的存储空间、存取时间和维护代价入手，对估算结果进行权衡、比较，选择出一个较优的合理的物理结构。如果该结构不符合用户需求，则需要改进设计。

6.6 小结

计算机中的存储介质有高速缓存、内存、快擦写存储器、磁盘存储器、光盘存储器、磁带存储器等，这 6 种存储介质在可靠性、访问速度、容量、成本等方面各有特点。

数据库以文件形式存储在磁盘等外设存储介质中。数据库物理组织涉及到 DB 文件的组织、DB 文件的结构和 DB 文件的存取等几个方面。关系数据库将关系表映射为文件，元组表示为文件记录，文件的组织可以采取定长纪录格式或变长纪录格式。

文件的结构是指如何在外设存储介质上安排、存放文件的逻辑纪录，其核心是文件的逻辑纪录与其物理地址间的映射关系。常见的文件结构有堆文件（也称为无序文件）、顺序文件、聚集文件、索引文件组织（如索引顺序文件和B^+-树文件组织）和散列文件组织等。每一种文件结构都有与之对应的文件存取方法。为了提高文件查找速度，可以为文件建立索引机制。

广义上的索引技术包括有序索引和散列技术。根据索引文件本身不同特点，有序索引可以分为聚集索引和非聚集索引、主索引和辅索引、稠密索和稀疏索引、单层索引和多层索引等。聚集索引的查找键指定的顺序与数据文件中数据记录的排列顺序相一致。非聚集提供了按照多个查找键访问数据文件的方法，但通常会增加数据库修改的开销。如果聚集索引的查找键就是数据文件的主键，则聚集索引同时又是主索引。

索引顺序文件组织和B^+-树文件组织是数据库系统中常用的 2 类基于索引的文件组织。索引顺序文件既可以进行顺序访问，又可以根据聚集索引进行直接访问。索引顺序文件的主要缺陷是随着文件数据量的增大，文件访问性能会下降。为克服这一缺点，可以使用B^+-树索引和B-树索引。B^+-树和B-树索引都属于平衡树，树的查找简单有效，查找代价与树的高度成正比，但插入和删除操作比较复杂。

散列文件组织通过散列函数直接得到文件记录所在的物理地址（如桶地址）。静态散列采用固定桶地址集合，不容易适应数据库数据量的快速增长。可扩充散列结构属于允许修改散列函数的动态散列技术，可以通过桶的分裂或合并来适应数据库数据量的变化。散列方法还可以用于创建索引结构，将查找键值与指针组织成散列文件结构，得到散列索引。

网格文件、位图索引和分区散列属于多属性索引技术，可以有效支持涉及多条件查询的数据库多健访问。

数据字典是数据库管理系统 DBMS 的一个重要组成部分，用于存储有关数据库数据对象的描述信息和 DBMS 所需的控制信息，即数据库元数据。这些元数据可供 DBMS 的子系统、数据库设计者、DBA 和一般用户使用。

数据库物理设计由数据库逻辑模式转换、数据库文件组织和存取路径设计、数据分布设计、安全模式设计、确定系统配置和物理模式评估等步骤组成，每一步骤中包括若干设计内容。数据库物理设计与 DBAS 系统采用的具体硬件环境、操作系统和 DBMS 密切相关。数据库物理设计中的一项重要工作是为数据库文件建立合适的索引，以提高数据库访问效率。

习题

6.1 名词解释

(1) 文件组织，文件结构，文件存取

(2) 堆文件，顺序文件，散列文件，聚集文件

(3) 有序索引，散列索引

(4) 聚集索引，非聚集索引；主索引，辅索引；稠密索引，稀疏索引

(5) 查找键；索引顺序文件

(6) 静态散列，动态散列

6.2 从数据访问速度、成本、容量和可靠性等方面比较计算机系统中的 6 类存储介质。

6.3 设关系数据库中有两个关系：

Course (course_name, teacher)

Eneollment(course_name, student, grade)

，设有三门课程，五个学生，学生与课程间有选修联系。试用聚集文件表示这两个关系的文件结构。

6.4 设文件的查找键值集为｛2, 3, 5, 7, 11, 17, 19, 23, 29, 31｝，散列函数为 $h(x) = (x \bmod 8)$，每个桶可存储 3 个记录。试建立一个可扩充散列结构，并画出示意图。

6.5 为什么散列结构不适宜对查找键进行范围查询？

6.6 数据字典主要包括哪些元数据信息？。

6.7 数据字典的用途是什么？

6.8 简述物理数据库设计的主要步骤和设计内容。

6.9 简述数据库设计中建立索引的原则。