数 据 库 系 统 基 础

（上 集）

计算机工程与应用编辑部

前　言

电子计算机对现代科学技术的发展起着极大的推动作用。有人将电子计算机的应用比作第二次工业革命。现在计算机的应用已经非常广泛，它已经渗透到工农业生产、商业、行政管理、科学研究和工程技术的每一个角落里，成为这些部门不可缺少的有力工具。在整个计算机应用领域里，数据处理占着最大的比重，而数据库系统又是进行数据处理的核心机构。它的效能往往决定了整个计算机应用的经济效益。

数据库系统对于一个发展中的国家也是十分重要的。它能提高数据存取的效率，减少重复和浪费，使计算机更巧妙地被用来解决问题。使得数量较少而价格相对昂贵的计算机更能充分的用来为人类服务。

选用或设计高效能的数据库是一件十分慎重和细致的工作，即使在先进的工业化国家里也常见到这样的情形：由于盲目地选用不恰当的数据库会造成巨大的浪费。往往在花费了大量的劳动完成了大规模的软件工程之后，才发现它是不合应用要求的，这时再想更改其基本设计已经极其困难。所以无论是设计数据库系统还是选用已有的数据库系统，都一定要掌握数据库系统的基本原理，详尽了解它的设计特点及使用方法。同时也应该了解有关方面最新的研究成果，充分考虑这些成果对未来数据库系统的影响。只有这样，才可以避免不必要的损失，免蹈外人覆辙，使数据库发挥出最大的经济效益来。

这本书取材于我在美国普渡大学 (Purdue Univ.) 时编的两个课程的讲义："文件系统结构"和"数据库系统基础"。内容包括数据库系统设计和应用的基础知识。选材原则是"理论"与"实践"并重。尽量避免纯理论的探讨。中国科技大学研究生院计算机教研室部分教师和研究生读过我的讲义，他们觉得这本讲义的内容对目前中国数据库的研究和使用很有参考价值，因此决定编译成中文出版。经过几个月的努力，他们付出了巨大的劳动，他们辛勤刻苦，严肃认真，终于使本书得以和广大的中国读者见面。在此我向他们致谢。我衷心地希望，由于他们的努力，本书能对中国科学技术现代化做出有益的贡献。

姚诗斌
美国马利兰大学
(University of Maryland)
一九八一年四月七日

---

*注：参加本文编译的有：赵延光、邢俊英、李凯、邹真葆、程允怡，张文宽、李海宽、郑远新、周明陶．姚诗斌教授又亲自进行了校阅。

数据库系统基础

目　录

第一章　存贮器和文件结构概论

第一节　存贮技术概述

在存贮技术中，主要需要考虑的因素有以下三点：

1．存贮每位(bit)数据所花费的代价；

2．存取时间；

3．基本代价，即建立起一套存贮设备起码要花费的代价。

迄今，各种存贮技术如图 1-1 所示。三角形的顶部表示存贮时间最快，但每位代价较高；三角形底部，存贮时间最慢，但每位代价较低，因此可选为大量数据的存贮介质。目前顶部三层一般为半导体存贮器，底部二层一般为磁介质存贮器，而中部则是两者之间的"间隙"，其中：

FHD——固定磁头的磁盘，由于磁头固定，故存取时间快，不过磁头量大，价格较高；

CCD——电荷耦合存贮器件，将电荷存贮在半导体表面的电位井中，可控制其沿半导体表面移动，类似一个移动的寄存器。这种器件存取速度比磁泡快，但在停电情况下所存贮的信息将丢失(即可挥发性(volatility))。多用作为中间缓冲存贮器件。

MBM(Magnitic Bubble Memory)——磁泡存贮器，将信息存贮在微小磁性区域中（磁泡），磁泡可根据控制信号在磁性薄膜中移动。存贮的基本形式也是移位寄存器式。这种存贮器不具有挥发性。看来，这种存贮层次仍将维持一个相当长的时期，即 FHD、CCD、MBM 等仍将处于"间隙"的地位，即只少量做为中间存贮之用，而不会很快取代其他层次的存贮部件。在数据库技术中，目前数据存贮主要使用磁带和磁盘，因此下面将介绍一下磁带和磁盘的存贮技术。

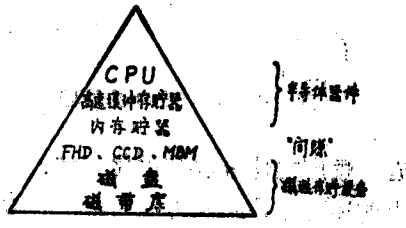


图 1-1　存贮技术层次示意图

第二节　磁带存贮技术

一、磁带的基本特性：

磁带发展始于五十年代早期，是顺序存贮设备，价格较便宜。目前所使用的典型的磁带尺寸为：厚×宽×长＝0.002 英寸×0.5 英寸×2400 英尺，分为七道和九道两种。七道是六位数据加一位奇偶校验位；九道是 8 位数据加一位奇偶校验位。常用的存贮密度为：

七　道	200　bpi 556　bpi	bpi (bytes per inch——每英寸存贮字节数)
九　道	800　bpi 1600　bpi	

磁带的存取时间主要花在将磁带转到所需的位置上，其最小存取时间约为 20 毫秒，最大可达 10 分钟。

磁带的运转速度一般为 120 英寸/秒左右，必要时还可以快进或快速回卷。但磁带的起动和停止都需要花费一定的时间，如图 1-2 所示。因此，在磁带上所存贮的数据记录之间必须空出一段间隙 (IRG:Inter-record Gap)，以

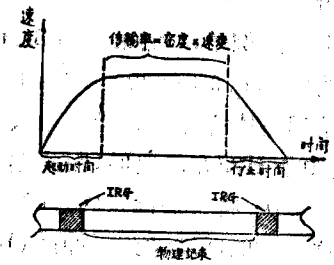


图 1-2　磁带的起动和停止及 IRG

适应起动时磁带达到正常速度和停止时的滞后。一般IRG＝0.6英寸。

二、磁带文件的组织方式：

可以有两种组织方式：

(1) 由定长的记录组成：每一块中的各个记录长度相等，因此块的大小可直接由B. F.(块因子)确定。

(2) 由不定长的记录组成：这时如果每一块仍具有同样的B. F.，则块的大小将不确定，缓冲区的大小应按最大的块来确定。如果确定块的大小不变，则每块中将可能有不同数量的记录数，即B. F. 不同。而且每块中还可能剩下一段不够存放一个记录的无用区(Padding)。如图1-3所示。

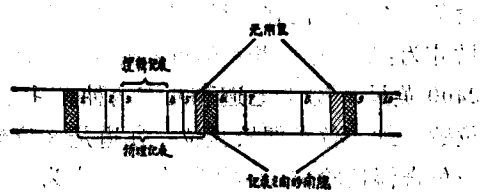


图 1-3 磁带文件的组织

三、分块方法(blocking)

举一个实例，存贮密度为800 bpi的磁带，假设每个记录有80个字节，共有1000个记录，则可计算出每个记录所占据的物理长度为0.1英寸，我们知道，加上IRG后的磁带总长度为700英寸，其中真正用于记录的长度只有100英寸。

为了减少IRG所造成的浪费，可以采用组块方法进行存贮，即将每个逻辑记录组合成比较大的物理记录(块)，只在块之间留IRG。每个块中的逻辑记录的数目称为块因子(Blocking factor即B. F.)，

对于上面的例子，如果采用B. F.＝100，则总长度为 $10\times0.6+1000\times0.1=106$ 英寸。

可见采用组块方法后，能大大减少所需的磁带长度。由于IRG在块之间，因此起动磁带后最好读取一整块。将一块读入内存时，在内存中暂时存放这一块的区域称为缓冲区(Buffer)，如果B. F. 较大，则要求的缓冲区也将较大。

四、磁带文件的处理方式：

由于磁带的存取一般较慢，物理存贮位置的顺序性强，因此对磁带文件的处理以成批(Batch)处理方式为好。这是由于在成批处理中，可能有多个对磁带文件的处理要求，这样在一次顺序读写中就可能处理多个处理要求，即所谓"命中率高"(higher hit rate)，可以节约存取时间。因此最好对这些处理要求按磁带文件的物理顺序进行分类(sort)，这样前面的处理用到前面的磁带文件，后面的处理用到后面的磁带文件，则将磁带顺序地从头一次读到尾即可以处理完所有的处理要求。但对这些处理要求进行分类是相当花时间的，这将成为主要处理代价。

为了形成成批处理，一般做法是专用一个处理磁带文件(Transaction file)，将平时遇到的对磁带文件的处理要求(修改、删除等)记入这个新的处理文件中去，而不直接修改旧的主文件(Old master file)。当需要读取原始主文件时，先读处理文件，看要读取的记录是否修改过。如修改过，则按修改后的内容读取；如未曾修改过，再回去读旧主文件。这样积累到一定时间(例如处理文件记满了)，做一次成批顺序处理，即根据处理文件的内容，从头到尾修改旧主文件的内容，将修改后的内容记入一个新的主文件(New master file)，以后就用这个新的文件作为原始文件。如此重复，如图1-4所示。

旧的主文件和旧的处理文件，如有必要可以保留，以便用以检查处理的正确性，或必要时恢复到处理前的内容。

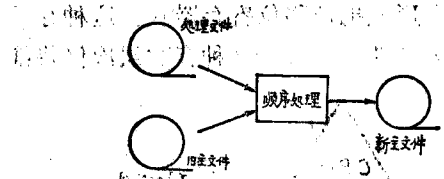


图 1-4 磁带文件的处理

五、顺序文件分块的优化方法：

由前述可见，组块方法可以减少存取时间和所需的存贮空间，而大的B. F. 要求大的

缓冲区。现在考虑给定一个缓冲区大小，问最优的 B. F 是多少？

假定：给定N个顺序文件(即N个磁带)，对于每个文件 i，$i=1,2,\cdots N$

R_i——总记录数；

C_i——每个记录的长度(字符数)；

B_i——B. F.；

$I_i=R_i\ C_i$　　文件 i 中的字符数；

$M_i=\dfrac{R_i}{B_i}$　　文件 i 中的总块数；

$S_i=B_iC_i$　　文件 i 中块的大小(字符数)；

D_i——为文件 i 所准备的缓冲区数目；

T_i——数据传输率(K字符/秒)；

ρ_i——存贮密度(bpi)；

L_i——IRG 的长度(字符数)；

G_i——磁带的起动/停止时间 (ms) (或寻找时间 (seek time) + 等待时间 (latency)；

S——最大总缓冲区的大小。

如图 1-5 所示。

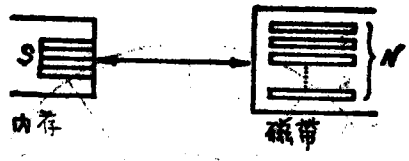


图 1-5　N 个顺序文件的组块问题

可见，$B=\sum_{i=1}^{N} G_i\ M_i$ 为总存取时间 ①

希望选择 B_i，而使得 B 最小，同时满足：

$$S\geqslant\sum_{i=1}^{N} D_i\ S_i \quad \text{(即缓冲区大小足够大)} \quad ②$$

由式②，取等号，我们有

$$S-\sum_{i=1}^{N} D_i\ B_i\ C_i=0 \qquad ③$$

利用拉格郎日(Lagrange)乘数 λ，则问题变为：

$$B'=\sum_{i=1}^{N}\frac{G_iR_i}{B_i}-\lambda\left(S-\sum_{i=1}^{N} D_iB_iC_i\right)$$

对 B_i 求偏导数并令其为 0，则有：

$$\frac{\partial B'}{\partial B_i}=-\frac{G_iR_i}{B_i^2}+\lambda D_iC_i=0$$

$$\therefore B_i=\sqrt{\frac{G_iR_i}{\lambda D_iC_i}} \qquad ④$$

将④代入③：

$$S=\sum_{i=1}^{N} D_i\sqrt{\frac{G_iR_i}{\lambda D_iC_i}}C_i=\frac{1}{\sqrt{\lambda}}\sum_{i=1}^{N}\sqrt{G_iR_iD_iC_i}$$

$$\therefore\sqrt{\lambda}=\frac{1}{S}\sum_{i=1}^{N}\sqrt{G_iR_iD_iC_i} \qquad ⑤$$

将⑤代回④，得：

$$B_i=\frac{S\sqrt{\dfrac{G_iR_i}{C_iD_i}}}{\sum_{i=1}^{N}\sqrt{G_iR_iD_iC_i}} \qquad ⑥$$

即所求最优的块因子，正比于$\sqrt{\dfrac{C_iR_i}{C_iD_i}}$。

这时总的存取时间为：

$$B=\sum_{i=1}^{N} G_iM_i=\sum_{i=1}^{N}\frac{G_iR_i}{B_i}=$$

$$\sum_{i=1}^{N}\frac{G_iR_i\sum_{i=1}^{N}\sqrt{G_iR_iD_iC_i}}{S\sqrt{\dfrac{G_iR_i}{C_iD_i}}}$$

$$=\frac{1}{S}\left(\sum_{i=1}^{N}\sqrt{G_iR_iD_iC_i}\right)^2$$

但须注意，当对一个磁带文件设有多个缓冲区时，由于 CPU 的处理要快得多，因此当读入最后一个缓冲区时，第一个缓冲区数据可能已由 CPU 处理完毕，这样就可以将磁带文件连续不断地读入缓冲区。在这种情况下，G_i 就不再是磁带起动/停止时间了，磁带实际上只在开始时起动一次就一直运转下去，因此这时

$$G_i=\frac{L_i}{T_i}\text{(IRG 的传输时间)}$$

显然，还需要考虑两条附加的限制：

1．对于可变长度的记录，块的大小应比最长的记录还要长些：

$$M_i' \leqslant S_i$$

2．块的大小不能比整个磁道还要大：

$$S_i \leqslant W_i$$

如果由式⑥得出的 B_i 不能满足上述限制条件，则根据限制条件加以修改后，再重新计算。

下面举一个例子。假定在下面的给定条件下：

$R_1=600,000$　$C_1=15$　$T=40$ kc

$S=50,000$

$R_2=1,000,000$　$C_2=100$　$\rho=800$ bpi

$D_i=1$

$R_3=1,000,000$　$C_3=100$　$L=500$ c

$R_4=40,000$　$C_4=100$　$G=20$ ms

① 如果不采取组块方法：

	B_i	S_i	M_i	文件长度
文件 1	1	15	600,000	14 卷（2400 英尺/卷）
文件 2	1	100	1,000,000	27 卷
文件 3	1	100	1,000,000	27 卷
文件 4	1	100	40,000	1 卷
合　计		315	2,640,000	69 卷

总存取时间＝2640000×20 ms＝14 小时 40 分

② 如采取最优组块方法：

	B_i	S_i	M_i	文件长度
文　件　1	400	6000	1,500	1 卷
文　件　2	200	20,000	5,000	5 卷
文　件　3	200	20,000	5,000	5 卷
文　件　4	40	4,000	1,000	1 卷
合　　计		50,000	12,500	12 卷

总存取时间＝12500×2ms＝4.17 分

第三节　磁盘存贮技术

一、磁盘的基本特性

目前多使用带有可移动式磁头的磁盘（活动臂磁盘）。如图 1-6 所示。常用盘组为 11 片式，沿一个固定方向高速旋转，除最上一面和

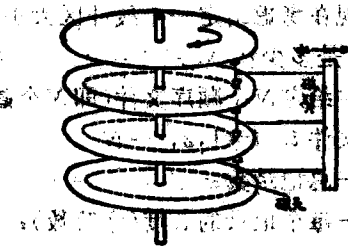


图 1-6　磁盘结构示意图

最下一面不用外，共有 20 个记录面用于贮存数据，每个记录面有一个读/写磁头，所有的读/写磁头是固定在一起同时移动的。在一个记录面上读/写磁头的轨迹称为磁道（Track），在同一磁头位置下各个磁道的总和称为同位标磁道组或"柱面"（Cylinder），在一个磁道内又可分成若干个扇面（Sector），如图 1-7 所示。

以常见的 IBM 2314 型磁盘为例，其参数为：

20 记录面/磁盘组，200 磁道/记录面，7294 字节/磁道。

因此整个磁盘组的总存贮容量为 80 兆字节。

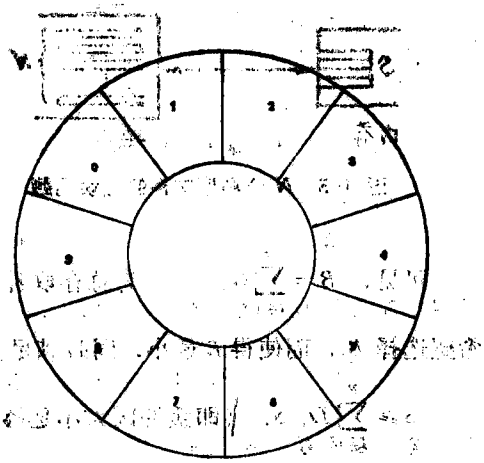


图 1-7　磁盘的扇面示意图

磁盘的特点是：

① 每位存贮代价低廉；

② 技术较成熟，这是由于对涉及到的物理现象了解较透彻，适于制造磁盘的材料较多，在存贮密度上的最后限制还远未达到。（可达几千 M 字节）；

表 1-1 磁盘技术的进步(以IBM 产品为例)

时 间	型 号	存贮量 (MB)	存贮密度 (bits/英寸²)	价 格 ($/MB)	磁头压力/磁头重量	磁头定位方式	磁头高度	平均查找时间
1956	350	5	2K	153				
1965	2314	29.1	220K	23	350 克/3.25 克	液 压	100 微英寸	12 ms
1970	3330-1	100	775K	8.30	350 克/5 克	音 圈	45 微英寸	6 ms
1973	3380-11	200	1495K	4.85				
1975	3350	317	3058K	2.25	10 克/0.25 克接触式	音 圈	20 微英寸	6 ms
1979	3370	571	7500K					

③ 对环境条件的要求不严格；

④ 可靠性较高；

表 1-1 列出了某些磁盘技术进步的情况。

二、磁盘的存取时间：

① 当有多个磁盘组时，选定某个磁盘组是由电子线路实现的，因此可以很快。

② 选定某个磁盘组后，首先要确定数据所在的同位标磁道组，这就需要将磁头移到所需的位置，通常这个动作是机械的，因此较慢，一般此称为"寻找时间"(seek time)，视设备的不同而不同，当磁头所需移动的距离越大时，所需的寻找时间越长。

③ 下一步需确定数据所在的记录面，即选定哪一个磁头读/写数据，这也是由电子线路实现，故较快。

④ 选定记录面后，就需要在一个磁道上确定准确的位置了，由于磁盘一直在旋转，因此最好的情况是正好要选的位置刚转过来立即可以读/写；而最坏的情况是要选的位置刚刚转过去，则须等磁盘转一圈后才能进行读/写。因此平均地看，须要等待磁盘转半圈后才能进行读/写，这段时间称为旋转延迟 (rolation delay)或等待时间(latency)，一般为 10 到 20 ms(毫秒)。

⑤ 当在磁盘上进行读/写时，由于电子线路传输信息的速度比磁盘的旋转速度要快得多，因此信息的传输率实际上决定于磁盘的旋转速度。而在读取信息的同时，也就可以对所读的内容进行必要的检查。这样，在查到要读的内容后，再接着读出并送往缓冲区即可。因此，在磁盘旋转一周的时间内，总可以完成对数据的读/写。

三、磁盘的寻址方法：

① 定长寻址：

基本的 I/O 交换单位是存贮桶(Buckel)，其大小可大于或等于扇面的大小，固定不变。寻址时须先后确定同位标磁道组号、磁道号、存贮桶号，由于存贮桶的大小是固定的，因此一般会有一部分存贮空间被浪费，如图 1-8 所示。

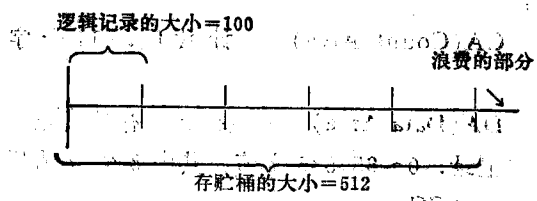


图1-8 定长寻址示意图

这种寻址方法较简单。

② 可变长寻址法

基本I/O交换单位是物理记录，寻址时须先后确定同位标磁道组号、磁道号、记录号。例如：IBM 2314 型磁盘，有两种格式，如图 1-9 所示：

其中

R_0——磁道描述记录(系统使用)；

$R_i, i \geqslant 1$——数据记录；

∇——标志点(每个磁道的物理起始点)；

G(GAP)——字段间的间隙，长短随记录长度而变。

HA(Home Address)——定义的物理地址和该磁道的物理条件，7 个字节；

F(Flag)——该磁道的物理条件，1 个字节；

C(Cylinder No.)——同位标磁道组号，2^n

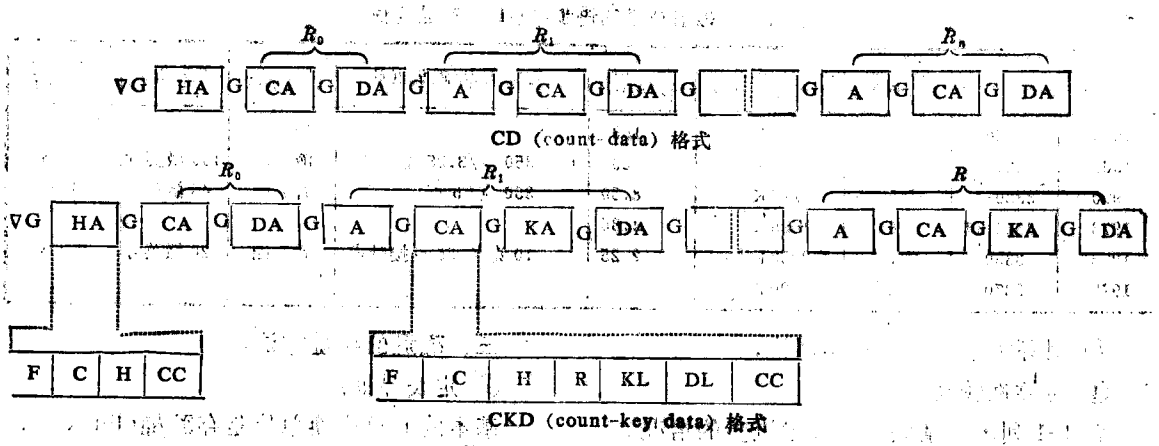


图 1-9 磁道格式示意图

个字节；

H(Head No.)——磁头号(即磁道号)，2个字节；

CC(Cyclic Check)——错误校验，2个字节；

CA(Count Area)——计数区，11个字节；

DA(Data Area)——数据区，存放真正的数据记录，0～65,535字节，其中8个字节用于 R_0+2CC；

A(Address Marker)——地址标志，2个字节；

R(Record No.)——记录号，1个字节；

KL(Key Length)——关键字长度，

对于CKD格式，为1～255字节；

对于CD格式，为0；

DL(Data Length)——数据长度(在0～65,535字节的范围内)，2个字节，0值指示文件的结尾，8为 R_0；

KA(Key Area)——关键字区，存放数据区中记录的关键字，1～255个字节，在一个文件中，它是固定长度的，同时，关键字不再出现在数据区中了。

各信息段之间也由间隙部分隔开。

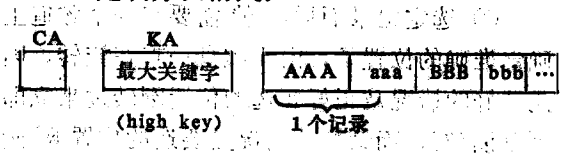
③ 分块方式

当采用分块方式时，可以节约原记录前的A、CA，KA和间隙部分，因而节约了存贮空间。当对记录连续处理时，只在深入一块时有一次旋转延迟所造成的等待时间，而不是每个记录需要一个等待时间，因此也可以节约存取时间。但当只对一个块中的一个记录进行处理时，则因需要将整个块输入到内存，反而会比不分块时慢一点。

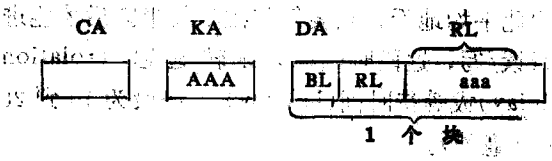
总之，可以有以下四种格式：

a 定长未分块格式

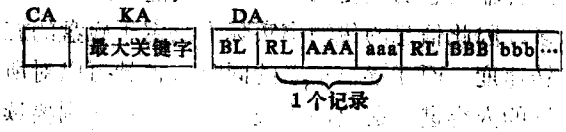
b 定长分块格式：



c 可变长未分块格式：



d 可变长分块格式：



四、磁道容量

对于可变长度寻址方法来说，由于有许多非数据信息(如A,CA,CC,gap等等)同时需要

记入磁道，因此真正存放数据的磁道容量与提供的容量之间可能相差较大，取决于每个磁道存放多少个记录，记录越多，其他信息占去的存贮位置就越多，真正存放数据的磁道容量与提供的容量之间的差别就越大。

以IBM 2314 型磁盘为例：

提供的磁道容量为 7294 字节，假定是采取带有关键字的可变长不分块的寻址方式，则：

最后一个记录的长度：KL＋DL＋C 字节

其他记录的长度：［534(KL＋DL)/512］＋C＋101 字节

其中

$$C=\begin{cases}0\text{ 字节} & \text{如果 KL}=0\\ 45\text{ 字节} & \text{如果 KL}\neq 0\end{cases}$$

而每个磁道可容纳的数据记录数

$$=1+\frac{\text{提供的容量}-\text{最后一个记录的长度}}{\text{其他记录的长度}}$$

对于 KL＝6，DL＝60

最后一个记录的长度＝6＋60＋45＝111字节

其他记录的长度＝［534(6＋60)/512］＋45＋101＝215 字节

每个磁道可容纳的数据记录数＝

$$1+\frac{7294-111}{215}=1+33=34$$

因此，真正的磁道容量＝34(6＋60)＝2244 字节

可见与提供的容量 7294 字节差别相当大。

五、磁盘存取的调度策略

对于活动臂磁盘，曾提出过几种对于读/写请求队列的调度方法，其中最简单的一种就是简单排队法(FCFS—First—Come—first—Serve，即"先来先服务")，在这种调度策略下，磁臂的移动完全是随机的，完全不考虑各个存取请求之间的位置关系及磁臂当前位置。

对 FCFS 方式的改进有以下几种方法：

① SSTF (Shortest—access—time—first)，即首先执行存取时间最短的那个请求，这种调度策略考虑了各个请求间的区别。

② SCAN(即扫描方式)：即当磁臂移动时，则一直沿一个方向移动到头扫过所有的同位标磁道组，然后再向相反的方向移动回来。在移动的过程中，执行相应的请求。

③ Look(即扫描方式的一种变种)：磁臂向一个方向移动并同时执行相应的存取请求，当在存取的方向上已不再有存取请求时，就掉转方向向回移动，即并不是每次移动都扫过所有同位标磁道组。

④ N—Step SCAN(即N步扫描)：将存取请求分成组，每组不超过N个请求，每次选一个组进行扫描，处理完一组后再选下一组。由于每个组构成之后不再变动，因此，这种调度方法可以减少各个"存取请求"的等待时间之间的差距。使之不致于太大。也就是说不会有某个存取请求总是不被执行或等待太久才被执行。

⑤ Eschenbach scheme：这是为了适应极大量存取请求的情况而设计的一种扫描方式。假是C个同位标磁道组的编号为 0,1,……C-1,则其扫描是从 0 号到 C-1 号顺序进行，然后再直接返回 0 号，再顺序扫描到 C—1 号，而在每一个同位标磁道组上，一直停留到磁盘转过 E 圈，然后再移向下一个同位标磁道组。为了在磁盘转动每一圈的时间内执行更多的对扇面的存取，因此，还考虑了旋转优化的问题。

为了对上述调度策略的适应性进行比较，图 1～10 给出了在 IBM2314 型磁盘上进行计算机模拟所得出的结果。

在进行模拟时作了如下假定：

① 只考虑单一的磁盘组和单 I/O 通道的情况；

② 所要存取的记录长度相等，均匀地分布在整个磁盘上，每个磁道有 m 个扇面，每个扇面只有一个记录，这里设 $m=50$；

③ 每个记录面只有一个读/写磁头，各个记录面的磁头相连在一起共同移动，共有 C 个同位标磁道组，这里 $C=200$；

④ 查找时间与磁臂扫过的同位标磁道组数成线性关系；

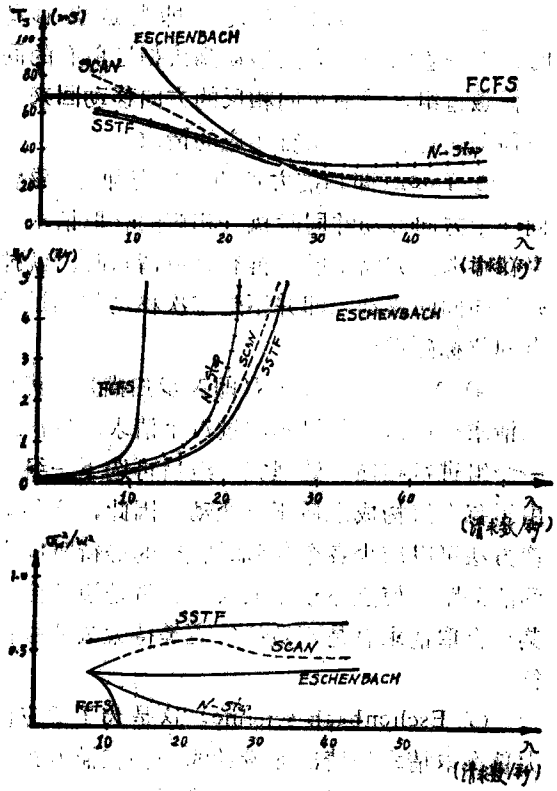


图 1-10 磁盘调度策略性能与输入速率的关系

⑤ 不考虑读和写之间的差别，不考虑选择存取请求时所化费的系统开销的影响。

图中：

T_s——存取请求的执行时间，也就是请求队列中执行相邻两个存取请求之间的时间间隔，等价于系统的吞吐量的倒数。

$$T_s=T_{sx}+\frac{T}{2}+\frac{T}{m}$$

T_{sx}——查找时间，即磁臂移动所花费的时间；

T——磁盘旋转一周的时间，这里$T=25$毫秒；

W——存取请求的等待时间，即从加入请求队列起至完全执行完毕止；

σ_w——各个存取请求之间的等待时间的差距；

λ——输入速率；

由图可见，当存取请求的量很小时，FCFS的性能相比之下尚可，但它与λ几乎无关，因此λ变大时，其性能较差。因此这种调度策略不适用于大多数实际情况。

SCAN和SSTF情况下，T_s较小，即吞吐量可以较大，同时等待时间也最大，这些方面两种方法性能很接近。但等待时间的差距最大，其中SSTF更大一些。因此一般来说，除掉λ小的情况之外，认为SCAN比SSTF好些。当λ很小时，不顾存取请求的具体情况而仍然扫过全部同位磁道组，当然造成性能不够好，因此Look是SCAN的一个改进。

在N—step—SCAN的情况下，等待时间之间的差距最小。而吞吐量和平均等待时间尚可，（比SCAN和SSTF稍差）由模拟的结果得出，如果对N的值不加限制，则可以改善。

对于Eschenbach的情况，虽然λ很大时，性能很好。λ较小时不好。

为了将T_s、W、和σ_w三个因素结合在一起进行考虑，可采用形如$Q_i=T_s^a(bW+C\sigma_w)$的公式对调度策略进行评价，Q_i为第i种调度策略的性能评价函数。

图1-11给出了当$a=1$，$b=1$，$c=2$时，各种调度策略的性能评价函数与输入速率λ之间的关系。（$M=4$）图1-12给出了$m=50$时

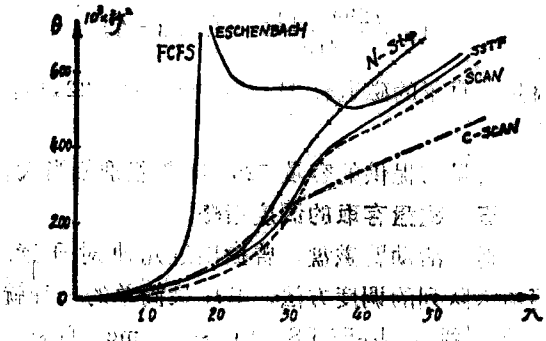


图 1-11 各种调度策略的性能评价函数示意图

的情况。

由图可见Eschenbach scheme方式在磁道上存放记录数量大时性能较好，这是由于磁臂的动作是从1到C—1号同位标磁道组不断重复进行的，因此最大等待时间最多是C个可

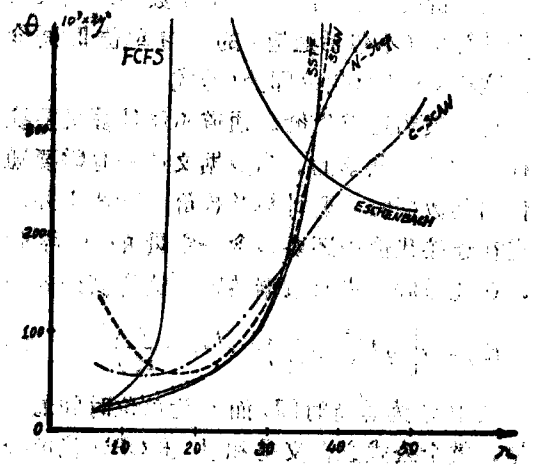


图 1-12　m=50 时的情况

能的停顿的一个周期。但对于 SCAN 来说，由于磁臂是来回运动的，因此最大等待时间为 $(2C-2)$个停顿(例如：假定磁臂刚从 1 号移向 2 号，这时有 1 个对 1 号的存取请求，但磁臂必须一直移动到 C—1 号，然后再向回移到 1 号，这将最大有$(2C-2)$次停顿)，可见，在这一方面，不断重复地从 1 到 $(C-1)$ 的移动方式，比来回移动的方式要好些，将 SCAN的磁臂移动方式作这样的改进方式，称为 C—SCAN，相应的实际方式称为 C—Look。

对 N—Step SCAN 来说不会有这方面的好处。这是因为它只执行预先构成的请求组的要求，而对新到的请求即使正好满足磁臂当时的位置，也不能加以执行。因此对于 N—Step SCAN 方式，不管磁臂以何种方式移动，都是差不多的。

第四节　在存储层次中数据文件的最优放置

在计算机系统中使用多种存储设备，例如磁带、磁盘、磁鼓、磁芯等等，这些形成了计算机的存储层次。一个数据文件应放在那一级上最合理，这要综合考虑存储代价和存取时间代价两方面的因素。为了定量地分析这些代价，从而找出最优的安排位置，我们要对其中的一些细节加以研究。这些原则同样地适用于数据库，为了叙述方便，我们下面只对数据文件加以讨论。

数据文件有二个状态：活动状态和不活动状态。在活动状态，即打开该文件后，程序对文件内容进行存取。反之，在不活动状态(关闭状态)，没有程序要对文件进行存取。显然，可以考虑这种工作方式：在不活动状态时，把文件放在第 i 层慢速存储设备上，这样节省其存储代价；而在活动状态时，将文件传送到较高的存储层次 j 层上去，这样在加工阶段，虽然花费了较高的存储代价，但也相应的节省了 CPU 存取时间代价。当加工完成后，根据对文件是否有更新，决定要不要把文件回记到较低的存储层次中。我们用下图来说明这一点。

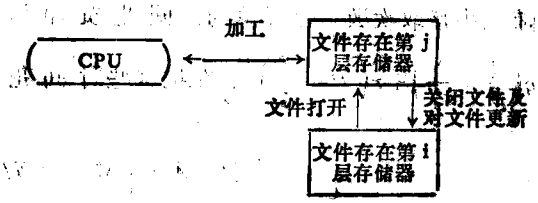


图 1-13

在一段时间间隔 T 中，要计算总的代价来决定对一个文件之最优存储层次 i 和 j。这个总代价包括以下几个方面：存储代价、CPU 代价、通道代价以及将数据在二个存储介质中传送时所需之代价。

存储代价为

$$G_{ij}=(P_iN_i+p_jn_s)\cdot S \qquad (1)$$

其中 P_i为文件存放在第 i 层存储器上所占时间百分比，而 n_i为第 i 层存储器单位存储之价格，S 为文件长度。如上所述，我们采用的是一个简化模型——第 i 层存储器上永远保留着原来文件。这样 $P_i=1$，公式化为—

$$G_{ij}=(n_i+p_jn_j)\cdot S$$

CPU 代价为

$$H_{ij}=m\cdot q\left(\frac{t_a^j}{\beta}+\frac{S}{t_s^j}\right)+m\cdot r\left(t_r^j+\frac{S}{t_s^j}\right) \qquad (2)$$

这一公式包含两部分，它们分别描述对文件顺序存取和随机存取的近似估计，分开考虑是因为两种存取方式在时间花费上可以存在显著的差异。每部分估计中又包括了存取时间和

传输时间两项。

先分析顺序存取模式。第 j 层存储设备顺序存取时间 t_i^j 中包括查找时间(Seektime)和等待时间(Latency)在内。一般说来它比随机存取时间 t_i^j 要少。因为顺序存取一般不需要存取臂的移动，即使需要移动时它也只需把存取臂移动一个柱面(Cylinder)。这两种情况的平均效果是把 t_i^j 除以 β，β 为缓冲区个数。另一项 S/t_s^j 代表数据块的传输时间。在顺序存取和随机存取中全都包含这一项。在这个式子中 t_s^j 为数据传输率，S 为块的大小。这样 $\left(\frac{t_i^j}{\beta}+\frac{S}{t_s^j}\right)$ 为顺序存取时间，类似地 (t_i^j+S/t_s^j) 为随机存取时间。我们用 q 和 r 分别记顺序块和随机块存取次数，这样总的时间花费即为 $\left[q\left(\frac{t_i^j}{\beta}+\frac{S}{t_s^j}\right)+r\left(t_i^j+\frac{S}{t_s^j}\right)\right]$。在把这些时间花费化成代价时，我们用到一个参数 m，它代表整个计算机(不包括通道)的平均单位时间价格。这样 CPU 总代价即为(2)式。

再一项为通道代价。通道不像计算机系统那样，它是不共享的。当数据文件一旦需要通道传输其数据时，要计算其代价。除此之外，通道在连接代价中还要包含一定量的等待时间，在此时间内没有数据传输，这样其公式为

$$K_{ij}=u\left[q\left(\frac{t_i^j}{\beta}+\frac{S}{t_s^j}\right)+r\left(t_i^j+\frac{S}{t_s^j}\right)\right] \qquad (3)$$

其中 t_i^j 为等待时间，而 u 为单位时间通道价格，整个公式之含义是和上述 CPU 代价分析类似的。

最后我们还要讨论把数据由 i 层搬到 j 层存储设备，以及因有更新可能还要搬回来之代价，这种代价为

$$L_{ij}=\begin{cases}(1+\lambda)d\left[MW=S/b_i\left(m\left(t_i^j+\frac{b_i}{t_s^j}\right)=u\frac{b_i}{t_s^j}\right)+\frac{S_j}{B_i}mt_o^j\right] & i=f \\ 0 & i=j\end{cases}$$

b_i 是在 i 到 j 层(或反过来)迁移中，每次存取之传输尺寸

B_i 为不用附加存取代价所能传输之最大尺寸

t_o^j 为最小存取臂移动时间

d 为数据打开次数

我们来解释一下这个式子的含义。因子 λ 刻划这种移动过程是如何进行的。例如，当打开时整个数据顺序地由 i 层移动到第 j 层，而当关闭时又一定回写到第 i 层，则取 $\lambda=1$。另一方面肯定不需要回写时，则 $\lambda=0$。一般情况用 λ 反映更新的百分比。括号中 MW 是准备数据传送所花费的代价。其中 W 代表打开操作所花时间，M 为计算机系统单位时间代价。mt_o^j 是机器等待下一个传输块之等待代价。再下面两项处理块传输代价，我们不仔细分析了，其中

这样总的代价

$$F_{ij}=G_{ij}+H_{ij}+K_{ij}+L_{ij}$$

为了确定数据文件的最优存放位置 i 和 j，我们对所有可能之 i,j 组合，计算花费之代价，然后挑选花费最少代价之位置。

我们举二个例子，假定所用到参数其值为：

	内存	磁鼓	磁盘	磁带	
t_i^j	10^{-6}	5.10^{-3}	60.10^{-3}	5	sec
t_s^j	——	10^6	3×10^6	5×10^4	byte/sec
t_o^j	0	8×10^{-3}	13.10^{-3}	25.10^{-3}	sec
t_i^j	0	8×10^{-3}	12.10^{-3}	20.10^{-3}	sec
t_o^j	0	0	25.10^{-3}	40.10^{-3}	sec
n_i	2.10^{-3}	5.10^{-4}	3.10^{-5}	3.10^{-7}	$/byte
b_i		20000	7000	2000	byte
B_i		4×10^6	14000	10000	byte

表 1-1

m=10 $/小时	$S=10^8$ byte	$P_1=1$
u=8 $/小时	q=90000	$P_2=P_3=P_4=\frac{1}{4}$
$\beta=2$	r=210000	
$w=0$	s= 1500 byte	
$\lambda=1$	d=30	

例 1：设 $S=10^8$ byte，q=90000，r=210000，s=1500 byte，d=30。计算 Fvi 得表 1-2 所列的结果。

表 1-2

i \ j	内存	鼓	盘	带	$S=10^8$ byte
内存	2×10^5				
鼓	7.2×10^5	5×10^4			
盘	6.7×10^5	17000	3118		
带	6.7×10^5	17390	1832	3007	

例 2：问题同上，假设 S 只有 10^5 字节，则此时结果如表 1-3。

表 1-3

i \ j	内存	鼓	盘	带	$S=10^5$ byte
内存	2000				
鼓	720	61			
盘	670	30	53		
带	667	28	52	2976	

在这两个例子中，文件最好存放位置 i 和 j 已用方框标出。关于这一部分更细致讨论可参见 Lum 等人的文章[6]。

第五节 文件结构概述

一批有用的数据可以分为三级，即数据项(data item，是基本的数据单位)、记录(record，是由一个或多个数据项根据一定的目的而组成的数据集合)和文件(file，是为了处理的方便，由多个记录组合而成的数据的逻辑单位)。用于组织文件的基本数据项称为关键字(key)。

文件的结构方式一般可分为四种，即顺序结构、随机结构、表结构和树状结构。下面分别加以简单介绍。较详细的讨论将在第二章到第六章中给出。

一、顺序结构

顺序文件结构在数据处理历史上是最早使用的。其中的数据记录是根据记录中某一共同的属性来排列，有时是按照一定的次序进行排列的。这个次序一般是由记录中的某一基本数据项值的大小顺序而排列，这时这一基本数据项就是关键字。

顺序文件结构的基本优点就是在连续存取时速度较快。即，如果文件中的第 n 个记录刚被存取过，而下一个要存取的是第 $n+1$ 个记录，则这个存取将会很快完成。当顺序文件是存在单一存贮设备(如磁带)上时，这一优点总是可以保持的。但如果它是存在多路存贮设备(如磁盘)上，则在多道程序的情况下，由于其他用户可能驱使磁头移向其他柱面，就会降低这一优越性。因此，顺序文件结构多用于磁带。

当需要对顺序文件中的某个记录进行查找时，一般是采用扫描的方式，即扫视整个文件，直到所需的记录被找到为止。当文件规模很大时，这个扫描过程会相当地长。因此，通常是将那些处理程序(transaction)对顺序文件进行检索的要求加以积累，把它们所要查找的那些积累起来的记录加以排序，然后一次从头查到尾，即可完成多个处理程序的查找要求。这种处理方式称为成批处理技术。这时对于每个处理程序，所花费的查找代价很小。但这种技术对于在线(实时)应用来说，不大实用。因为在很短的时间间隔内，积累的成批处理的规模太小，不能表现出其优越性。

当需要在多于一个关键字的基础上处理一个文件时，则需要建立双份以至于多份重复的文件，每个文件以不同的关键字排序。在这种情况下，效率显然降低了。

总之，对于连续处理的情况，顺序文件结构是一种最经济的结构方式。但如果对它不按顺序进行存取，则处理起来很慢。同时，小量的修改是非常不合算的。这是因为任何对顺序文件的修改，都要求把整个文件重新复制一遍。

我们在第二章要详细讨论顺序文件的有关问题。

二、随机结构

在随机结构中，数据记录存放于直接存取型存贮设备上，在数据记录的关键字与其地址之间建立了某种关系，随机文件的记录就是按这种关系排布，并利用这种关系进行存取。

有三种随机文件结构：直接地址结构，索引(index)结构和计算寻址结构。

1. 直接地址结构

当知道某个记录的地址时，可直接使用这个地址进行存取。但这意味着，用户必须知道每个记录的具体地址，或者在记录中必须有某个数据段的值直接作为地址使用。这都是不方便的。因此，直接地址结构并不常用。当然在使用这种结构时，存取效率是最高的，因为不需要进行任何"查找"。在某些数据库系统中的"数据库码"(Database Key)，就是这种结构方法。

2. 索引结构

这种结构使用一个索引表，其中包括一组关键字和对应的地址。查阅索引表，由记录的关键字可以得出这个记录所在的地址，由此地址则可找到该记录。当增加或删除记录时，应对索引表及时加以修改。由于每次存取都涉及到索引表的查找，因此，所采用的查找策略对文件设计的效率有很大的影响。通常使用的查找策略有两种：二分查找和顺序扫描。

如果索引表没有任何次序，则只能采取顺序扫描的方法。假定索引表有n项，则查到所需地址的平均查找次数为$(n+1)/2$。

如果索引表根据关键字的大小按顺序排列，则可以采用二分查找法。即首先查看索引表的中间，看其是否为所要找的关键字。如不是，则根据其值的大小就确定所要找的关键字是在索引表的哪一半里。然后再在剩下的这一半里重复上述过程。可见每次可将索引表的规模减小一半再进行查找，直到找到（或核实该索引表中无此关键字）为止。

为了更有效地使用索引表，应将它存放在主存贮器里。但在很多情况下，索引表也是相当大的，不可能全部放在主存中。因此需要将索引表也分成若干块。但如果这些块也太大，则须再分。如此下去，就形成一个索引表的层次结构，即树结构。在这种情况下，还有一个因素要注意，即应把最经常使用的那部分索引表存放在主存内，以便提高使用效率。索引结构的详细讨论将在第四章给出。

3. 计算寻址结构

在这种方法中，关键字经过某种计算处理，转换成相应的地址。由于一般来说地址的总数比可能的关键字的值的总数要小得多，即不会是一对一的关系，因此不同的关键字在计算之后，可能会得出相同的地址，称为"冲突"。

这种计算式方法就是通常所说的散列(hash，也译为"杂凑")。一种散列算法是否成功的一个重要标志，是看其将不同的关键字映射到同一地址的几率有多大。这个几率越小，则此散列算法的性能就越好。即"冲突"产生的越少越好。

一种流行的散列算法是把一个关键字分裂为两半，将这两半的值相乘，取其乘积的中间部分数位作为散列地址。

散列算法的主要问题就是"冲突"问题。必须有某种方法来处理"冲突"，此时将增加相当多的额外处理代价。因此，"冲突"是使计算寻址结构性能变坏的主要因素。在计算寻址结构中，应该仔细选择好的散列算法和好的处理冲突的方法。

与散列有关的一个概念是"压缩"(Compression)，这是为了尽量减少关键字所占用的存贮空间，而将长的关键字"压缩"成较短的字符串。因此，压缩算法也面临着与散列同样的问题——避免将不同的关键字压缩成同一个关键字。

在关键字内含有某些错误的情况下，压缩技术特别有用。例如当需要处理由人口授的关键字时，如果同一词汇由于口授时拼法的差别，可能得出几种不同的关键字。我们可以根据语音学的原理，找出特殊的压缩算法，把这几种

本来应该一样的不同的关键字又压缩回同一关键字。这时如果同时在检索时又使用散列算法，这就可能成为同时使用压缩和散列的一个例子。

在第三章中，将专门讨论散列问题。

三、表结构(list)

表可以被定义为一个或多个记录或表的有穷的序列。

表结构的特点是使用了指针(Pointer)来表达各个记录之间的关联。指针可以表示记录的绝对地址，也可以表示相对于文件中第一个记录的相对地址。使用指针的一个特点是可以将文件的逻辑排列与其物理排列完全分开，即记录在物理上可以任意排布，而利用指针来表示它们之间的逻辑关系。因此使用指针甚至可以表示出循环的数据结构。而如果不使用指针，我们就无法表示出一个循环的数据结构。当然，如果一个表中没有任何循环或相交的关系，同时记录之间的物理顺序与其逻辑顺序相一致，则可以不使用指针。而这时的存放方式即为顺序结构。反之，如果使用指针将有关的记录连接起来，则称为链接结构。链接结构比较容易修改，但由于要存放指针，需要较多的存贮空间。

表结构又可分为线性表、倒排表和环形表。

1. 线性表(linear list)

按记录之间的相对线性位置组成的一组记录，称为线性表。线性表按其插入和删除操作的方式，可以有以下三种特殊情况。

1) 堆栈(stack)

其中所有的插入和删除只能在其同一端进行，这一端称为"顶端"(top)。这是一个LIFO(last in first out)队，即"后进先出"队。

2) 队列(queue)

其插入在"后端"(back)进行，而删除在"前端"(front)进行。这是一个FIFO(first in first out)队，即"先进先出"队。

3) 两用队(deque)

有左(left)和右(right)两端，其中所有的插入和删除既可以在左端进行，也可以在右端进行。

这三种结构是经常碰到的，也是很有用的。

一个线性表可以是顺序结构，也可以是链接结构。一般来说，增加和删除可以在表中的任意一点进行。有时，一个记录还可以同时处于多个表内。在链接结构中，其主要优点是，记录的增加和删除很容易进行。

2. 倒排文件(inverted file)

倒排文件由一些倒排表组成。每一个倒排表涉及到记录中某个关键字的一个特定的值。同时，有一些指针指向文件中所有具有该关键字的这个值的那些记录。通常，对于记录中的所有的数据项都有对应的倒排表，因此可以允许用户基于任一数据项，迅速地对这个文件进行存取操作。可是，这种结构修改起来很困难。因为所有有关的倒排表都必须相应地进行修改。因此，如果相对修改操作很少，或修改可以成批地进行，则倒排文件结构用于检索还是相当好的。

倒排文件结构的一个变种是多目表(multilist)结构。它由一个顺序的索引组成，其中每一个关键字给出另一个表的起始地址，而这个表将所有与该关键字有关的记录链接在一起。在这种结构中，每一个原始记录均不重复出现，因此所有与该记录有关的关键字的链都将通过它。这样，一个记录中就可能包含多于一个的链接字段(link field)。

如果把多目表分成为若干页，在索引中的地址由页号和页中的记录号表示，而又将每个链接表的长度限制在一个页内，则这种结构称为一个页式多目表(cellular multilist)。这时，每一个倒排表是由若干个子表(sublist)表示，而每个子表都是在一个页内的链接表，索引则指向每个子表的第一个记录。

多目表比倒排文件易于修改，因为它避免了像倒排表所要求的全部进行重组的必要性。但多目表检索起来比较慢，因为在链接表中需要有一个查找的过程。页式多目表在修改和检