BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2102
DATABASE MANAGEMENT SYSTEM

Unit 4

File Organization for Conventional DBMS

Table of Contents

1. INTRODUCTION

Although a database system provides a high-level view of data, ultimately data have to be stored as bits on one or more storage devices. A vast majority of databases today store data on a magnetic disk and fetch data into main space memory for processing, or copy data onto tapes and other backup devices for archival storage. The physical characteristics of storage devices play a major role in the way data are stored, in particular because access to a random piece of data on disk is much slower than memory access: Disk access takes tens of milliseconds, whereas memory access takes a tenth of a microsecond.

1.1 Objectives:

By the end of unit 4, the learners should be able to understand:

- *Different Storage devices and their characteristics*
- *File Organization and records in file organizations*
- *Organization of Sequential files*
- *Indexed Sequential Access Method (ISAM)*
- *Virtual Storage Access Method (VSAM)*

2. STORAGE DEVICES AND ITS CHARACTERISTICS

Several types of data storage exist in most computer systems. These storage media are classified by the speed with which data can be accessed,by the cost per unit of data to buy the medium, and by the medium's reliability. Among the media typically available are these:

- **Cache.** The cache is the fastest and most costly form of storage after register. Register has less storage capacity than cache. Cache memory is small; its use is managed by the computer system hardware. We shallnot be concerned about managing cache storage in the database system.
- **Main memory.** The storage medium used for data that are available to be operated on the main memory. The general-purpose machine instructions operate on the main memory. Although main memory may contain many megabytes of data or even gigabytes of data in large server systems, it is generally too small (or too expensive) for storing theentire database. The contents of the main memory are usually lost if a power failure or system crash occurs.
- **Flash memory.** Also known as electrically erasable programmable read-only memory (EEPROM), flash memory differs from main memory in that data survives power failure. Reading data from flash memory takes less than 100 nanoseconds (a nanosecond is 1/1000 of a microsecond), which is roughly as fast as reading data from the main memory.
- **Magnetic-disk storage.** The primary medium for the long-term online storage of data is the magnetic disk. Usually, the entire database is stored on a magnetic disk. The system must move the data from the disk to the main memory so that they can be accessed. After the system has performed the designated operations, the data that have been modified must be written to disk.
- **Optical storage.** The most popular forms of optical storage are the compact disks (CD), which can hold about 640 megabytes of data, and the digital video disk (DVD) which can hold 4.7 or 8.5 gigabytes of data per side of the disk (or up to 17 gigabytes on a two-sided disk). Data arestored optically on a disk and are read by a laser. The optical disks used in read-only compact disks (CD-ROM) or read-only digital video disk (DVD-ROM) cannot be written, but are supplied with data prerecorded.

- **Tape storage.** Tape storage is used primarily for backup and archival data. Although magnetic tape is much cheaper than disks, access todata is much slower, because the tape must be accessed sequentially from the beginning. For this reason, tape storage is referred to as**sequential-access** storage. In contrast, disk storage is referred to as **direct-access** storage because it is possible to read data from any location on a disk.

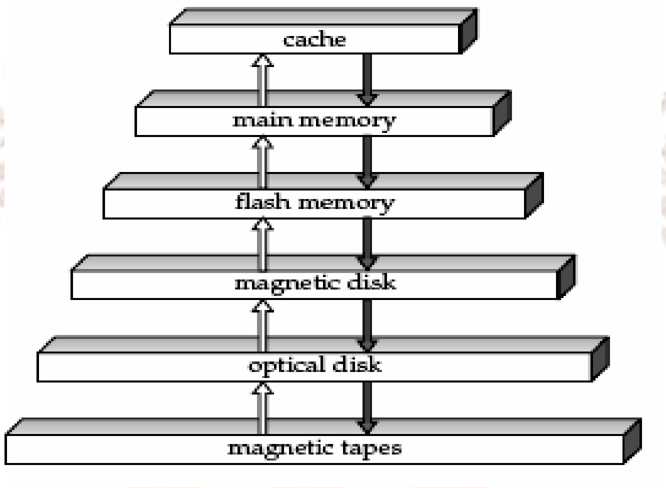


Fig 4.1: Storage Hierarchy

The fastest storage media – for example, cache and the main memory – are referred to as **primary storage**. The media in the next level in the hierarchy – for example, magnetic disks – are referred to as **secondary storage** or **online storage**. The media in the lowest level in the hierarchy –for example, magnetic tape and optical disk jukeboxes – are referred to as **tertiary storage**, or **offline storage**.

In addition to the speed and cost of the various storage systems, there is also the issue of storage volatility. **Volatile storage** loses its contents when the power to the device is removed. In the hierarchy shown in Figure 4.1,the storage systems from main memory up are volatile, whereas the storagesystems below main memory are nonvolatile. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage** for safekeeping.

2.1 Magnetic Disks

Magnetic disks provide the bulk of secondary storage for modern computer systems. Disk capacities have been growing at over 50 percent per year, but the storage requirements of large applications have also been growing very fast, in some cases even faster than the growth rate of disk capacities. A large database may require hundreds of disks.

2.2 Physical Characteristics of Disks

Physically, disks are relatively simple (Figure 4.2). Each disk **platter** has a flat circular shape. Its two surfaces are covered with a magnetic material, and information is recorded on the surfaces. Platters are made from rigid metal or glass and are covered (usually on both sides) with magnetic recording material. We call such magnetic disks **hard disks**, to distinguish them from **floppy disks**, which are made from a flexible material.

The disk surface is logically divided into **tracks**, which are subdivided into **sectors**. A **sector** is the smallest unit of information that can be read from or written to the disk. The **read-write head** stores information on a sector magnetically as reversals of the direction of magnetization of the magnetic material. There may be hundreds of concentric tracks on a disk surface, containing thousands of sectors.

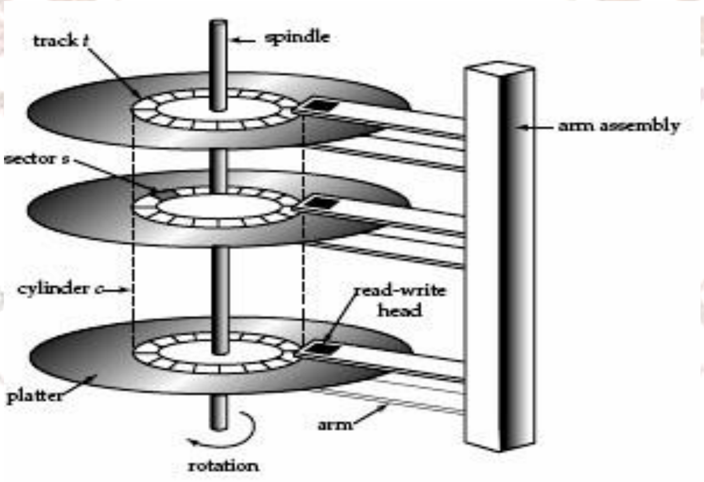


Fig 4.2: Magnetic Disks

Each side of a platter of a disk has a read-write head, which moves across the platter to access different tracks. A disk typically contains many platters, and the read-write heads of all the tracks are mounted on a single assemblycalled a **disk arm**, and move together. The disk platters mounted on a spindle and the heads mounted on a disk arm are together known as**head–disk assemblies**.

A **fixed-head disk** has a separate head for each track. This arrangement allows the computer to switch from track to track quickly, without having to move the head assembly, but because of the large number of heads, the device is extremely expensive. Some disk systems have multiple disk arms, allowing more than one track on the same platter to be accessed at a time. Fixed-head disks and multiple-arm disks were used in high-performance mainframe systems, but are no longer in production.

A **disk controller** interfaces between the computer system and the actual hardware of the disk drive. It accepts high-level commands to read or writea sector and initiates actions, such as moving the disk arm to the right track and reading or writing the data. Disk controllers also attach **checksums** to each sector that is written; the checksum is computed from the data written to the sector. When the sector is read back, the controller computes the checksum again from the retrieved data and compares it with the stored checksum; if the data are corrupted, with a high probability the newly computed checksum will not match the stored checksum. If such an error occurs, the controller will retry the read several times; if the error continues to occur, the controller will signal a read failure. A disk controller interface is shown in Figure 4.3.

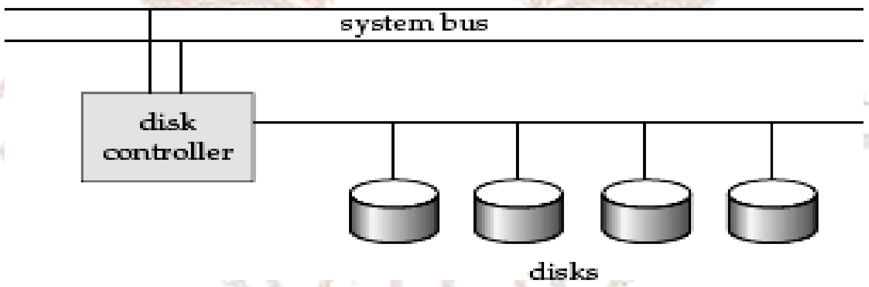


Fig 4.3: Disk controller interface

In the **storage area network** (**SAN**) architecture, large numbers of disks areconnected by a high-speed network to a number of server computers. The disks are usually organized locally using **redundant arrays of independentdisks** (**RAID**) storage organizations, but the RAID organization may be hidden from the server computers: the disk subsystems pretend each RAID system is a very large and very reliable disk.

2.3 Performance Measures of Disks

The main measures of the qualities of a disk are capacity, access time, data transfer rate, and reliability.

Access time is the time from when a read or write request is issued to when data transfer begins. To access (that is, to read or write) data on a given sector of a disk, the arm first must move, so that it is positioned over the correct track, and then must wait for the sector to appear under it as the disk rotates. The time for repositioning the arm is called the **seek time**, andit increases with the distance that the arm must move. Typical seek times range from 2 to 30 milliseconds, depending on how far the track is from the initial arm position. Smaller disks tend to have lower **seek time** since the head has to travel a smaller distance.

The **average seek time** is the average of the seek times, measured over a sequence of (uniformly distributed) random requests. If all tracks have the same number of sectors, and we disregard the time required for the head to start moving and to stop moving, we can show that the average seek time isone-third of the worst case seek time. Taking these factors into account, the average seek time is around one-half of the maximum seek time. Average seek time currently ranges between 4 milliseconds and 10 milliseconds, depending on the disk model.

Once the seek time has started, the time spent waiting for the sector to be accessed to appear under the head is called the **rotational latency time**. On an average, one-half of a rotation of the disk is required for the beginning of the desired sector to appear under the head. Thus, the **average latency time** of the disk is one-half the time for a full rotation of thedisk.

The access time is then the sum of the seek time and the latency, and ranges from 8 to 20 milliseconds. Once the first sector of the data to be accessed has come under the head, the data transfer begins. The **data- transfer rate** is the rate at which data can be retrieved from or stored on the disk.

The final commonly used measure of a disk is the **meantime to failure (MTTF)**, which is a measure of the reliability of the disk. The mean time to failure of a disk (or of any other system) is the amount of time that, on average, we can expect the system to run continuously without any failure.

2.4 Optimization of Disk-Block Access

Requests for disk I/O are generated both by the file system and by the virtual memory manager found in most operating systems. Each request specifies the address on the disk to be referenced; that address is in the form of a *block number*. A **block** is a contiguous sequence of sectors from a single track of one platter. Block sizes range from 512 bytes to several kilobytes. Data is transferred between disk and main memory in units of blocks. The lower levels of the file-system manager convert block addressesinto the hardware-level cylinder, surface, and sector number.

Since access to data on disk is several orders of magnitude slower than access to data in main memory, equipment designers have focused on techniques for improving the speed of access to blocks on disk.

- **Scheduling:** If several blocks from a cylinder need to be transferredfrom disk to main memory, we may be able to save access time by requesting the blocks in the order in which they will pass under the heads. If the desired blocks are on different cylinders, it is advantageousto request the blocks in an order that minimizes disk-arm movement. **Disk-arm-scheduling** algorithms attempt to order accesses to tracks in a fashion that increases the number of accesses that can be processed. A commonly used algorithm is the **elevator algorithm**, which works in the same way many elevators do.
- **File organization:** To reduce block-access time, we can organize blocks on disk in a way that corresponds closely to the way we expect data to be accessed. For example, if we expect a file to be accessed sequentially, then we should ideally keep all the blocks of the file sequentially on adjacent cylinders. Older operating systems, such as theIBM mainframe operating systems, provided programmers with fine control on placement of files, allowing a programmer to reserve a set of cylinders for storing a file. However, this control places a burden on the programmer or system administrator to decide, for example, how many cylinders to allocate for a file, and may require costly reorganization if data is inserted into or deleted from the file.
- **Nonvolatile write buffers:** Since the contents of the main memory are lost in a power failure, information about database updates has to be recorded on the disk to survive possible system crashes. For this reason, the performance of update-intensive

database applications, such astransaction-processing systems, is heavily dependent on the speed of disk writes.

- **Log disk:** Another approach to reducing write latencies is to use a log disk – that is, a disk devoted to writing a sequential log – in much the same way as a nonvolatile RAM buffer. All access to the log disk is sequential, essentially eliminating seek time and several consecutive blocks can be written at once, making write to the log disk several times faster than random writes. As before, the data have to be written to their actual location on disk as well, but the log disk can do the write later, without the database system having to wait for the write to be completed. Furthermore, the log disk can reorder the writes to minimize disk arm movement. If the system crashes before some writes to the actual disk location have been completed, when the system comes back, it reads the log disk to find those writes that had not been completed, and carries them out then. File systems that support log disks as above are called **journaling file systems**.

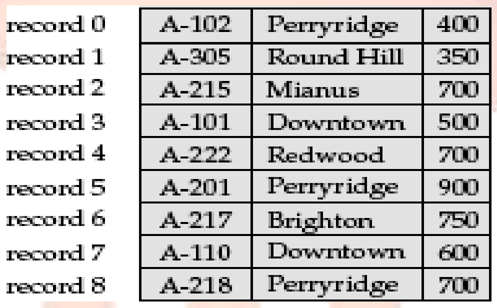
Self-Assessment Questions – 1

1. The____________is the fastest and most costly form of storage.
2. The contents of main memory are usually__________if a power failure or system crash occurs.
3. Flash memory differs from main memory in that data survive___________.
4. The primary medium for the long-term on-line storage of data is the ___________ .
5. The CD and DVD come under_____________memory storage.
6. Tape storage is referred to as____________access storage.
7. Each disk platter has a flat______________shape.
8. A____________interfaces between the computer system and the actual hardware of the disk drive.
9. In the_____________architecture, large numbers of disks areconnected by a high-speed network to a number of server computers.
10. ______________time is the time from when a read or write request is issuedto when data transfer begins.
11. The___________is the average of the seek times, measured over asequence of (uniformly distributed) random requests.
12. A____________is a contiguous sequence of sectors from a single trackof one platter.
13. A commonly used algorithm in the scheduling is___________.

3. FILE ORGANIZATION

A **file** is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, so we shall assume the existence of an underlying *file system*. We need to consider ways of representing logical data models in terms of files.

Although blocks are of a fixed size determined by the physical properties of the disk and by the operating system, record sizes vary. In a relational database, tuples of distinct relations are generally of different sizes. A file containing account records is shown in Figure 4.4.



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Fig 4.4: File containing account records

One approach to mapping the database to files is to use several files and store records of only one fixed-length in any given file.

3.1 Fixed-Length Records

As an example, let us consider a file of *account* records for our bankdatabase. Each record of this file is defined as:

type *deposit* = **record**

account-number : char(10);

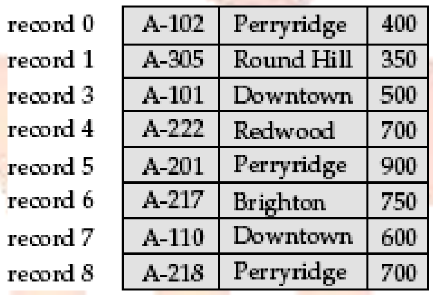
branch-name : char (22);

balance : real;

end

If we assume that each character occupies 1 byte and that a real occupies8 bytes, our *account* record is 40 bytes long. A simple approach is to usethe first 40 bytes for the first record, the next 40 bytes for the second record,and so on (Figure 4.5). However, there are two problems with this simple approach:

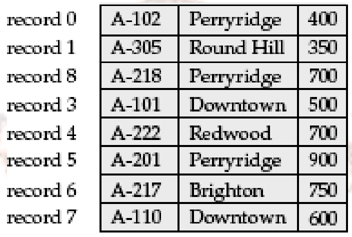
1. It is difficult to delete a record from this structure. The space occupiedby the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.
2. Unless the block size happens to be a multiple of 40 (which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two-block accesses to read or write such a record.



record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Fig 4.5: File of Figure 4.4, with record 2 deleted and all records moved.

When a record is deleted, we could move the record that came after it into the space formerly occupied by the deleted record, and so on, until every record following the deleted record has been moved ahead (Figure 4.5). Such an approach requires moving a large number of records. It might be easier simply to move the final record of the file into the space occupied by the deleted record (Figure 4.6).



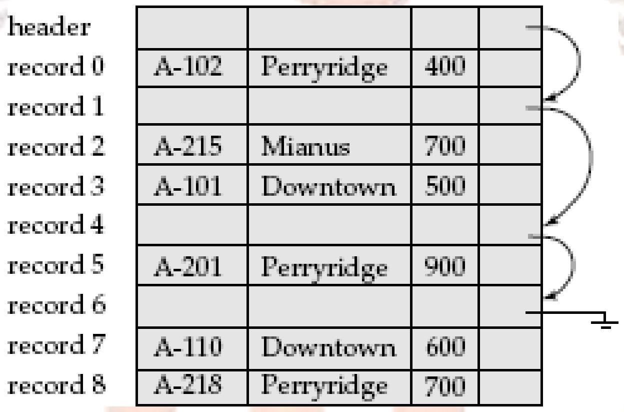
record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 8	A-218	Perryridge	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600

Fig 4.6: File of Figure 4.4, with record 2 deleted and final record moved.

It is undesirable to move records to occupy the space freed by a deleted record since doing so requires additional block accesses. Since insertions tend to be more frequent than deletions, it is acceptable to leave open the space occupied by the deleted record and to wait for a subsequent insertion before reusing the space. A simple marker on a deleted record is not sufficient, since it is hard to find this available space when an insertion isbeing done. Thus, we need to introduce an additional structure.

At the beginning of the file, we allocate a certain number of bytes as a **file header**.

The header will contain a variety of information about the file. For now, all we need to store there is the address of the first record whose contents are deleted. We use this first record to store the address of the second availablerecord, and so on. Intuitively, we can think of these stored addresses as *pointers*, since they point to the location of a record.



header				
record 0	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	

Fig 4.7: File of Figure 4.4, with free list after deletion of records 1, 4 and 6

The deleted records thus form a linked list, which is often referred to as a **free list**. Figure 4.7 shows the file of Figure 4.4, with the free list, after records 1, 4, and 6 have been deleted.

3.2 Variable-Length Records

Variable-length records arise in database systems in several ways:

- Storage of multiple record types in a file
- Record types that allow variable lengths for one or more fields
- Record types that allow repeating fields

Different techniques for implementing variable-length records exist. For purposes of illustration, we shall use one example to demonstrate the various implementation techniques. We shall consider a different representation of the *account* information stored in the file of Figure 4.4, in which we use one variable-length record for each branch name and for all the account information for that branch. The format of the record is

```
    type account-list = record
    branch-name : char (22);
    account-info : array [1 ..∞] of record;
account-number : char(10);
    balance: real;
end
    end
```

We define *account-info* as an array with an arbitrary number of elements. That is, the type definition does not limit the number of elements in the array, although any actual record will have a specific number of elements in its array. There is no limit on how large a record can be (up to, of course,the size of the disk storage!).

The **slotted-page structure** appears in Figure 4.8. There is a header at thebeginning of each block, containing the following information:

1. The number of record entries in the header
2. The end of free space in the block
3. An array whose entries contain the location and size of each record

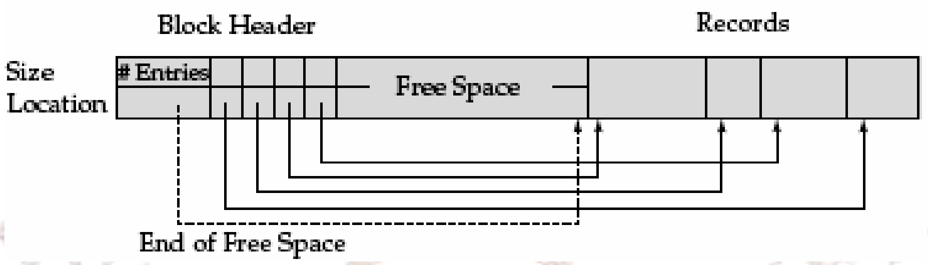


Fig 4.8: Slotted-page structure

The actual records are allocated *contiguously* in the block, starting from the end of the block. The free space in the block is contiguous, between thefinal entry in the header array, and the first record. If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

If a record is deleted, the space that it occupies is freed, and its entry is set to deleted (its size is set to −1, for example). Further, the records in the block before the deleted record are moved, so that the free space created by the deletion gets occupied, and all free space is again between the final entry in the header array and the first record. The end-of-free-space pointer in the header is appropriately updated as well.

3.3 Organization of Records in Files

So far, we have studied how records are represented in a file structure. An instance of a relation is a set of records. Given a set of records, the next question is how to organize them in a file. Several of the possible ways of organizing records in files are:

- **Heap file organization:** Any record can be placed anywhere in the file where there is space for the record. There is no ordering of records. Typically, there is a single file for each relation
- **Sequential file organization:** Records are stored in sequential order, according to the value of a "search key" of each record.
- **Hashing file organization:** A hash function is computed on some attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

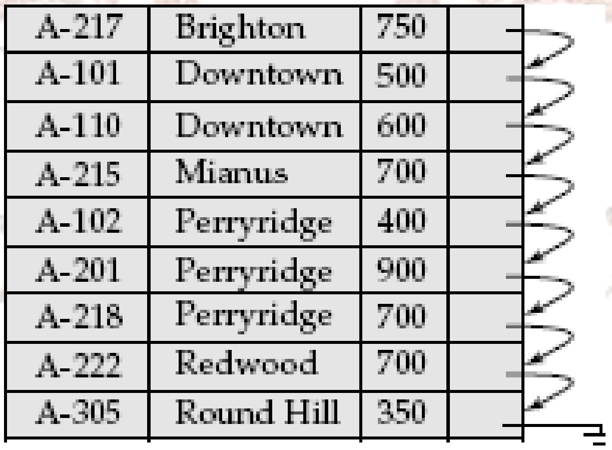
4. SEQUENTIAL FILE ORGANIZATION

A **sequential file** is designed for efficient processing of records in sorted order based on some search key. A **search key** is any attribute or set of attributes; it need not be the primary key, or even a super key. To permitfast retrieval of records in search-key order, we chain together records by pointers. The pointer in each record points to the next record in search-key order. Furthermore, to minimize the number of block accesses in sequential file processing, we store records physically in search-key order, or as close to search-key order as possible.

Figure 4.9 shows a sequential file of *account* records taken from our banking example. In that example, the records are stored in search-key order, using *branchname* as the search key.

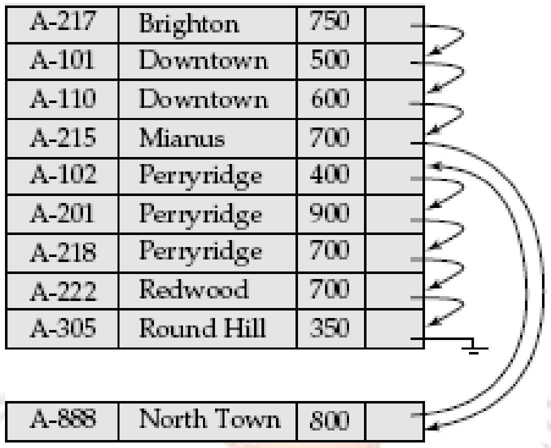
The sequential file organization allows records to be read in sorted order; that can be useful for display purposes, as well as for certain query-processing algorithms.

It is difficult, however, to maintain physical sequential order as records are inserted and deleted, since it is costly to move many records as a result of asingle insertion or deletion. We can manage deletion by using pointer chains, as we saw previously.



A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Fig 4.9: Sequential file for account records



A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	
A-888	North Town	800	

Fig 4.10: Sequential file after insertion

For insertion, we apply the following rules:

1. Locate the record in the file that comes before the record to be inserted in search-key order.
2. If there is a free record (that is, space left after a deletion) within the same block as this record, insert the new record there. Otherwise, insert the new record in an *overflow block*. In either case, adjust the pointersso as to chain together the records in search-key order.

Figure 4.10 shows the file of Figure 4.9 after the insertion of the record (North Town, A-888, 800). The structure in Figure 4.10 allows the fast insertion of new records, but forces sequential file-processing applications to process records in an order that does not match the physical order of the records.

5. INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

To understand the motivation for the ISAM technique, it is useful to begin with a simple sorted file. Consider a file of Students records sorted by *GPA (grade point average) as shown in Figure 4.11.*

Students(*sid:* string, *name:* string, *login:* string, *age:* integer, *gpa:* real)

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Fig 4.11: An Instance of the Students Relation

To answer a range selection such as "Find all students with a gpa higher than 3.0" we must identify the first such student by doing a binary search of the file and then scan the file from that point on. If the file is large, the initial binary search can be quite expensive; can we improve upon this method?

One idea is to create a second file with one record per page in the original (data) file, of the form (first key on-page, pointer to page), again sorted by the key attribute (which is *gpa* in our example). The format of a page in the second *index* file is illustrated in Figure 4.12.

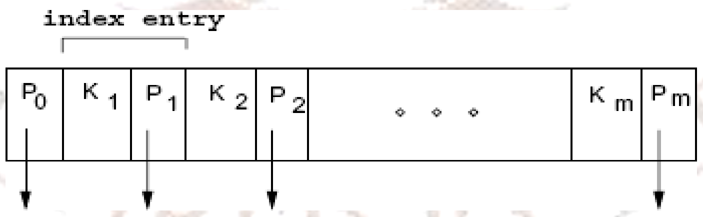


Fig 4.12: Format of an index page

We refer to pairs of the form <*key, pointer*> as *entries*. Notice that each index page contains one pointer more than the number of key searches. Key serves as a *separator* for the contents of the pages pointed to by the pointers to its left and right. This structure is illustrated in Figure 4.13.

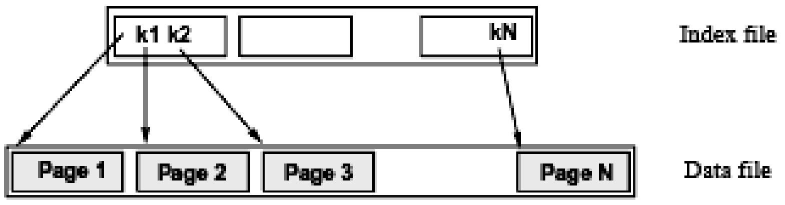


Fig 4.13: One level index structure

We can do a binary search of the index file to identify the page containing the first key (*gpa*) value that satisfies the range selection (in our example, the first student with *gpa* over 3.0) and follows the pointer to the page containing the first data record with that key value. We can then scan the data file sequentially from that point on to retrieve other qualifying records. This example uses the index to find the first data page containing a Students record with *gpa* greater than 3.0, and the data file is scanned from that point on to retrieve other such Student records.

Because the size of an entry in the index file (key value and page id) is likelyto be much smaller than the size of a page, and only one such entry exists per page of the data file, the index file is likely to be much smaller than the data file; thus, a binary search of the index file is much faster than a binary search of the data file. However, a binary search of the index file could still be fairly expensive, and the index file is typically still large enough to make inserts and deletes expensive.

The potential large size of the index file motivates the ISAM idea: Why not apply the previous step of building an auxiliary file on the *index file* and soon recursively until the final auxiliary file fits on one page? This repeated construction of a one-level index leads to a tree structure that is illustrated inFigure 4.14.

The data entries of the ISAM index are in the leaf pages of the tree and additional *overflow* pages that are chained to some leaf page. In addition, some systems carefully organize the layout of pages so that page boundaries correspond closely to the physical characteristics of the underlying storage device. The ISAM structure is completely static (except for the overflow pages, of which it is hoped, there will be few) and facilitates such low-level optimizations.

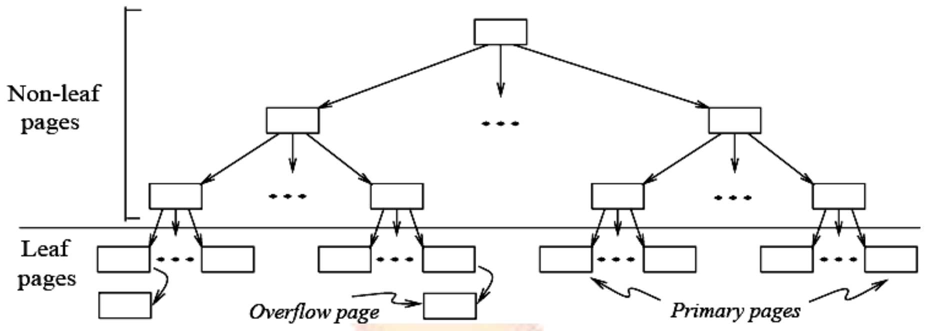


Fig 4.14: ISAM Index Structure

Each tree node is a disk page, and all the data resides in the leaf pages. This corresponds to an index that uses Alternative (1) for data entries, in terms of the alternatives; we can create an index with Alternative (2) by storing the data records in a separate file and storing <*key, rid*> pairs in the leaf pages of the ISAM index. When the file is created, all leaf pages are allocated sequentially and sorted on the search key value. (If Alternatives (2) or (3) are used, the data records are created and sorted before allocating the leaf pages of the ISAM index.) The non-leaf level pages are then allocated. If there are several inserts to the file subsequently, so that more entries are inserted into a leaf than will fit onto a single page, additional pages are needed because the index structure is static. These additional pages are allocated from an overflow area. The allocation of pages is illustrated in Figure 4.15.

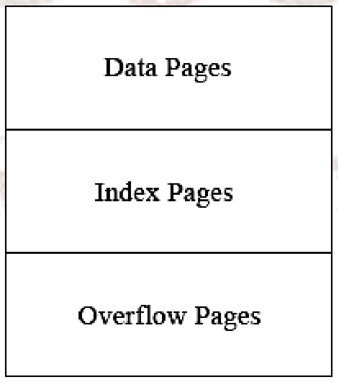


Fig 4.15: Page allocation in ISAM

The basic operations of insertion, deletion, and search are all quite straightforward. For an equality selection search, we start at the root node and determine which subtree to search by comparing the value in thesearch field of the given record with the key values in the node. For a range query, the starting point in the data (or leaf) level is determined similarly, and data pages are then retrieved sequentially. For inserts and deletes, the appropriate page is determined as for a search, and the record is inserted ordeleted with overflow pages added if necessary.

The following example illustrates the ISAM index structure. Consider the tree shown in Figure 4.16. All searches begin at the root. For example, to locate a record with the key-value 27, we start at the root and follow the left pointer, since 27 < 40. We then follow the middle pointer, since 20 <= 27 < 33. For a range search, we find the first qualifying data entry as for an equality selection and then retrieve primary leaf pages sequentially (also retrieving overflow pages as needed by following pointers from the primary pages). The primary leaf pages are assumed to be allocated sequentially this assumption is reasonable because the number of such pages is known when the tree is created and does not change subsequently under inserts and deletes and so no ‘next leaf page' pointers are needed.

We assume that each leaf page can contain two entries. If we now insert a record with key-value 23, the entry 23* belongs on the second data page, which already contains 20* and 27* and has no more space. We deal with this situation by adding an *overflow* page and putting 23* in the overflow page. Chains of overflow pages can easily develop.

For instance, inserting 48*, 41*, and 42* leads to an overflow chain of two pages. The tree of Figure 4.16 with all these insertions is shown in Figure 4.17.

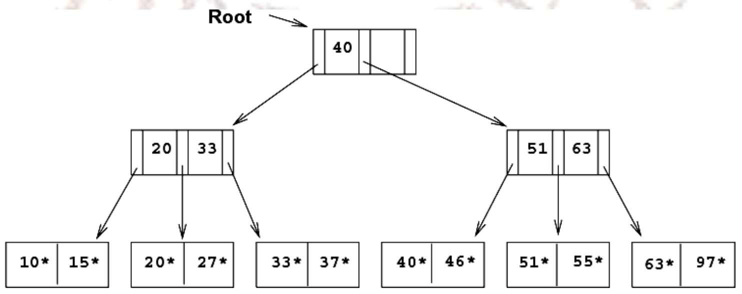


Fig 4.16: Sample ISAM Tree

The deletion of an entry *k** is handled by simply removing the entry. If this entry is on an overflow page and the overflow page becomes empty, the page can be removed. If the entry is on a primary page and deletion makes the primary page empty, the simplest approach is to simply leave the empty primary page as it is; it serves as a placeholder for future insertions (and possibly non-empty overflow pages, because we do not move records from the overflow pages to the primary page when deletions on the primary page create space).

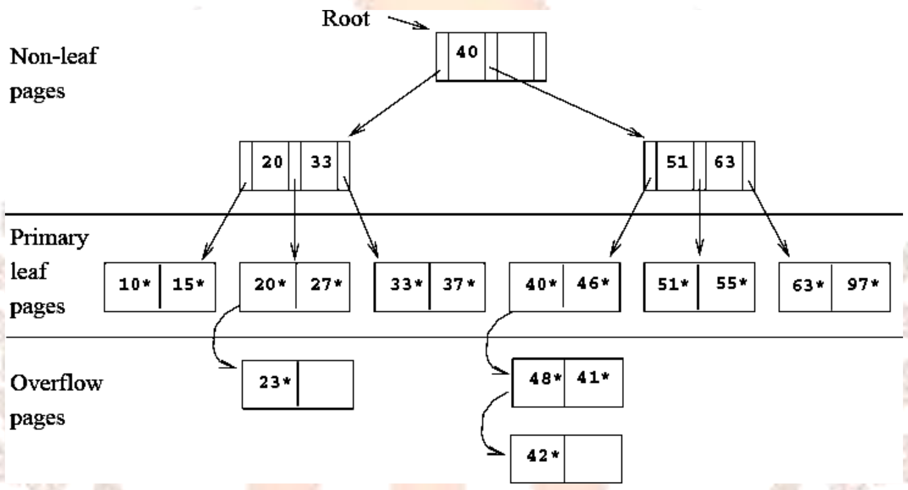


Fig 4.17: ISAM tree after inserts

Thus, the number of primary leaf pages is fixed at file creation time. Notice that deleting entries could lead to a situation in which key values that appear in the index, levels do not appear in the leaves! Since index levels are used only to direct a search to the correct leaf page, this situation is not aproblem. The tree of Figure 4.17 is shown in Figure 4.18 after the deletion of theentries 42*, 51*, and 97*.

Note that after deleting 51*, the key-value 51 continues to appear in the index level. A subsequent search for 51* would go to the correct leaf page and determine that the entry is not in the tree.

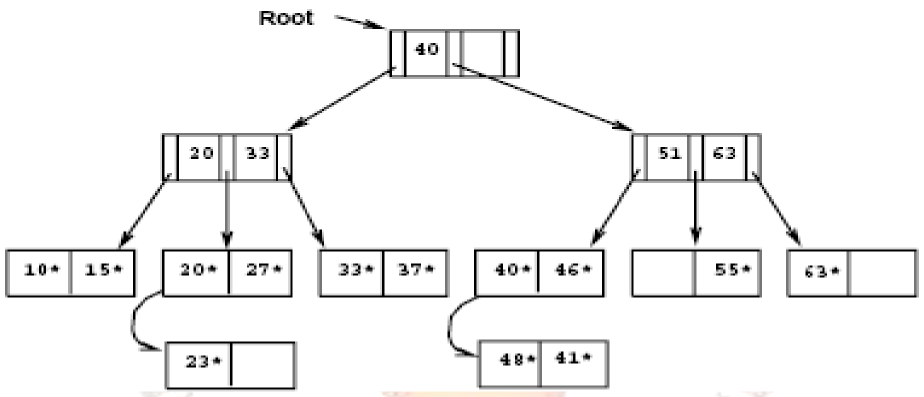


Fig 4.18: ISAM tree after deletes

The non-leaf pages direct a search to the correct leaf page. The number of disks I/Os is equal to the number of levels of the tree and is equal to *logFN*, where *N* is the number of primary leaf pages and the **fan-out** *F* is the number of children per index page. This number is considerably less than the number of disk I/Os for binary search, which is log_2N; in fact, it is reduced further by pinning the root page in memory. The cost of access viaa one-level index is $log_2(N=F)$. If we consider a file with 1,000,000 records, 10 records per leaf page, and 100 entries per index page, the cost (in page I/Os) of a file scan is 100,000, a binary search of the sorted data file is 17, a binary search of a one-level index is 10, and the ISAM file (assuming no overflow) is 3.

6. VIRTUAL STORAGE ACCESS METHOD (VSAM)

The term Virtual Storage Access Method (VSAM) applies to both a data set type and the access method used to manage various user data types. As anaccess method, VSAM provides much more complex functions than other disk access methods. VSAM keeps disk records in a unique format that is not understandable by other access methods.

You can use VSAM to organize records into **four types** of data sets: key-sequenced, entry-sequenced, linear, or relative record. The primarydifference among these types of data sets is the way their records are stored and accessed.

VSAM data sets

Key Sequenced Data Set (KSDS)

This type is the most common use for VSAM. Each record has one or more key fields and a record can be retrieved (or inserted) by key value. This provides random access to data. Records are of variable length.

Entry Sequenced Data Set (ESDS)

This form of VSAM keeps records in sequential order. Records can be accessed sequentially. It is used by IMS™, DB2®, and z/OS® UNIX®.

Relative Record Data Set (RRDS)

This VSAM format allows retrieval of records by number; record 1, record 2, and so forth. This provides random access and assumes the application program has a way to derive the desired record numbers.

Linear Data Set (LDS)

This type is, in effect, a byte-stream data set and is the only form of a byte-stream data set in traditional z/OS files (as opposed to z/OS UNIX files). A number of z/OS system functions use this format heavily, but it is rarelyused by application programs.

Several additional methods of accessing data in VSAM are not listed here. Most applications use VSAM for keyed data.

VSAM works with a logical data area known as a control interval (CI) that is diagrammed in Figure 4.19. The default CI size is 4K bytes, but it can be up to 32K bytes. The CI contains data records, unused space, record descriptor fields (RDFs), and a CI descriptor field.

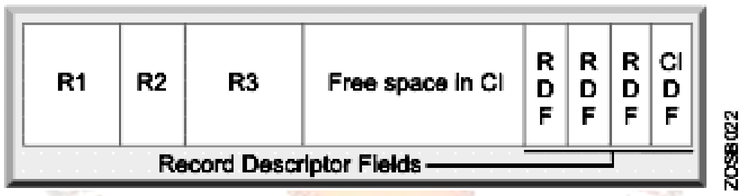


Fig 4.19: Simple VSAM control interval

Multiple CIs are placed in a control area (CA). A VSAM data set consists of control areas and index records. One form of index record is the sequence set, in which the lowest-level index is pointing to a control interval.

VSAM data is always variable-length and records are automatically blocked in control intervals. The RECFM attributes (F, FB, V, VB, U) do not apply to VSAM, nor does the BLKSIZE attribute. You can use the Access Method Services (AMS) utility to define and delete VSAM structures, such as files and indexes.

7. SUMMARY

This unit has covered storage devices, file organization, indexed, and virtual access methods which are used in DBMS. We discussed that several types of data storage media are used in the computer system and are classified bythe speed with which data can be accessed, by the cost per unit of data to buy the medium, and by the medium's reliability. We also discussed about the File organizations and ways of representing logical data models in terms of files. We also illustrated the ISAM index structure and the know-how of accessing files. Another accessing method discussed is Virtual Storage Access Method (VSAM) that applies to both a data set type and are used to manage various user data types. We also found out that VSAM keeps disk records in a unique format that is not understandable by other access methods.

8. TERMINAL QUESTIONS

1. Explain various storage devices and their characteristics.
2. Explain various performance measures of disks.
3. Explain various file organizations in detail.
4. Explain indexed and virtual storage access methods.

9. ANSWERS

Self-Assessment Questions

1. Cache
2. Lost
3. Power failure.
4. magnetic disk
5. optical
6. sequential
7. circular
8. disk controller
9. storage area network (SAN)
10. Access
11. average seek time
12. block
13. elevator algorithm

Terminal Questions

1. Various storage devices available in most computer systems are: main memory, flash memory, magnetic disk storage, optical storage, tape storage. Each of these storage devices has their own characteristics. (Refer section 4.2 for detail)
2. Various performance measures of disk are Access time, average seek time, capacity, latency time, data-transfer rate, reliability. (Refersection 4.2.3 for detail)
3. File are organized in Fixed-length records, Variable-length records. (Refer section 4.3 for detail)
4. In Indexed sequential access method (ISAM), index files are createdwith records in a page and pointer are used to do binary search for akey. In Virtual Storage access method, four types of data sets namely key-sequenced, entry-sequenced, linear, or relative record are used.