Record Storage and Primary File Organization

INTRODUCTION

The collection of data that makes up a computerized database must be stored physically on some computer storage medium.

Computer storage media form a storage hierarchy that includes two main categories:

- **Primary storage:** This category includes storage media that can be operated on directly by the computer central processing unit (CPU), such as the computer main memory and smaller but faster cache memories. Primary storage usually provides fast access to data but is of limited storage capacity.

- **Secondary storage:** This category includes magnetic disks, optical disks, and tapes. These devices usually have a larger capacity, cost less, and provide slower access to data than do primary storage devices. Data in secondary storage cannot be processed directly by the CPU; it must first be copied into primary storage.

SECONDARY STORAGE DEVICES

In this section we describe some characteristics of magnetic disk and magnetic tape storage devices.

Hardware Description of Disk Devices

Magnetic disks are used for storing large amounts of data. The most basic unit of data on the disk is a single **bit** of information. By magnetizing an area on disk in certain ways, one can make it represent a bit value of either 0 (zero) or 1 (one). To code information, bits are grouped into **bytes** (or characters). Byte sizes are typically 4 to 8 bits, depending on the computer and the device. We assume that one character is stored in a single byte, and we use the terms byte and character interchangeably. The **capacity** of a disk is the number of bytes it can store, which is usually very large. Small floppy disks used with microcomputers typically hold from 400 Kbytes to 1.5 Mbytes; hard disks for micros typically hold from several hundred Mbytes up to a few Gbytes; and large disk packs used with servers and mainframes have capacities that range up to a few tens or hundreds of Gbytes. Disk capacities continue to grow as technology improves.

Whatever their capacity, disks are all made of magnetic material shaped as a thin circular disk (Figure 1 a) and protected by a plastic or acrylic cover. A disk is single sided if it stores information on only one of its surfaces and double-sided if both surfaces are used. To increase storage capacity, disks are assembled into a disk pack (Figure 1 b), which may include many disks and hence many surfaces. Information is stored on a disk surface in concentric circles of small width,4 each having a distinct diameter. Each circle is called a track. For disk packs, the tracks with the same diameter on the various surfaces are called a cylinder because of the shape they would form if connected in space. The concept of a cylinder is important because data stored on one cylinder can be retrieved much faster than if it were distributed among different cylinders.

Sahaj Computer Solutions

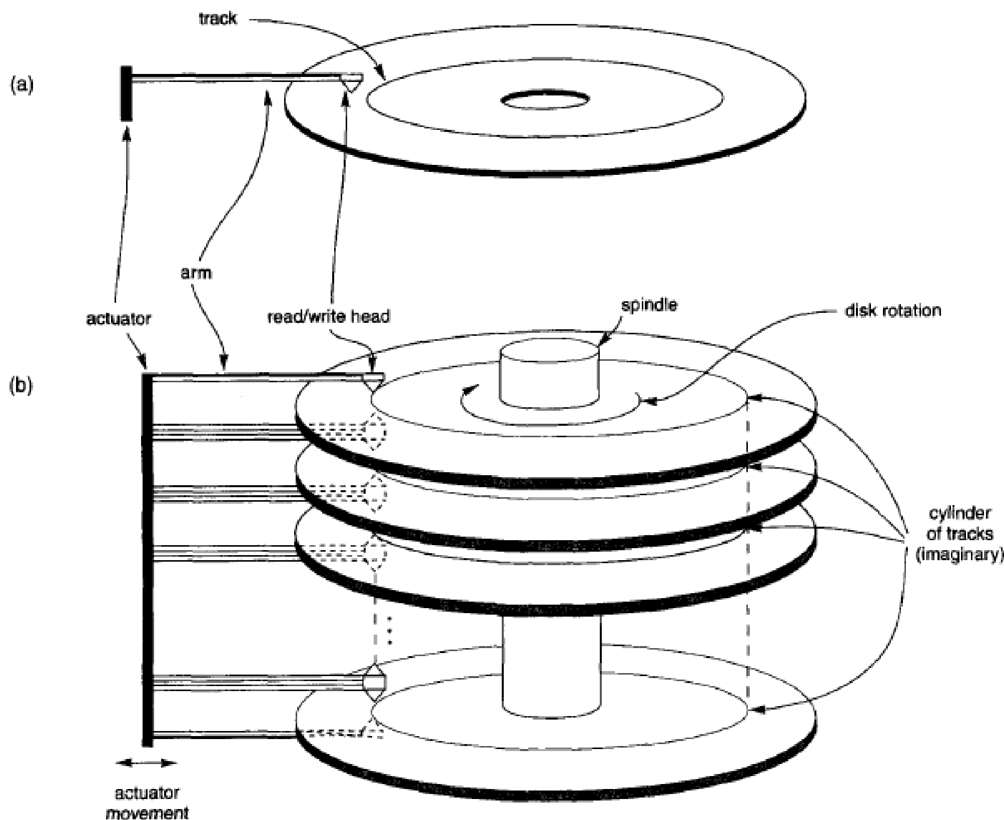


Fig1: (a)A single-sided disk with read/write hardware. (b) A disk pack with read/write hardware.

The number of tracks on a disk ranges from a few hundred to a few thousand, and the capacity of each track typically ranges from tens of Kbytes to 150 Kbytes. Because a track usually contains a large amount of information, it is divided into smaller blocks or sectors. The division of a track into **sectors** is hard-coded on the disk surface and cannot be changed.

The division of a track into equal-sized disk blocks (or pages) is set by the operating system during disk **formatting (or initialization).** Block size is fixed during initialization and cannot be changed dynamically. Typical disk block sizes range from 512 to 4096 bytes. Blocks are separated by fixed-size **interblock gaps**, which include specially coded control information written during disk initialization.

A disk is a random access addressable device. The **hardware address** of a block is a combination of a cylinder number, track number (surface number within the cylinder on which the track is located), and block number (within the track) is supplied to the disk i/o hardware. In many modern disk drives, a single number called **LBA (Logical Block Address)** which is a number between 0 and n (assuming the total capacity of the disk is n+1 blocks), is mapped automatically to the right block by the disk drive

controller. The address of a buffer-a contiguous reserved area in main storage that holds one block-is also provided. For a read command, the block from disk is copied into the buffer; whereas for a write command, the contents of the buffer are copied into the disk block.

Sometimes several contiguous blocks, called a **cluster**, may be transferred as a unit. The actual hardware mechanism that reads or writes a block is the disk **read/write head**, which is part of a system called a disk drive. A disk or disk pack is mounted in the disk drive, which includes a **motor** that rotates the disks. A read/write head includes an electronic component attached to a **mechanical arm**. Disk packs with multiple surfaces are controlled by several read/write heads-one for each surface (see Figure 1 b). All arms are connected to an **actuator** attached to another electrical motor, which moves the read/write heads in unison and positions them precisely over the cylinder of tracks specified in a block address.

Some disk units have fixed read/write heads, with as many heads as there are tracks. These are called fixed-head disks, whereas disk units with an actuator are called movable head disks. A disk controller, typically embedded in the disk drive, controls the disk drive and interfaces it to the computer system.

Magnetic Tape Storage Devices

Disks are random access secondary storage devices, because an arbitrary disk block may be accessed "at random" once we specify its address. Magnetic tapes are **sequential access** devices; to access the nth block on tape, we must first scan over the preceding n –1 block. Data is stored on reels of high-capacity magnetic tape, somewhat similar to audio or videotapes. A tape drive is required to read the data from or to write the data to a tape reel. Usually, each group of bits that forms a byte is stored across the tape, and the bytes themselves are stored consecutively on the tape. A read/write head is used to read or write data on tape. Data records on tape are also stored in blocks-although the blocks may be substantially larger than those for disks, and interblock gaps are also quite large.

The main characteristic of a tape is its requirement that we access the data blocks in sequential order. To get to a block in the middle of a reel of tape, the tape is mounted and then scanned until the required block gets under the read/write head. For this reason, tape access can be slow and tapes are not used to store online data, except for some specialized applications. However, tapes serve a very important function-that of backing up the database.

BUFFERING OF BLOCKS

When several blocks need to be transferred from disk to main memory and all the block addresses are known, several buffers can be reserved in main memory to speed up the transfer. While one buffer is being read or written, the CPU can process data in the other buffer.

Figure 2 illustrates how two processes can proceed in parallel. Processes A and B are running concurrently in an interleaved fashion, whereas processes C and Dare running concurrently in a parallel fashion. When a single CPU controls multiple processes, parallel execution is not possible. However, the processes can still run concurrently in an interleaved way. Buffering is most useful when processes can

Sahaj Computer Solutions

run concurrently in a parallel fashion, either because a separate disk I/0 processor is available or because multiple CPU processors exist.

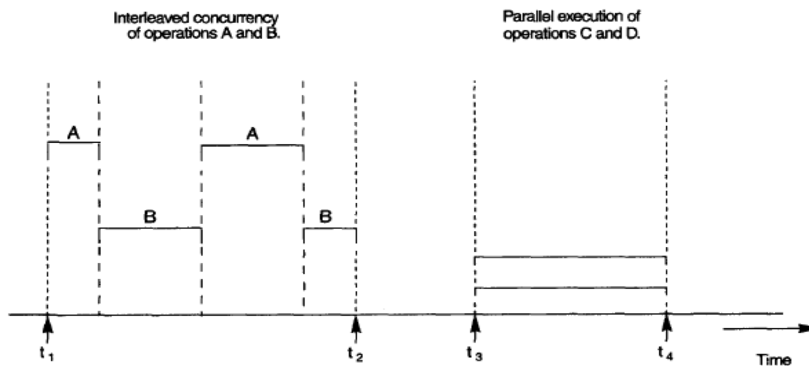


FIGURE 2: Interleaved concurrency versus parallel execution.

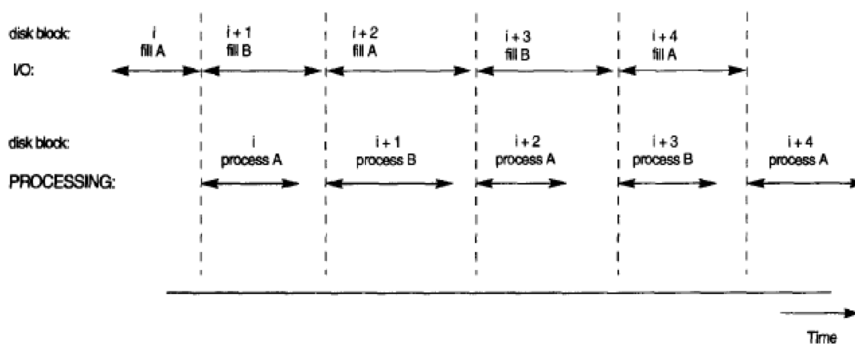


FIGURE 3 Use of two buffers, A and B, for reading from disk.

Figure 3 illustrates how reading and processing can proceed in parallel when the time required to process a disk block in memory is less than the time required to read the next block and fill a buffer. The CPU can start processing a block once its transfer to main memory is completed; at the same time the disk I/O processor can be reading and transferring the next block into a different buffer. This technique is called **double buffering** and can also be used to write a continuous stream of blocks from memory to the disk. Double buffering permits continuous reading or writing of data on consecutive disk blocks, which eliminates the seek time and rotational delay for all but the first block transfer. Moreover, data is kept ready for processing, thus reducing the waiting time in the programs.

PLACING FILE RECORDS ON DISK

Records and Record Types

Data is usually stored in the form of records. Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record. Records usually describe entities and their attributes. For example, an EMPLOYEE record represents an employee entity, and each field value in the record specifies some attribute of that employee, such as NAME, BIRTHDATE, SALARY, or SUPERVISOR. A collection of field names and their corresponding data types constitutes a record type or record format definition. A data type, associated with each field, specifies the types of values a field can take.

The data type of a field is usually one of the standard data types used in programming. These include numeric (integer, long integer, or floating point), string of characters (fixed-length or varying), Boolean (having 0 and 1 or TRUE and FALSE values only), and sometimes specially coded date and time data types. The number of bytes required for each data type is fixed for a given computer system. An integer may require 4 bytes, a long integer 8 bytes, a real number 4 bytes, a Boolean 1 byte, a date 10 bytes (assuming a format of YYYY-MM-DD), and a fixed-length string of k characters k bytes. Variable length strings may require as many bytes as there are characters in each field value.

Files, Fixed-length Records, and Variable-length Records

A file is a sequence of records. If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records. If different records in the file have different sizes, the file is said to be made up of variable-length records.

A file may have variable-length records for several reasons:

- The file records are of the same record type, but one or more of the fields are of varying size (variable-length fields). For example, the NAME field of EMPLOYEE can be a variable-length field.
- The file records are of the same record type, but one or more of the fields may have multiple values for individual records; such a field is called a repeating field and a group of values for the field is often called a repeating group.
- The file records are of the same record type, but one or more of the fields are optional; that is, they may have values for some but not all of the file records (optional fields).
- The file contains records of different record types and hence of varying size (mixed file).

The fixed-length record has the same fields, and field lengths are fixed, so the system can identify the starting byte position of each field relative to the starting position of the record. This facilitates locating field values by programs that access such files. Notice that it is possible to represent a file that logically should have variable-length records as a fixed-length records file.

Sahai Computer Solutions

Record Blocking and Spanned Versus Unspanned Records

The records of a file must be allocated to disk blocks because a block is the *unit of data transfer* between disk and memory. When the block size is larger than the record size, each block will contain numerous records, although some files may have unusually large records that cannot fit in one block. Suppose that the block size is B bytes. For a file of fixed-length records of size R bytes, with B ≥R, we can fit ***bfr* = |B/R |** records per block, where the |(x)| *(floor function) rounds down* the number x to an integer. The value *bfr* is called the **blocking factor** for the file. In general, R may not divide B exactly, so we have some unused space in each block equal to **B - *(bfr* * R) bytes.**

To utilize this unused space, we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record in case it is not the next consecutive block on disk. This organization is called **spanned**, because records can span more than one block. Whenever a record is larger than a block, we *must* use a spanned organization. If records are not allowed to cross block boundaries, the organization is called **unspanned**. This is used with fixed-length records having B > R because it makes each record start at a known location in the block, simplifying record processing. For variable-length records, either a spanned or an unspanned organization can be used.

Allocating File Blocks on Disk

There are several standard techniques for allocating the blocks of a file on disk. In contiguous allocation the file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast using double buffering, but it makes expanding the file difficult. In linked allocation each file block contains a pointer to the next file block. This makes it easy to expand the file but makes it slow to read the whole file. Another possibility is to use indexed allocation, where one or more index blocks contain pointers to the actual file blocks. It is also common to use combinations of these techniques.

File Headers

A file header or file descriptor contains information about a file that is needed by the system programs that access the file records. The header includes information to determine the disk addresses of the file blocks as well as to record format descriptions, which may include field lengths and order of fields within a record for fixed-length unspanned records and field type codes, separator characters, and record type codes for variable-length records.

OPERATIONS ON FILES

Operations on files are usually grouped into retrieval operations and update operations. The former do not change any data in the file, but only locate certain records so that their field values can be examined and processed. The latter change the file by insertion or deletion of records or by modification of field values. In either case, we may have to select one or more records for retrieval, deletion, or modification based on a selection condition (or filtering condition), which specifies criteria that the desired record or records must satisfy.

Sahai Computer Solutions

Actual operations for locating and accessing file records vary from system to system. Below, we present a set of representative operations.

- ***Open:*** Prepares the file for reading or writing. Allocates appropriate buffers to hold file blocks from disk, and retrieves the file header. Sets the file pointer to the beginning of the file.
- ***Reset:*** Sets the file pointer of an open file to the beginning of the file.
- ***Find* (or *Locate):*** Searches for the first record that satisfies a search condition. Transfers the block containing that record into a main memory buffer .
- ***Read (or Get):*** Copies the current record from the buffer to a program variable in the user program. This command may also advance the current record pointer to the next record in the file, which may necessitate reading the next file block from disk.
- ***FindNext:*** Searches for the next record in the file that satisfies the search condition. Transfers the block containing that record into a main memory buffer. The record is located in the buffer and becomes the current record.
- ***Delete:*** Deletes the current record and (eventually) updates the file on disk to reflect the deletion.
- ***Modify:*** Modifies some field values for the current record and (eventually) updates the file on disk to reflect the modification.
- ***Insert:*** Inserts a new record in the file by locating the block where the record is to be inserted, transferring that block into a main memory buffer, writing the record into the buffer, and (eventually) writing the buffer to disk to reflect the insertion.
- ***Close:*** Completes the file access by releasing the buffers and performing any other needed cleanup operations.

The preceding (except for Open and Close) are called **record-at-a-time** operations, because each operation applies to a single record. Additional set-at-a-time higher-level operations may be applied to a file. Examples of these are as follows:

- ***FindAll:*** Locates *all* the records in the file that satisfy a search condition.
- ***Find* (or *Locate) n:*** Searches for the first record that satisfies a search condition and then continues to locate the next n - 1 records satisfying the same condition.
- ***FindOrdered:*** Retrieves all the records in the file in some specified order.
- ***Reorganize:*** Starts the reorganization process.

FILES OF UNORDERED RECORDS (HEAP FILES)

In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a heap or pile file. Inserting a new record is *very efficient:* the last disk block of the file is copied into a buffer; the new record is added; and the block is then rewritten back to disk. The address of the last file block is kept in the file header. However, searching for a record using any search condition involves a linear search through the file block by block-an expensive procedure.

Sahaj Computer Solutions

To delete a record, a program must first find its block, copy the block into a buffer, then delete the record from the buffer, and finally rewrite the block back to the disk. This leaves unused space in the disk block. Deleting a large number of records in this way results in wasted storage space. Another technique used for record deletion is to have an extra byte or bit, called a deletion marker, stored with each record. A record is deleted by setting the deletion marker to a certain value. A different value of the marker indicates a valid (not deleted) record. Both of these deletion techniques require periodic reorganization of the file to reclaim the unused space of deleted records. During reorganization, the file blocks are accessed consecutively, and records are packed by removing deleted records.

We can use either spanned or unspanned organization for an unordered file, and it may be used with either fixed-length or variable-length records.

To read all records in order of the values of some field, we create a sorted copy of the file. Sorting is an expensive operation for a large disk file, and special techniques for external sorting are used.

For a file of unordered *fixed-length records* using *unspanned blocks* and *contiguous allocation,* it is straightforward to access any record by its position in the file. Such a file is often called a relative or direct file because records can easily be accessed directly by their relative positions.

FILES OF ORDERED RECORDS (SORTED FILES)

We can physically order the records of a file on disk based on the values of one of their fields-called the ordering field. This leads to an ordered or sequential file. If the ordering field is also a key field of the file-a field guaranteed to have a unique value in each record-then the field is called the ordering key for the file. Ordered records have some advantages over unordered files. First, reading the records in order of the ordering key values becomes extremely efficient, because no sorting is required. Second, finding the next record from the current one in order of the ordering key usually requires no additional block accesses, because the next record is in the same block as the current one (unless the current record is the last one in the block). Third, using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used, which constitutes an improvement over linear searches, although it is not often used for disk files.

To insert a record, we must find its correct position in the file, based on its ordering field value, and then make space in the file to insert the record in that position. For a large file this can be very time consuming because, on the average, half the records of the file must be moved to make space for the new record.

Reading the file records in order of the ordering field is quite efficient if we ignore the records in overflow, since the blocks can be read consecutively using double buffering.

Modifying a field value of a record depends on two factors: (1) the search condition to locate the record and (2) the field to be modified. If the search condition involves the ordering key field, we can locate the record using a binary search; otherwise we must do a linear search.

Ordered files are rarely used in database applications unless an additional access path, called a primary index, is used; this results in an indexed, sequential file.

HASHING TECHNIQUES

Another type of primary file organization is based on hashing, which provides very fast access to records on certain search conditions. This organization is usually called a **hash file**. The search condition must be an equality condition on a single field, called the **hash field** of the file. In most cases, the hash field is also a key field of the file, in which case it is called the hash key. The idea behind hashing is to provide a function *h*, called a **hash function** or randomizing function, which is applied to the hash field value of a record and yields the *address* of the disk block in which the record is stored.

Internal Hashing

For internal files, hashing is typically implemented as a hash table through the use of an array of records. Suppose that the array index range is from 0 to M - 1 then we have M slots whose addresses correspond to the array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and M - 1.One common hash function is the $h(K)$ = K mod M function, which returns the remainder of an integer hash field value K after division by M; this value is then used for the record address.

Other hashing functions can be used. One technique, called **folding**, involves applying an arithmetic function such as *addition* or a logical function such as *exclusive* or to different portions of the hash field value to calculate the hash address. Another technique involves picking some digits of the hash field value-for example, the third, fifth, and eighth digits-to form the hash address. The problem with most hashing functions is that they do not guarantee that distinct values will hash to distinct addresses, because the hash field space-the number of possible values a hash field can take-is usually much larger than the address space-the number of available addresses for records.

A **collision** occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called **collision resolution.** There are numerous methods for collision resolution, including the following:

• ***Open addressing:*** Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.

• ***Chaining:*** For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. In addition, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location.

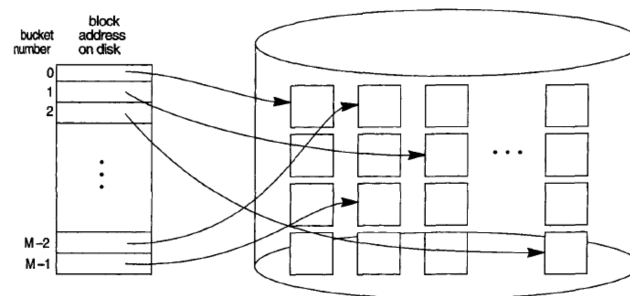
Sahai Computer Solutions

• ***Multiple hashing:*** The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

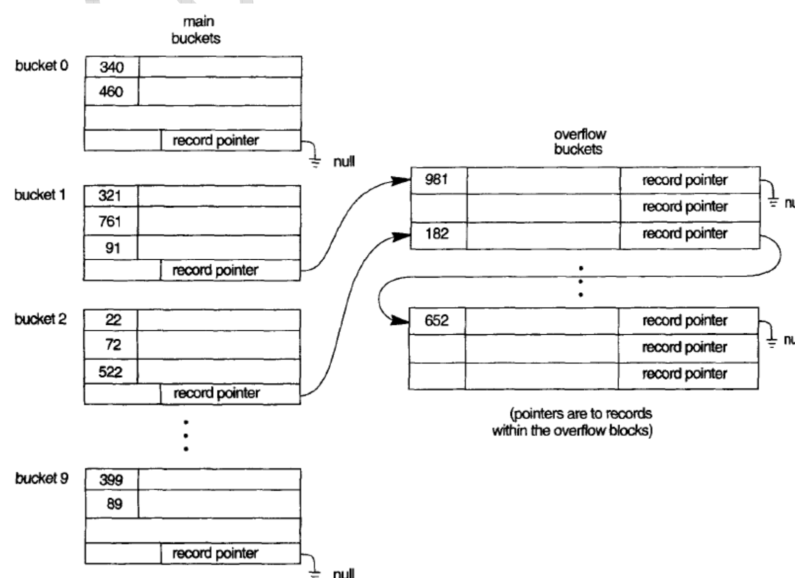
External Hashing for Disk Files

Hashing for disk files is called external hashing. To suit the characteristics of disk storage, the target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hashing function maps a key into a relative bucket number, rather than assign an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address, as illustrated in Figure .

.



The collision problem is less severe with buckets, because as many records as will fit in a bucket can hash to the same bucket without causing problems. However, we must make provisions for the case where a bucket is filled to capacity and a new record being inserted hashes to that bucket. We can use a variation of chaining in which a pointer is maintained in each bucket to a linked list of overflow records for the bucket, as shown in figure below.



The pointers in the linked list should be record pointers, which include both a block address and a relative record position within the block. Hashing provides the fastest possible access for retrieving an arbitrary record given the value of its hash field. Although most good hash functions do not maintain records in order of hash field values, some functions-called order preserving-do.

The hashing scheme described is called static hashing because a fixed number of buckets M is allocated. This can be a serious drawback for dynamic files. Suppose that we allocate M buckets for the address space and let m be the maximum number of records that can fit in one bucket; then at most (rn * M) records will fit in the allocated space. If the number of records turns out to be substantially fewer than (m* M), we are left with a lot of unused space. On the other hand, if the number of records increases to substantially more than (m * M), numerous collisions will result and retrieval will be slowed down because of the long lists of overflow records. In either case, we may have to change the number of blocks M allocated and then use a new hashing function (based on the new value of M) to redistribute the records. These reorganizations can be quite time consuming for large files. Newer dynamic file organizations based on hashing allow the number of buckets to vary dynamically with only localized reorganization.

When using external hashing, searching for a record given a value of some field other than the hash field is as expensive as in the case of an unordered file. Record deletion can be implemented by removing the record from its bucket. Modifying a record's field value depends on two factors: (1) the search condition to locate the record and (2) the field to be modified.

OTHER PRIMARY FILE ORGANIZATIONS

Files of Mixed Records

File organizations in object DBMSs, as well as legacy systems such as hierarchical and network DBMSs, often implement relationships among records as physical relationships realized by physical contiguity (or clustering) of related records or by physical pointers. These file organizations typically assign an area of the disk to hold records of more than one type so that records of different types can be physically clustered on disk. The concept of physical clustering of object types is used in object DBMSs to store related objects together in a mixed file. To distinguish the records in a mixed file, each record has-in addition to its field values-a record type field, which specifies the type of record. This is typically the first field in each record and is used by the system software to determine the type of record it is about to process. Using the catalog information, the DBMS can determine the fields of that record type and their sizes, in order to interpret the data values in the record.

B-Trees and Other Data Structures as Primary Organization

Other data structures can be used for primary file organizations. For example, if both the record size and the number of records in a file are small, some DBMSs offer the option of a B-tree data structure as the primary file organization.

Sahaj Computer Solutions

SUMMARY

Data on disk is stored in blocks; accessing a disk block is expensive because of the seek time, rotational delay, and block transfer time. Double buffering can be used when accessing consecutive disk blocks, to reduce the average block access time.

Records of a file are grouped into disk blocks and can be of fixed length or variable length, spanned or unspanned, and of the same record type or mixed types. File header describes the record formats and keeps track of the disk addresses of the file blocks. Information in the file header is used by system software accessing the file records.

Three primary file organizations were then discussed: unordered, ordered, and hashed. Unordered files require a linear search to locate records, but record insertion is very simple.

Ordered files shorten the time required to read records in order of the ordering field. The time required to search for an arbitrary record, given the value of its ordering key field, is also reduced if a binary search is used. However, maintaining the records in order makes insertion very expensive; thus the technique of using an unordered overflow file to reduce the cost of record insertion was discussed. Overflow records are merged with the master file periodically during file reorganization.

Hashing provides very fast access to an arbitrary record of a file, given the value of its hash key. The *most* suitable method for external hashing is the bucket technique, with one or more contiguous blocks corresponding to each bucket. Collisions causing bucket overflow are handled by chaining. *Access* on any non hash field is slow, and *so* is ordered access of the records on any field. We then discussed two hashing techniques for files that grow and shrink in the number of records dynamically-namely, extendible and linear hashing.

Finally, other possibilities for primary file organizations, such as B-trees, and files of mixed records, which implement relationships among records of different types physically as part of the storage structure

Sahaj Computer Solutions