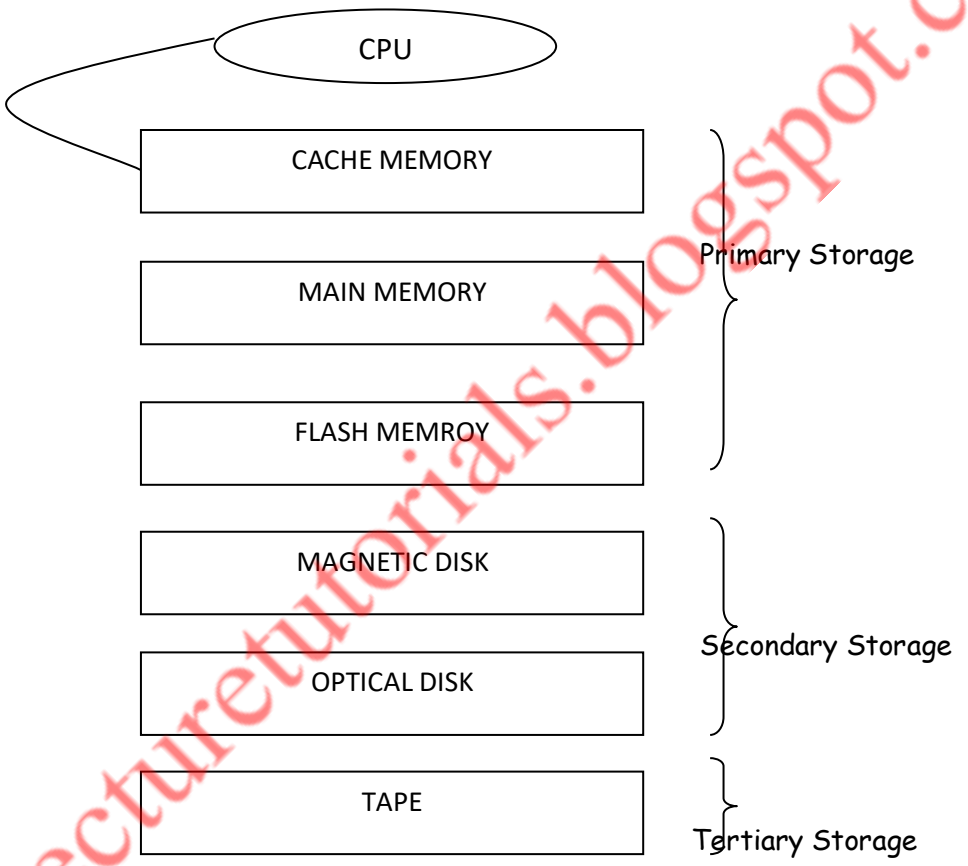
UNIT – 6 :: Storage and Indexing

Syllabus: Data on External Storage, File Organization and Indexing, Cluster Indexes, Primary and Secondary Indexes, Index data Structures, hash Based Indexing: Tree base Indexing, Comparison of File Organizations, Indexes and Performance Tuning.

Data on External Storage: The data stored on the database is very large that cannot accommodate in main memory and cannot store permanently. To store large volume of data permanently, an external storage devices were developed.

Different Kind of memory in a Computer System (or) Memory Hierarchy: Memory in a computer system is arranged in a hierarchy is shown as follows.



→ At the top, there is Primary storage that has cache, flash and main memory to provide very fast access the data.
→ The secondary storage devices are Magnetic disks that are slower and permanent devices.
→ The Tertiary Storage is a permanent and slowest device when compared with Magnetic disk.

Cache memory: The cache is the fastest but costliest memory available. It is not concern for databases.

Main Memory: The processor requires the data to be stored in main memory. Although main memory contains Giga byte of storage capacity but it is not sufficient for databases.

Flash Memory: Flash memory stores data even if the power fails. Data can be retrieved as fast as in main memory, however writing data to flash memory is a complex task and overwriting data cannot be done directly. It is used in small computers.

Magnetic Disk Storage: Magnetic disk is the permanent data storage medium. It enables random access of data and it is called "Direct-access" storage. Data from disk is transferred into main memory for processing. After modification, the data is loaded back onto the disk.

Optical Disk: Optical disks are Compact Disks (CDs), Digital Video Disks (DVDs). These are commonly used for permanent data storage. CD s are used for providing electronically published information and for distributing software such as multimedia data. They have a larger capacity that is upto 640 MB. These are relatively cheaper. To storage large volume of data CDs are replaced with DVDs. DVDs are in various capacities based on manufacturing.

Advantages:

1. Optical disks are less expensive.
2. Large amount of data can be stored.
3. CDs and DVDs are longer durability than magnetic disk drives.
4. These provide nonvolatile storage of data.
5. These can store any type of data such text data, music, video etc.

Tape Storage (or) Tertiary Storage media: Tape (or) Tertiary storage provides only sequential access to the data and the access to data is much slower. They provide high capacity removable tapes. They can have capacity of about 20 Giga bytes to 40 GB. These devices are also called "tertiary storage" or "off the storage". In a larger database system, tape (tertiary) storage devices are using for backup storage of data.

Magnetic tapes are fragmented into vertical columns referred as frames and horizontal rows referred as tracks. The data is organized in the form of column string with one data/frame. Frames are in turn fragmented into rows or tracks. One frame can store one byte of data and individual track can store a single bit. The rest of the tract is treated as a parity track.

Advantages:

1. Magnetic tapes are very less expensive and durable compared with optical disks.
2. These are reliable and a good tape drive system performs a read/write operation successfully.
3. These are very good choice for archival storage and any number of times data can be erased and reused.

Disadvantages:

The major disadvantage of tapes is that they are sequential access devices.

They work very slow when compared to magnetic disks and optical disks.

Performance Implications of Disk Structure:

1. Data must be in memory for the the DBMS to operate on it.

2. The unit for data transfer between disk and main memory is a block;if a single item on a block is needed,the entire block is transferred.Reading or writing a disk block is called an I/O(for input/output)operation.

3. The time to read or Write a block varies, depending on the location of the location of the data:

   Access time=seek time+rotational delay+transfer time

4.the time for moving blocks to or from disk usually dominates the time taken for database operations.To minimize this time, it is necessary to locate data records strategically on disk because of the geometry and mechanics of disks.

Buffer Manager: The buffer manager is the software layer that is responsible for bringing pages from physical disk to main memory as needed. The buffer manager manages the available main memory by dividing into a collection of pages, which we called as buffer pool. The main memory pages in the buffer pool are called frames.

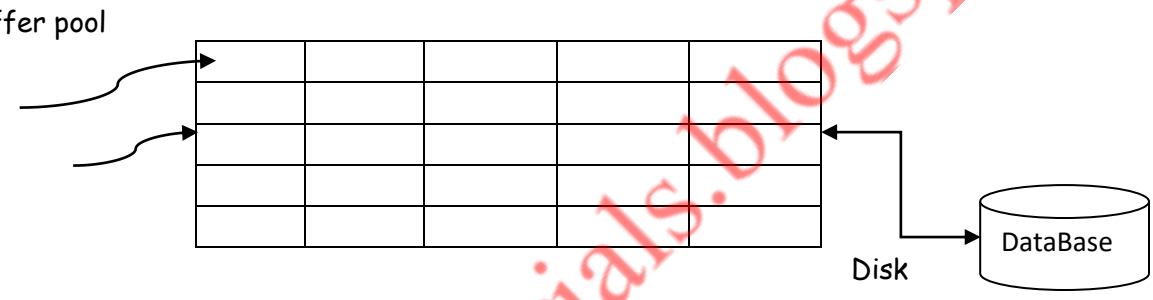
The goal of the buffer manager is to ensure that the data requests made by programs are satisfied by copying data from secondary storage devices into buffer. In fact, if a program performs an input statement, it calls the buffer manager for input operation to satisfy the requests by reading from existing buffers.

Similarly, if a program performs an output statement, it calls the buffer manager for output operation to satisfy the requests by writing to the buffers. Therefore, we can say that input and output operations occurs b/w the program and the buffer area only.

In addition to the buffer pool itself, the buffer manager maintains two variables for each frame in the pool. They are '**pin-count**' and '**dirty**'. The number of times the page is requested in the frame, each time the pin-count variable is incremented for that frame (i.e. set to 1 (pin-count=1)). For satisfying each request the pin-count is decreased for that frame (i.e. set to 0).

Thus, if a page is requested the pin-count is incremented, if it fulfills the request the pin-count is decremented.

In addition to this, if the page has been modified the Boolean variable, 'dirty' is set as 'on'. Otherwise set to 'off'.



→ **Buffer Manager Writing the page to disk:** When a page is requested the buffer manager does the following.

1. Checks the buffer pool that frame contains the requested page and if so, increment the pin-count of that frame. If the page is not in the pool, the buffer manager brings it into the main memory from disk and set the pin-count value as 1. Otherwise set to 0.
2. If the 'dirty' variable is set to 'on' then that page is modified and replaced by its previous page and writes that page on to the disk.

→ **Pinning and Unpinning of pages**: In buffer pool if some frame contains the requested page, the pin-count variable is set to 1 of that frame. So, pin-count = 1 is called **pinning** the requested page in its frame. When the request of the requestor is fulfilled, the pin-count variable is set to 0 of that frame. Thus, the buffer manager will not read any page into a frame unit when pin-count becomes 0 (zero).

→ **Allocation of Records to Blocks**: The buffer management uses block of storage and it is replaced by next record allocation when current record is deleted. The buffer manager also provides concurrency control system to execute more than one process. In this case, the records are mapped onto disk blocks.

File Organization and Indexing:

File Organization: A file is organization is a method of logically arranging the records in file on the disk. These records are mapped onto disk blocks.

In DBMS, the file of records is an important. That is, create a file, destroy it and also can insert records into it, delete from it. It supports scan also. The most important and widely used storage technique is Heap file. A Heap file is the simplest file structure. In a heap file, records are stored in random order across the pages of the file.

Thus, a file organization can be defined as the process of arranging the records in a file, when the file is stored on disk.

Types of File Organizations: Data is organized on secondary storage in terms of files. Each file has several records.

The enormous data cannot be stored in main memory so, it is stored on magnetic disk. During the process, the required data can be shifted to main memory from disk. The unit of information being transferred b/w main memory and disk is called a page.

Tapes are also used to store the data in the database. But this can be accessed sequentially. So, most of the time is wasted to transfer each page.

Buffer manager is software used for reading data into memory and writing data onto magnetic disks. Each record on a file is identified by record id or rid. Whenever a page needs to be processed, the buffer manager retrieves the page from the disk based on its record id.

Disk space manager is software that allocates space for records on the disk. When DBMS requires an additional space, it calls disk space management to allocate the space. DBMS also informs disk space manager when it's not going to use the space.

Most widely used file organizations are 1. Heap (Unordered) Files. 2. Sequential (ordered) files. 3. Hash files.

1. **Heap file:** It is the simplest file organization and stores the files in the order they arrive. It also called unordered file.
   →**Inserting a record:** Records are inserted in the same order as they arrive.
   →**Deleting a record:** The record is to be deleted, first access that record and then marked as deleted.
   →**Accessing a Record:** A linear search is performed on the files starting from the first record until the desired record is found.
2. **Ordered File:** Files are arranged in sequential order. The main advantage of this file organization is that we can now use binary search as the file are sorted.
   →**Insertion of Record:** This is a difficult task. Because, first we need to identify the space where record need insert and file is arranged in an order. If the space is available then record can directly be inserted. If space is not sufficient, then that record moved to next page.
   → **Deletion of Files:** This task is also difficult to delete the record. In this first find deleted record and then remove the empty space of deleted record.
   →**Access of files:** This is similar as we can use binary search on files.
3. **Hash files:** Using this file organization, files are not organized sequentially, instead they are arranged randomly. The address of the page where the record is to be stored is calculated using a 'hash function'.

Index: An index is a data structure which organizes data records on disk to optimize certain kinds of retrieval operations. Using an index, we easily retrieve the records which satisfy search conditions on the search key fields of the index. The 'data entry' is the term which we use to refer to the records stored in an index file. We can search an index efficiently for finding desired data entries and use them for obtaining data records. There are three alternatives for to store a data in an index,

1) A data entry K* is the actual data record with search key value of k
2) A data entry is a <k, rid> pair (Here rid is the row id or record id and k is key value).
3) A data entry is a <k, rid-list> pair (rid-list is a list of record ids of the data records with search key value k).

Types of Index: There two index techniques to organize the file. They are 1. Clustered. 2. Un-clustered.
Clustered: A file organization in which the data records are ordered in same way as data entries in index is called clustered.

Un-clustered: A file organization in which the data records are ordered in a different way as data entries in index is called un-clustered.

Indexed Sequential files : Indexed sequential file overcomes the disadvantages of sequential file, where in it is not possible to directly access a particular record. But, in index sequential file organization, it is possible to access the record both sequentially and randomly. Similar to sequential file, the records in indexed sequential file are organized in sequence based on primary key values. In addition to this, indexed sequential file consists of the following two features that distinguishes it from a sequential file.

i)**Index**: It is used so as to support random access. It provides a lookup capability to reach quickly to the desired record.

ii)**Overflow file**: The overflow file is similar to the log file used in the sequential file. Indexed sequential file greatly reduces the time required to access a single record without sacrificing the sequential nature of the file. In order to process a file sequentially, the records of the main file are initially processed in a sequence until a pointer to the overflow file is found. Then accessing continues in the overflow file until a null pointer is encountered.

Hash File Organization: Hash file organization helps us to locate records very fast with a given search key value. For example, "Find the sailor record for "SAM", if the file is hashed on name field. In hashed files, the pages are grouped into bucket. Every bucket has bucket number which allows us to find the primary page for that bucket. Then the record belonging to that bucket can be determined by using hash function to the search fields, when inserting the record into the appropriate bucket with overflow pages are maintained in a linked list. For searching a record with a given search key value, apply the hash function to identify the bucket to which such records belongs and look at. This organization is called static hashed file.

Fixed-Length File Organization: A file is organized logically as a sequence of records. These records are mapped onto disk block.

One approach to mapping the database to files is to use several files and to store records of only one fixed length in any given file. An alternative is to structure our files so that we can accommodate multiple lengths for records, however, files of fixed length records are easier to implement than the files of variable-length records.

Fixed-length Records: As an example, consider a file of account records for our bank database. Each record of this file is defined as

```
Type deposit = record
        Account_number      : char(10);
        Branch_name         : char(20);
        Balance             : real;
    End
```

→ In the above definition, each character occupies 1 byte and a real occupies 8 bytes. So totally the Above record occupies 38 bytes long.

Record 1
Record 2
Record 3
Record 4
Record 5
Record 6
Record 7

Acc.no	branch name	balance
101	ongole	2000
205	chimakurthy	3500
301	kandukur	4300
102	ongole	2200
222	chimakurthy	1200
333	kandukur	2600
343	kandukur	2200

→There is a problem to delete a record from this structure. The space occupied by 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. This is shown in fig.

Record 1

Acc.no	branch name	balance
101	ongole	2000
301	kandukur	4300
102	ongole	2200
222	chimakurthy	1200

The main disadvantages of this is when one record to be deleted then entire database need to modify why because after deletion of record that space is occupied by its next record and so on.

Record 3
Record 4
Record 5
Record 6
Record 7

To rectify this drawback, DBMS provided a facility that deleted record space is replaced by last record. This is shown in fig.

	Acc.no	branch name	balance
Record 1	101	ongole	2000
Record 7	343	kandukur	2200
Record 3	301	kandukur	4300
Record 4	102	ongole	2200
Record 5	222	chimakurthy	1200
Record 6	333	kandukur	2600

On insertion of a new record, we use the record pointer by the header. We change the head pointer to point to the next available record. If no space is available, we add the new record to the end of the file.
Thus, Insertion and deletion for files of fixed-length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record.

Byte-String Representation: A simple method for implementing variable-length records is to attach a special end-of-record ( ) symbol to the end of each record. We can then store each record as a string of consecutive bytes. This is shown in fig.

Acc_name	accno	amt	aacno	amt	accno	amt	
suman	1001	4000	2001	9000	2310	7000	
Sreenivaas	3001	5000					
Kalyan	1201	2300					
Kiran kumara	3021	5000	3233	2200			
Sowjanya	3201	5500					
Sravani	2301	2500					

The byte-string representation has some disadvantages:

1) It is not easy to reuse space occupied by a deleted record.
2) There is no space, for records to grow longer. If a variable-length record becomes longer, it must be moved, movement is costly if pointers to the records are stored elsewhere in the database, since the pointers must be located and updated.

Thus, the basic byte-string representation described here not usually used for implementing variable-length records. However, a modified form of the byte-string representation, called the slotted-page structure, is commonly used for organizing records within a single block. This is shown in following structure.

size

# Entries							

Location

→ There is a header at the beginning of each block, containing following information,

1) The header contains number of record entries.
2) The end of free space in the block
3) In an array, entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, b/w the final entry in the header array and the first records. 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 delete. When a record is to be deleted then space is replaced by final entry record.

Fixed-Length Representation: The Fixed-Length representation is another way to implement variable-length records efficiently in a file system to use one or more fixed-length records.

There are two ways to do this, 1) Reserved Space 2) List Representation

1) Reserved Space: If there is a maximum record length that is never exceeded, we can use fixed-length records of that length. Unused space is filled with a special null, or end-of-record, symbol. This is shown in following figure (1).

2) List Representation: We can represent variable-length record by lists of fixed-length records, chained together by pointer. This is shown in fig(2).

fig(1)

suman	1001	4000			
ramu	2201	4400			
Mamatha	3051	3400			
Sumathi	4011	2400			
swapna	3331	4200			
kalyan	6001	4000			

2)

Suman	1001	4000	
Ramu	2201	4400	
Mamatha	3051	3400	
Kalyan	6001	4000	

- The **reserved-space** method is useful when most records have a length close to the maximum. Otherwise, a specified amount of space may be wasted.
-
- The **List Representation** method is useful when an account of bank customer has more accounts in other branches. So, this method add a pointer field to represent the file when an accountant having more accounts in different branches.

→ The disadvantage of the structure is that we waste space in all records except the first in a chain. The first method needs to have the branch_name, but subsequent records do not.

- To overcome this drawback, it allows two kind of blocks in file.
  →**Anchor-block**: Which contain the first records of a chain.

→ **Overflow-block:** Which contain records other than those that are the first records of a chain.

Thus, all records within a block have the same length, even though not all records in the file have the same length.

Types of Indexing: Indexes can perform the DBMS. By the index can locate the desired record directly without scanning each record in the file.

An index can be defined as a data structure that allows faster retrieval of data. Each index is based on certain attribute of the field. This is given in 'search key'.

An index can refer the data based on several search keys. It means, the term data entry refer to the records stored in an index file. The data entry can be a,

1) Search Key. 2) Search key with record id. 3) Search key with list of record ids).

- A file organization when the records are stored and referred in same way as data entries in index is called "**clustered**".
- A file organization when the records are stored and referred in a different way as data entries in index is called "**un-clustered**".

Differences b/w clustered and un-clustered:

→ A file organization when the records are stored and referred in same way as data entries in index is called clustered. Whereas un-clustered referred refer in a different way as data entries in index is called un-clustered.

→ A clustered index is an index which uses alternative (1) whereas un-clustered index uses alternative (2) and (3).

→ Clustered index refer few pages when we require retrieving the records. Whereas un-clustered index refer several pages when we require retrieving the records.

→ If a file contains the records in sequential order, the index search key specifies the same order to retrieve the records from the sequential order of the file is called clustered index. Whereas the index search key specifies the different order to retrieve the records from the sequential order of the file is called un-clustered index.

Primary Index and Secondary Index:

Primary Index: A primary index is an index on a set of fields that includes the primary key is called a primary index. A primary index is an ordered file whose records are of fixed length with two fields. The first field of the record of file must be defined with a constraint "primary key" is called the primary key of the data file and the second field is a pointer to a disk block.

There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer of that block as its two field values. The two field values of index entry are referred to as key[i], primary key[i].

Primary indexes are further divided into **dense index** and **sparse index**.

1) Dense Index: An index record appears for every search key value in the file. The index record contains the search-key value and a pointer to the first record with that search-key.

2) Sparse Index: An index record is created for only some of the values. As it is true in dense indexes, each index record contains a search-key value and a pointer to the first data record with the largest search-key value that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry and follow the pointers in the file, until we find the desired record.

Secondary Index: An index that is not a primary key index is called a secondary index. That is an index on a set of fields that does not include the primary key is called a secondary index. A secondary index on a candidate key looks just like a dense primary index, except that the records pointed to successive values in the index are not stored sequentially. In general, secondary indices are different from primary indices. It the search key of a primary index in not a primary key, it suffices the index pointing to the first record with a particular value for the search key, since the other records can be fetched by a sequential scan of the file.

A secondary index must contain pointers to all the records, because if the search ey of a secondary index is not a primary key, it is not enough to point to just the first record.

Thus, the records are ordered by the search key of the primary index but same search-key value could be anywhere in the file.

4000	
3400	
2240	
4500	
4050	
4200	
4500	
4050	

kalyan	1001	4000
jaishnav	1022	3400
priyanka	2301	2240
kishore	1001	4500
sumahi	4001	4050
chaitanya	3030	4200
Anjali	1001	4500
swapna	4001	4050
mamatha	3030	4200

The above fig. shows the structure of a secondary index that uses an extra level of indirection on the account file, on the search key balance.

A sequential scan in primary index is efficient because records in the file are stored physically in the same order as the index order. We cannot store a file physically ordered both by the search key of the primary index and the search key of a secondary index. Because secondary-key order and physical-key order differ, but if we attempt to scan the file sequentially in secondary-key order, the reading of each record is likely to require the reading of a new block from disk. If a secondary index stores only some of the search key values, records with intermediate search key values may be anywhere in the file and in general, we cannot find them without searching the entire file. Secondary indices must therefore be dense, with an index entry for every search-key value and a pointer to every record in the file but not the sparse.

Secondary indices improve the performance of queries that use keys other than the search key of the primary index. They also impose a significant overhead on modification of the database. The design of a database decides which secondary indices are desirable on the basis of an estimate of the relative frequency of queries and modifications.

Index Data Structures: The two methods in which file data entries can be organized in two ways.

1) Hash-based indexing, which uses search key

2) Tree-based indexing, it refers to the process of

i) Finding a particular record in a file using one or more index or indexes.

ii) Strong a record in any order (randomly on the disk).

1) Hash Based Indexing: This type of indexing is used to find records quickly, by providing a search key value.

In this, a group of file records stored in pages based on bucket method. The first bucket contains a primary page and along with other pages is chained together. In order to determine the bucket for a record, a special function is called a hash function along with a search key is used. By providing a bucket number, we can obtain the primary page in one or more disk I/O operations.

Records Insertion into the Bucket: The records are inserted into the bucket by assigning (allocating) the need "overflow" pages.

Record Searching: a hash function is used to find first, the bucket containing the records and then by scanning all the pages in a bucket, the record with a given search key can be found.

Suppose, if the record doesn't have search key value then all the pages in the file needs to be scanned.

Record Retrieval: By applying a hash function to the record's search key, the page containing the needed record can be identified and retrieved in one disk I/O.

Consider a file student with a hash key rno. Applying the hash function to the rno, represents the page that contains the needed record. The hash function 'h' uses the last two digits of the binary value of the rno as the bucket identifier. A search key index of marks obtained i.e., marks contains <mrks, rid> pairs as data entries in an auxiliary index file which is shown in the fig. The rid (record id) points to the record whose search key value is mrks.

2) Tree-based indexing: In Tree Based indexing the records arranged in tree-like structure. The data entries are started according to the search key values and they are arranged in a hierarchical number to find the correct page of the data entries.

Examples:

1) Consider the student record with a search key rno arranged in a tree-structured index. In order to retrieve the nodes (A'_1, B'_1, L'_{11}, L'_{12} and L'_{13}) that need to perform disk I/O.

The lowest leaf level contains these records. The additional records with rno's < 19 and > 42 are added to the left side of the leaf node L'_{11} and to the right of the leaf node L'_{13}.

The root node is responsible for the start of search and these searches are then directed to the correct leaf pages by the non-leaf pages which contain node pointers separated by the search key values. The data entries in a subtree smaller than the key value k_i are pointed to by the right node pointer of k_i, shown in fig.

2)In order to find the students whose roll numbers lies b/w '19' and '24', the direction of the search is shown in the fig.

Suppose we want to find all the students roll numbers lying b/w 17 and 40, we first direct the search to the node A'_1 and after analyzing its contents, then forwarded the search to B'_1 followed by the leaf node L'_{11}, which actually contains the required data entry. The other leaf nodes L'_{12} and L'_{13} also contains the data entries that fulfills our search criteria. For this, all the leaf pages must be designed using double linked list.

Thus, L'_{12} can be fetched using next pointer on L'_{11} and L'_{13} can be obtained using the next pointer on L'_{12}. Number of disk I/Os = Length of the path from the root to a leaf (occurs in search) + The number of satisfying data entries leaf pages.

Closed and Open Hashing: File organization based on the technique of hashing allows us to avoid accessing an index structures. Hashing also provides a way of constructing indexes. There are two types of hashing techniques. They are,

1) Static/ Open hashing. 2) Dynamic/Closed hashing.

In a hash file organization, we obtain the address of the disk block containing a desired record directly by computing a function on the search key value of the record. In our description of hashing, we shall use the term bucket to denote a unit of storage that can store one or more records. A bucket is typically a disk block, but could be chosen to be smaller or larger then a disk block.

Static hashing can obtain the address of the disk block containing a desired record directly by computing hash function on the search key value of the record. In static hashing, the number buckets is static (fixed). The static hashing scheme is illustrated as shown in fig.

The pages containing the index data can be viewed as a collection of buckets, with one page and possible additional overflow pages for overflow buckets. A file consists of buckets 0 through N – 1 for N buckets. Buckets contain data entries which can be any of the three choices K*, < k, rid > pair, <k, rid-list> pair.

To search for a data entry, we apply a hash function 'h' to identify the bucket to which it belongs and then search this bucket.

To insert a data entry, we use the hash function to identify the correct bucket and then put the data entry there. If there is no space for this data entry, we allocate a new overflow bucket, put the data entry and add to the overflow page.

To delete a data entry, we use the hash function to identify the correct bucket, locate the data entry by searching the bucket and then remove it. If this data entry is the last in an overflow page, the overflow page is removed and added to a list of free pages.

Thus, the number of buckets in a static hashing file is known when the file is created the pages can be stored as successive disk pages.

Drawbacks of Static Hashing:

→The main problem with static hashing is that the number of buckets is fixed.

→ If a file shrinks greatly, a lot of space is wasted.

→ If a file grows a lot, long overflow chains develop, resulting in poor performance.

Dynamic Hashing: The dynamic hashing technique allow the hash function to be modified dynamically to accommodate the growth or shrinkage of the database, because most databases grow larger over time and static hashing techniques presents serious problems to deal with them.

Thus, if we are using static hashing on such growing databases, we have three options:

1) Choose a hash function based on the current file size. This option will result in performance degradation as the database grows.
2) Choose a hash function based on predicted size of the file for future. This option will result in the wastage of space.
3) Periodically reorganize the hash structure in response to file growth.

Thus, using dynamic hashing techniques is the best solution. They are two types,

1. **Extensible Hashing Scheme:** Uses a directory to support inserts and deletes efficiently with no overflow pages.
2. **Linear Hashing Scheme:** Uses a clever policy for creating new buckets and supports inserts and deletes efficiently without the use of a directory.

Comparison of File Organization: To compare file organizations, we consider the following operations that can be performed on a record. They are,

1) **Record Insertion:** For inserting a record we need to identify and fetch the page from the disk. The record is then added and the (modified) page is written back to the disk.
2) **Record Deletion:** It follows the same procedure as record insertion except that after identifying and fetching a page, the record with the given rid is deleted and again the changed page is added back to the disk.
3) **Record Scanning:** In this all the file pages must be fetched from the disk and are stored in a pool of buffers. Then the corresponding record can be retrieved.
4) **Record Searching Based on Equality Selection:** In this, all the records that satisfies a given equality selection criteria are fetched from the disk.
   Example: To find a student record based on the following equality selection criteria "student whose roll number (rno) is 15 and whose marks (mrks) are 90* is the topper of the class.
5) **Record Searching Based on Selected Range:** In this, all the records that satisfies a given equality selection are fetched.
   Example: Find all the records of the students whose secured marks are greater than 50.

Cost Model in terms of time needed for execution:

It is a method used to calculate the costs of different operations that are performed on the database.

Notations:

B = The total number of pages without any space wastage when records are grouped into it.
R = The total number of records present in a page.
D = The average time needed to R/W (Read/Write) a disk page.
C = The average time needed for a record processing
H = Time required to apply the hash function on a record in hashed-file organization.
F = Fan-out (in tree indexes)

For calculating I/O costs (which is the base for costs of the database operations) we take, D = 15 ms, C and H = 100 ns.

Heap Files:

1) Cost of Scanning: The cost of scanning heap files is given by B(D + RC). It means, Scanning R records of B pages with time C per record would take BRC and scanning B pages with time D per page would take BD. Therefore the total cost of scanning is BD + BRC ➔ B(D + RC)

2) Cost of Insertion: The cost to insert a record in heap file is given as 2D + C. It means, to insert a record, first we need to fetch the last page of the file that can take time 'D' then we need to add the

record that takes time 'C' and finally the page is written back to the disk from main memory will take time 'D'. So, the total cost is, D + D + C ➔ 2D + C.

3) Cost of Deletion: The cost to delete a record from a heap file is given as, D + C + D = 2D + C. It means, in order to delete a record, first search the record by reading the page that can take time

4) Record Searching based on some quality criteria: Searching exactly one record that meet the equality that involves scanning half of the files based on the assumption to find the record.
This takes time = $\frac{1}{2}$ x scanning cost ➔ $\frac{1}{2}$ x B (D + RC).
In case of multiple records the entire file need to be scanned.

5) Record searching with a Range selection: It is the same as the cost of scanning because it is not known in advance how many records can satisfy the particular range. Thus, we need to scan the entire file that would take B(D + RC).

Sorted Files:

1) Cost of scanning: The cost of scanning sorted files is given by B(D + RC) because all the pages need to be scanned in order to retrieve a record. i.e. cost of scanning sorted files = Cost of scanning heap files.

2) Cost Insertion: The cost of insertion in sorted files is given by search cost + B(D + RC).
It includes, Finding correct position of the record + Adding of record + Fetching of pages + rewriting the pages.

3) Cost of Deletion: The cost of deletion in sorted files is given by
Search cost + B(D + RC).
It includes, Searching a record + removing record + rewriting the modified page.
Note: The record deletion is based on equality.

4) Cost of Searching with equality selection Criteria: This cost of sorted files is equal to D $\log_2$ B = It is the time required for performing a binary search for a page that contain the records.
If many records satisfy, then record is equal to, D $\log_2$ B + C $\log_2$ R + Cost of sequential reading of all the records.

5) Cost of Searching with Range Selection: This cost is given as,
Cost of fetching the first matching record's page + Cost of obtaining the set of qualifying records.

If the range is small, then a single page contain all the matching records, else additional pages needs to be fetched.

Clustered Files:

1) Cost of Scanning: The cost of scanning clustered files is same as the cost of scanning sorted files except that it has more number of pages and this cost is given as scanning B pages with time 'D' per page takes BD and scanning R records of B pages with time C per record takes BRC. Therefore the total cost is, 1.5B(D + RC).

2) Cost of Insertion: The cost of insertion in clustered files is, Search + Write (D $\log_F$1.5B + C$\log_2$R) + D.

3) Cost of Deletion: It is same as the cost of insertion and includes,
the cost of searching for a record + removing of a record + rewriting the modified page.
i.e. D $\log_F$ 1.5B + C$\log_2$R + D

4) Equality Selection Search:
i) For a single Qualifying Record: The cost of finding a single qualifying record in clustered files is the sum of the binary searches involved in finding the first page in D $\log_F$ 1.5B and finding the first matching record in C $\log_2$ R. i.e. D $\log_F$1.5B + C $\log_2$ R.

ii) For several Qualifying Records: If more than one record satisfies the selection criteria then they are assumed to be located consecutively.

Cost required to find record is equal to,

$D \log_F 1.5B + C \log_2 R$ + cost involved in sequential reading of all records.

5) Range Selection Search: This cost in an equality search under several matched records.

Heap File with Un-clustered Tree Index:

1) Scanning: For scanning a student's file,

i) Scan the index's leaf level.

ii) Get the relevant record from the file for each data entry.

iii) Obtain sorted data records according to < rno, mrks >.

The cost of reading all the data entries is 0.15B (D + 6.7RC) I/Os. For each index entry a record has to be fetched in one I/O.

2) Insertion: The record is first inserted at 2D + C in students heap file and the associated entry in the index. The correct leaf page can be found in $D \log_F 0.15B + C \log_2 6.7 R$ followed by the addition of a new entry and rewriting in D.

3) Deletion: The cost of deletion includes,

Cost of finding the record in a file + cost of finding the entry in index + Cost of rewriting the modified page in the index and the file.

It corresponds, $D \log_F 0.15B + C \log_2 6.7R + D + 2D$.

4) Equality Selection Search: The cost involved in this operation is the sum of,

i) The cost of finding the page containing a matched entry.

ii) The cost of finding the first matched entry and

iii) The cost of finding the first matched record.

It is given as, $D \log_F 0.15B + C \log_2 6.7R + D$

5) Range Selection Based-search: This is same as search with range selection in clustered files except from having data pages it has data entries.

Heap file with Un-clustered Hash Index:

1) Scan : The total cost is the sum of the cost in the retrieval of all data entries and one I/O cost for each data record. It is given as, $0.125B(D + BRC) + BR(D + C)$.

2) Insertion: It involves the cost of inserting a record i.e., 2D + C in the heap file, the cost of finding the page cost of adding a new entry and rewriting of the page, it is expressed as,

$2D + C + (H + 2D + C)$.

3) Deletion: It involves the cost of finding the data record and the data entry at H + 2D + 4RC and writing back the changed page to the index and file at 2D. The total cost is, (H + 2D + 4RC) + 2D.

4) Equality Selection Search: The total cost in the search accounts to,

i) The page containing the qualifying entries is identified at the cost H.

ii) Retrieval of the page, assuming that it is the only page present in the bucket occurs at D.

iii) The cost of finding an entry after scanning half the records on the page is 4RC.

iv) Fetching a record from the file is D. The total cost is,

H + D + 4RC + D ➔ (H + 2D + 4RC)

In case of many matched records the cost is,

H + D + 4RC + one I/O for each record that qualifies.

5) Range Selection Search: The cost of this is B(D + RC).

Comparing Advantages and Disadvantages of Different File Organizations:

File Organization	Advantages	Disadvantages
1) Heap file	- Good storage efficiency - Rapid scanning - Insertion is fast	- slow searches - slow deletion
2) Sorted file	- Good storage efficiency - Search is faster than heap file.	- Insertion is slow. - Slow deletion.
3) Clustered file	- Good storage efficiency - Searches fast - Efficient insertion and deletion	- Space overhead
4) Heap file with Un-clustered tree index.	- Fast insertion, deletion and searching	- Scanning and range searches are slow.
5) Heap file with Un-clustered Hash index.	- Fast insertion, deletion and searching	- Doesn't support range searches.

Dangling Pointer: Dangling pointer is a pointer that does not point to a valid object of the appropriate type. Dangling pointers arise when an object is deleted or de-allocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the de-allocated memory.

In abject-oriented database, dangling pointer occur if we move or delete a record to which another record contains as pointer that pointer no longer points to the desired record.

Detecting Dangling pointer in object-oriented Databases: Mapping objects to files is similar to mapping tupples to files in a relational system, object data can be stored using file structures. Objects are identified by an object identifier (OID), the storage system needs a mechanism to locate given its OID.

Logical identifiers do not directly specify an objects physical location, must maintain an index that maps an OID to the object's actual location.

Physical identifiers encode the location of the object so the object can be found directly. Physical OIDs have the following parts.

1) A volume or file identifier.
2) A page identifier within the volume or file.

3) An offset within the page.

Physical OIDs may have a unique identifier. This identifier is stored in the object also and is used to detect reference via dangling pointers.

Vol. page offset	Unique_id

Unique_id	Data .....

Indexes and Performance Tunning:

The performance of the system depends greatly on the indexes. This is done in terms of the expected work load.

Work Load Impact: Data entries that qualify particular selection criteria can be retrieved effectively by means of indexes. Two selection types are,

1) Equality 2) Range Selection.

1) Equality:

→ An equality query for a composite search key is defined as a search key in which each field is associated with a constant.

For **Example** data entries in a student file where rno = 15 and mrks = 90 can be retrieved by using equality query.

→ This is supported by Hash-file organization

2) Range Query: A range query for a composite search key is defined as a search key in which all the fields are not bounded to the constants.

Example: Data entries in a student file where rno = 15 with any mrks can be retrieved.

Thus, Tree-based indexing supports both the selection criteria as well as inserts, deletes and updates whereas only equality selection is supported by hash-based indexing apart form insertion, deletion and updation.

Advantages of using tree-structured indexes:

1) By using tree-structured indexes, insertion and deletion of data entries can be handled effectively.
2) It finds the correct leaf page faster than binary search in a sorted file.

Disadvantages:

The stored file pages are in accordance with the disk's order hence sequential retrieval of such pages is quicker which is not possible in tree-structured indexes.