UNIT-III

Contents:

Network Data Model:

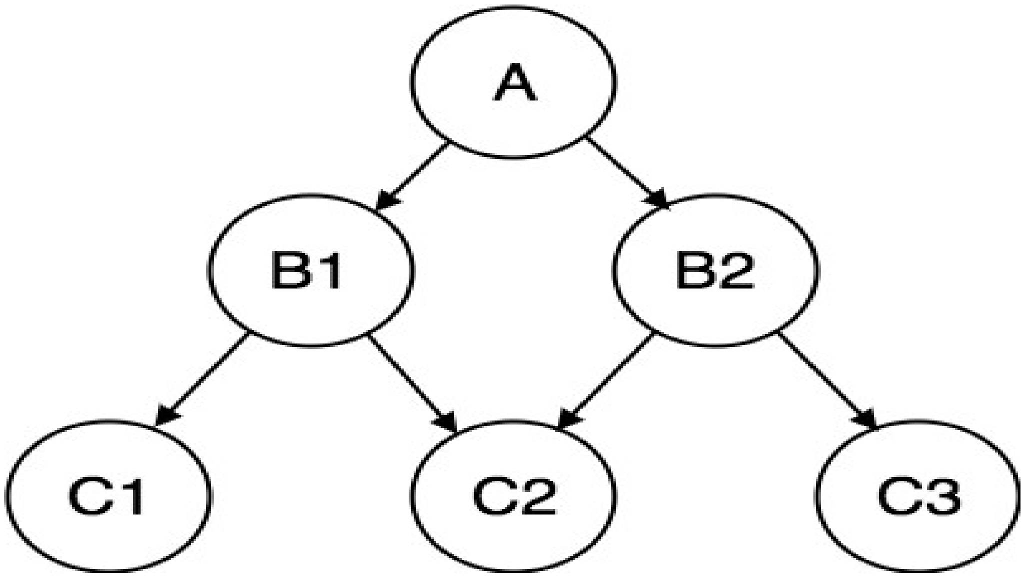
Introduction:

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

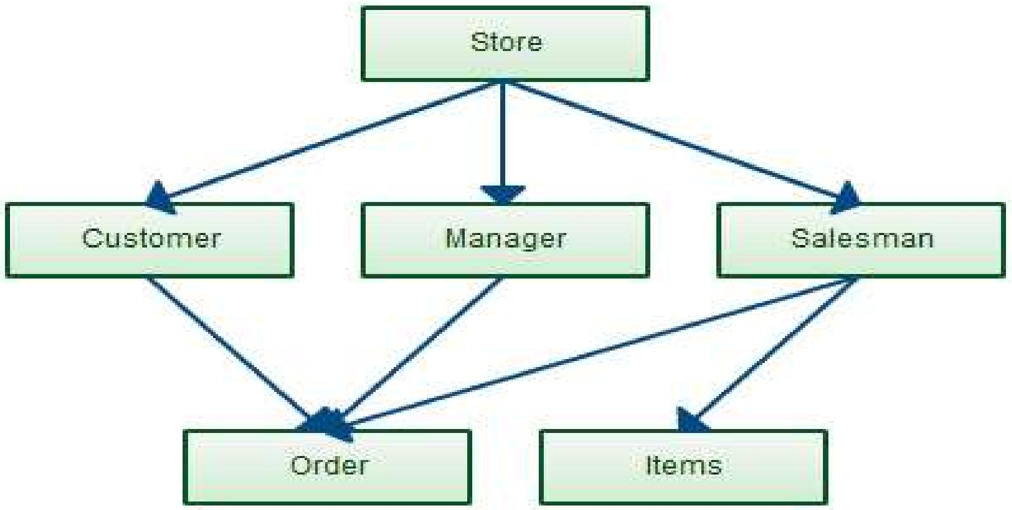
In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.

Example 1:



Example 2:



Characteristics of network model:

Better than hierarchical model

Supports many to many relationships i.e. many parents can have many child's and many children's can have many parents.

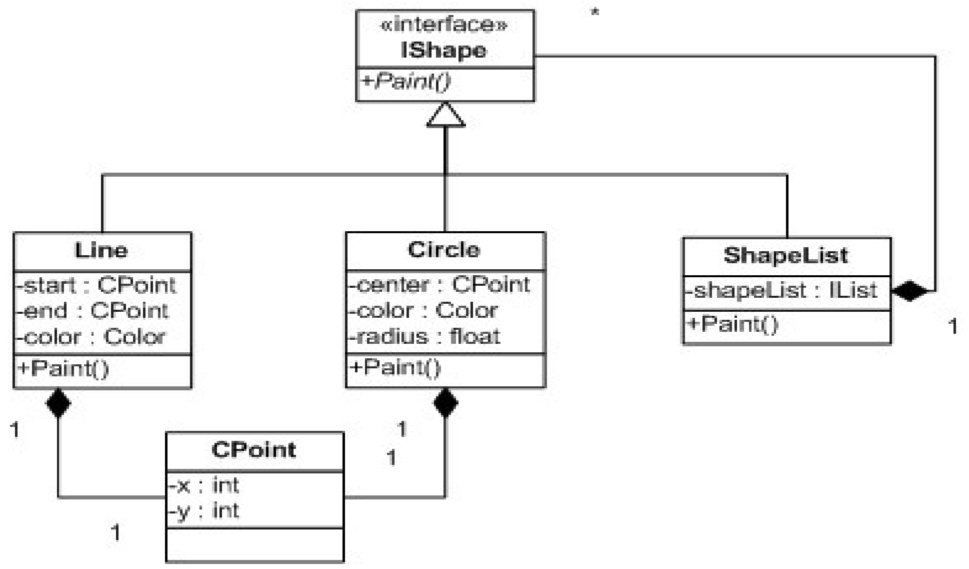
High performance

Complex structure

Object Oriented Data Model:

Object oriented data model is one of the developed data model and this can hold the audio, video and graphic files. These consist of data piece and the methods which are the DBMS instructions.

Example 1:



Memory Hierarchy:

A computer system has a well-defined hierarchy of memory. A CPU has direct access to it main memory as well as its inbuilt registers. The access time of the main memory is obviously less than the CPU speed. To minimize this speed mismatch, cache memory is introduced. Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.

The memory with the fastest access is the costliest one. Larger storage devices offer slow speed and they are less expensive, however they can store huge volumes of data as compared to CPU registers or cache memory.

Storage Strategies: Architecture:

Primary Storage − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory

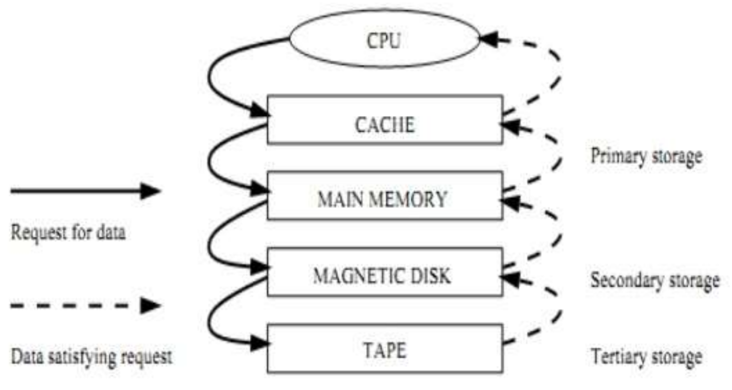
(RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost. Primary storage is generally volatile.

Cache: The collection of data which is stored in a hidden or inaccessible place. A cache is a smaller, faster memory, closer to a processor which stores copies of the data from frequently used main memory.

Main Memory: It refers to the physical memory that is internal to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives that is in main memory.

THE MEMORY HIERARCHY(revisited)

Memory in a computer system is arranged in a hierarchy, as shown in Figure



-61-

Figure The Memory Hierarchy

<u>**Secondary Storage**</u> − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes. Secondary storage is non-volatile.

Magnetic Disk: Hard disk drives are the most common secondary storage devices in present computer systems. These are called magnetic disks because they use the concept of magnetization to store information. Hard disks consist of metal disks coated with magnetisable material. These disks are placed vertically on a spindle. A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it. A magnetized spot can be recognized as 0 (zero) or 1 (one).

Hard disks are formatted in a well-defined order to store data efficiently. A hard disk plate has many concentric circles on it, called **tracks**. Every track is further divided into **sectors**. A sector on a hard disk typically stores 512 bytes of data.

Tertiary Storage − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

Tape: Tape is a narrow plastic strip covered with a magnetic substance. It is used to record sounds, pictures, and computer data.

Tape is faster than disk when performing streaming read/write operations, but content on magnetic tape can only be read or written in a sequential format

RAID: "**Redundant Arrays of Independent Disks**" is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both. The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

Why data redundancy?

Data redundancy, although taking up extra space, adds to disk reliability. This means, in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation. On the other hand, if the data is spread across just multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

Key evaluation points for a RAID System

- **Reliability:** How many disk faults can the system tolerate?
- **Availability:** What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
- **Performance:** How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
- **Capacity:** Given a set of N disks each with B blocks, how much useful capacity is available to the user?

RAID is very transparent to the underlying system. This means, to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes in the existing code.

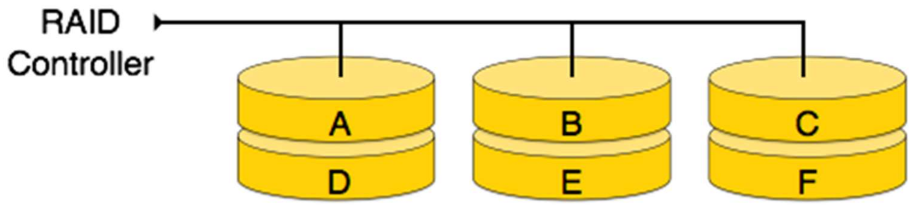
RAID Controller: A RAID controller is a hardware device or software program used to manage hard disk drives.

A controller offers a level of abstraction between an operating system and the physical drives. A RAID controller presents groups to applications and operating systems as logical units for which data protection schemes can be defined. Because the controller has the ability to access multiple copies of data on multiple physical devices, it has the ability to improve performance and protect data in the event of a system crash.

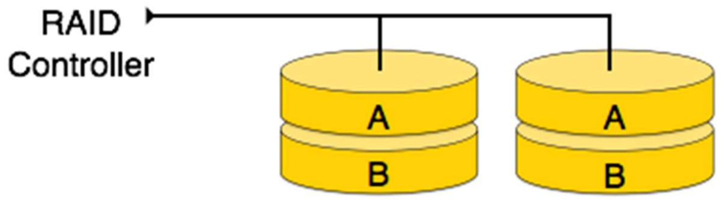
Different RAID Levels:

RAID 0 (Data Striping): In Data Striping, the data is segmented into equal-size partitions distributed over multiple disks. The size of the partition is called striping unit.

In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.

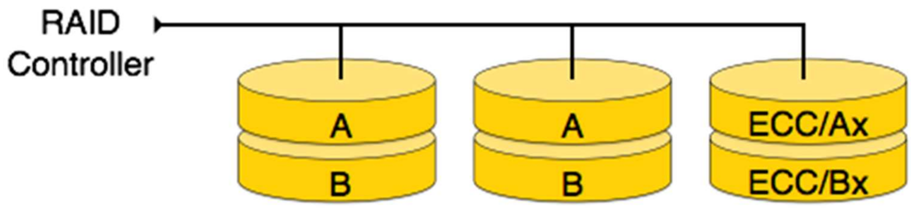


RAID 1(Mirroring): RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called **mirroring** and provides 100% redundancy in case of a failure.



RAID 0+1(Striping and Mirroring): Sometimes it is also called RAID level 10 combines striping and mirroring.

RAID 2(Error Correcting Codes): RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.

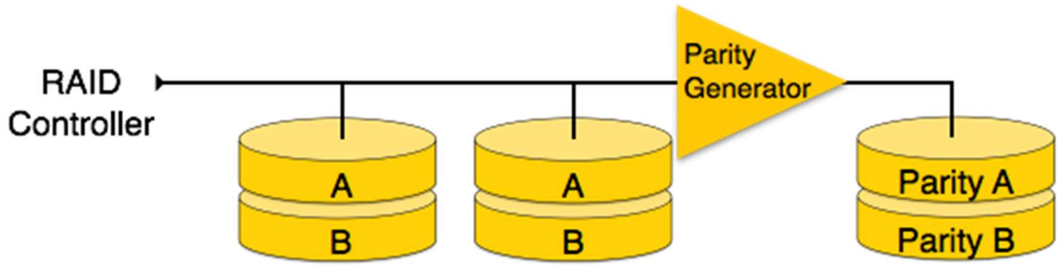


Parity: In computers, parity (from the Latin *paritas*, meaning equal or equivalent) is a technique that checks whether data has been lost or written over when it is moved from one place in storage to another or when it is transmitted between computers.

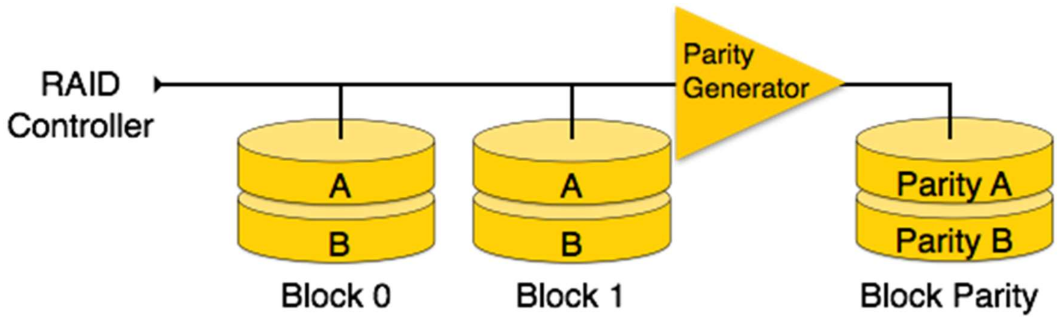
Because data transmission is not an entirely error-free process, data is not always received in the same way as it was transmitted. A parity bit adds checksums into data that enable the target device to determine whether the data was received correctly.

- **In even parity**, the number of bits with a value of one are counted. If that number is odd, the parity bit value is set to one to make the total number of ones in the set (including the parity bit) an even number. If the number of bits with a value of one is even, the parity bit value is set to zero, so that the total number of ones in the set (including the parity bit) remains an even number.
- **In odd parity**, if the number of bits with a value of one is an even number, the parity bit value is set to one to make the total number of ones in the set (including the parity bit) an odd number. If the number of bits with a value of one is odd, the parity bit value is set to zero, so that the total number of ones in the set (including the parity bit) remains an odd number.

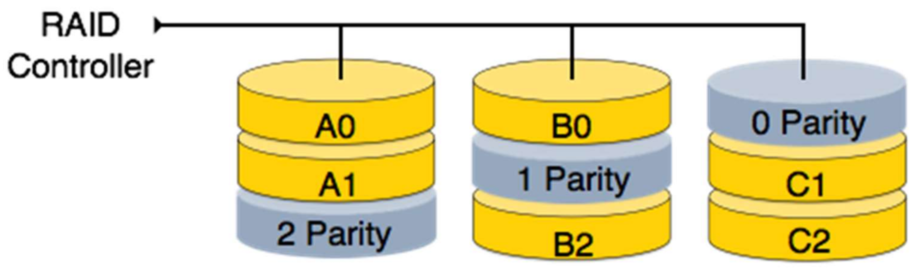
RAID 3 (Byte Level Parity): RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



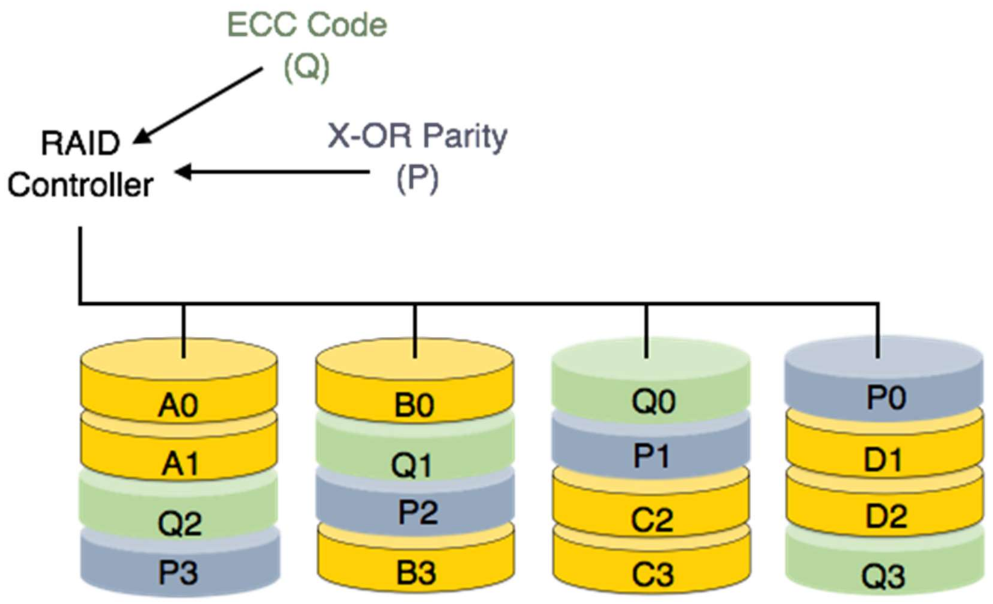
RAID 4 (Block Level Parity): In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



RAID 5 (Rotating Parity): RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID 6(Tolerates failure of two disk drives): RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.



File Organization:

Introduction: A database consists of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records. A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.

What is File Organization?

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in certain order is called file Organization. **File Structure** refers to the format of the label and data blocks and of any logical control record.

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

File Operations: Operations on database files can be broadly divided in to categories

- **Update Operations**- it change the data values by insertion, deletion or update
- **Retrieval Operations**- do not alter the data, but retrieve the data.
- Other than this operations, we have different operations.

Open- A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.

Locate − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.

Read − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.

Write − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.

Close − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system

- removes all the locks (if in shared mode),
- saves the data (if altered) to the secondary storage media, and
- releases all the buffers and file handlers associated with the file.

Delete – deletes the current record
Find Next – searches for the next record
Modify – modifies some field values for the current record
Insert – inserts new record
Find all – it locates all the records in the file
Find ordered – retrieves all the records in the file in some specified order
Scan – if the file has just been opened or rest, scan returns the first record; otherwise it returns next record

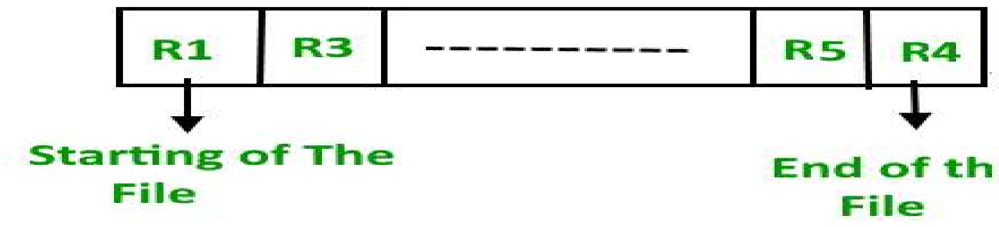
Types of File Organization: In a database we have lots of data. Each data is grouped into related groups called tables. Each table will have lots of related records. Any user will see these records in the form of tables in the screen. But these records are stored as files in the memory. Usually one file will contain all the records of a table.

There are various methods of file organizations. These methods may be efficient for certain types of access/selection meanwhile it will turn inefficient for other selections. Hence it is up to the programmer to decide the best suited file organization method depending on his requirement.

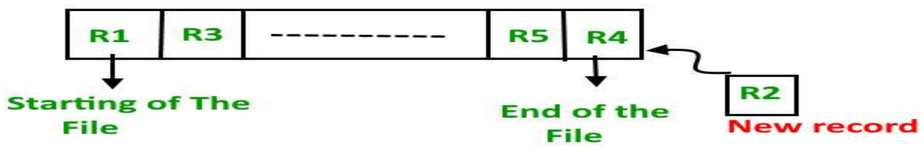
1. Sequential File Organization
2. Heap File Organization
3. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization
6. Cluster File Organization

1. Sequential File Organization: The easiest method for file Organization is Sequential method. In this method the files are stored one after another in a sequential manner. There are two ways to implement this method:

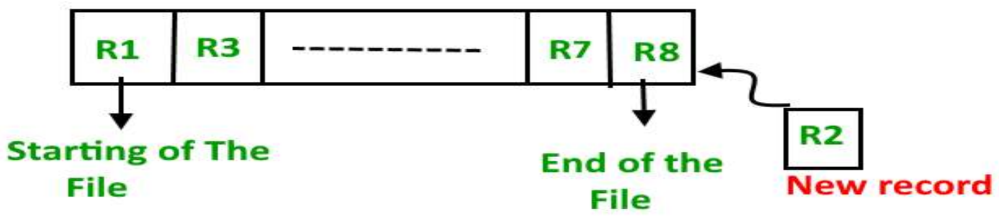
1. **Pile File Method** – This method is quite simple, in which we store the records in a sequence i.e. one after other in the order in which they are inserted into the tables.



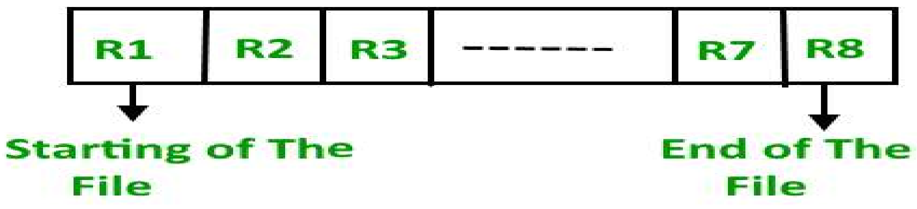
Insertion of new record – Let the R1, R3 and so on up to R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



2. **Sorted File Method** –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.



Insertion of new record – Let us assume that there is a pre-existing sorted sequence of four records R1, R3, and so on up to R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence.



Pros and Cons of Sequential File Organization –

Pros –

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e. cheaper storage mechanism.

Cons –

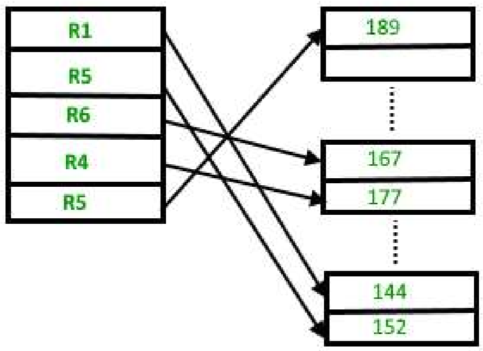
- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

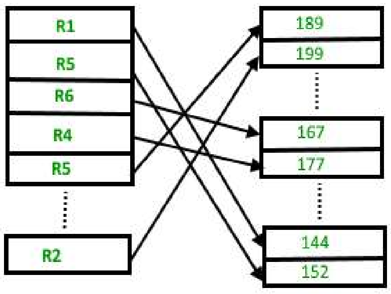
2. Heap File Organization: An unordered file, sometimes called a heap file, is the simplest type of file organization. Records are placed in file in the same order as they are inserted.

A new record is inserted in the last page of the file; if there is insufficient space in the last page, a new page is added to the file. This makes insertion very efficient. However, as a heap file has no particular ordering with respect to field values, a linear search must be performed to access a record.

A linear search involves reading pages from the file until the required is found. This makes retrievals from heap files that have more than a few pages relatively slow, unless the retrieval involves a large proportion of the records in the file. It is the responsibility of DBMS to store and manage the new records.

Insertion of new record – Suppose we have five records in the heap R1, R5, R6, R4 and R5 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e. data block 3 is full it will be inserted in any of the database selected by the DBMS, let's say data block 1.





If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Pros and Cons of Heap File Organization –

Pros –

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Cons –

- Problem of unused memory blocks.
- Inefficient for larger databases.

3. Hashing file organization: In database management system, when we want to retrieve a particular data, it becomes very inefficient to search all the index values and reach the desired data. In this situation, hashing technique comes into picture.

Hashing is an efficient technique to directly search the location of desired data on the disk without using index structure. Data is stored at the data blocks whose address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket.

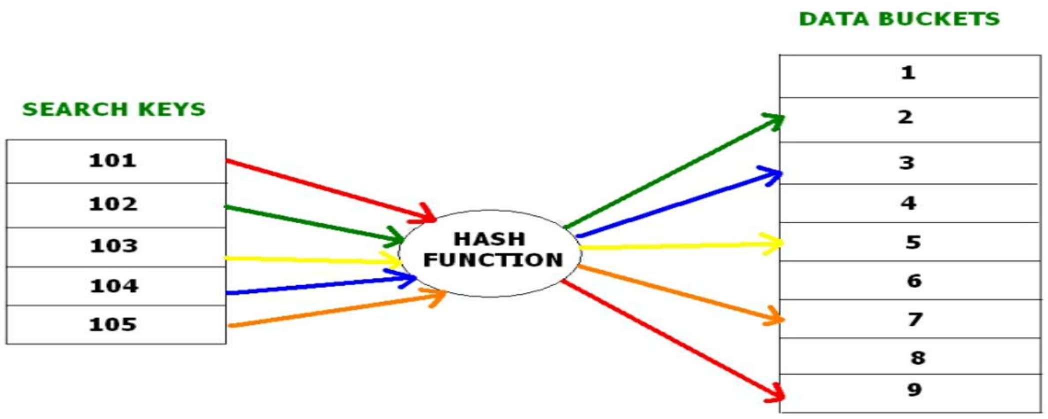
• **Data bucket –** Data buckets are the memory locations where the records are stored. These buckets are also considered as *Unit of Storage*.

• **Hash Function –** Hash function is a mapping function that maps all the set of search keys to actual record address. Generally, hash function uses primary key to generate the hash index

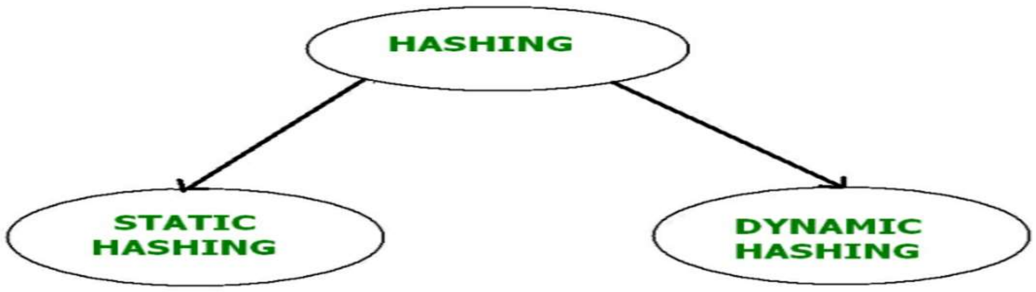
– address of the data block. Hash function can be simple mathematical function to any complex mathematical function.

• **Hash Index-**The prefix of an entire hash value is taken as a hash index. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed? then the depth value is increased linearly and twice the buckets are allocated.

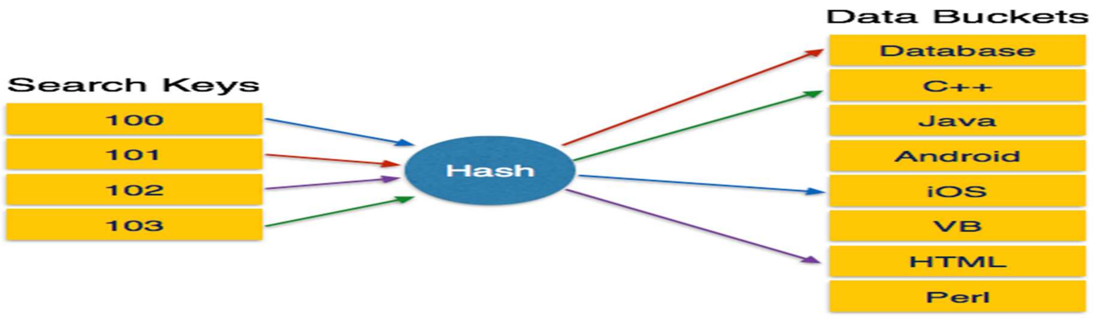
Below given diagram clearly depicts how hash function work:



Hashing is further divided into two sub categories:



<u>Static Hashing:</u> In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.



Operations –

• **Insertion –** When a new record is inserted into the table, the hash function h generates a bucket address for the new record based on its hash key K.

Bucket address = h(K)

• **Searching –** When a record needs to be searched, the same hash function is used to retrieve the bucket address for the record.

• **Deletion –** If we want to delete a record, Using the hash function we will first fetch the record which is supposed to be deleted. Then we will remove the records for that address in memory.

• **Updating –** The data record that needs to be updated is first searched using hash function, and then the data record is updated.

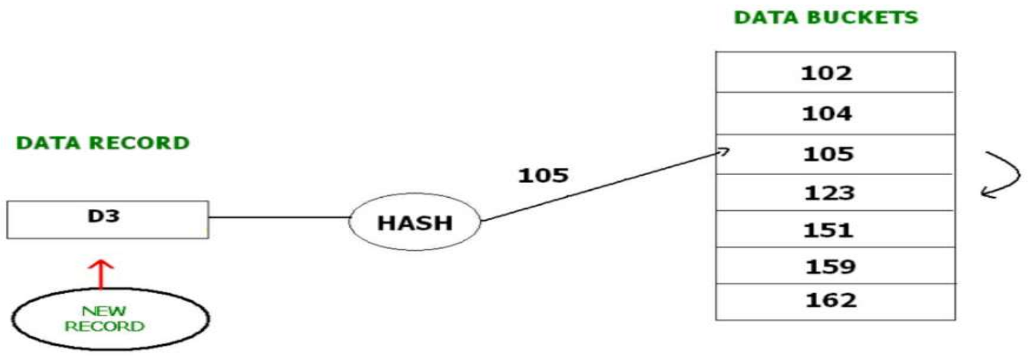
Now, if we want to insert some new records into the file but the data bucket address generated by the hash function is not empty or the data already exists in that address. This becomes a critical situation to handle. This situation in the static hashing is called **bucket overflow**.

How will we insert data in this case?

There are several methods provided to overcome this situation. Some commonly used methods are discussed below:

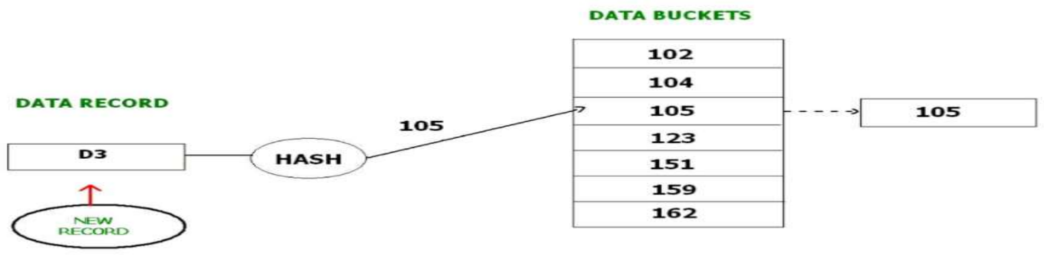
1. Open Hashing – In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one. This method is also called linear probing.

For example, D3 is a new record which needs to be inserted, the hash function generates address as 105. But it is already full. So the system searches next available data bucket, 123 and assigns D3 to it.



2. **Closed hashing –** In Closed hashing method, a new data bucket is allocated with same address and is linked it after the full data bucket. This method is also known as overflow chaining.

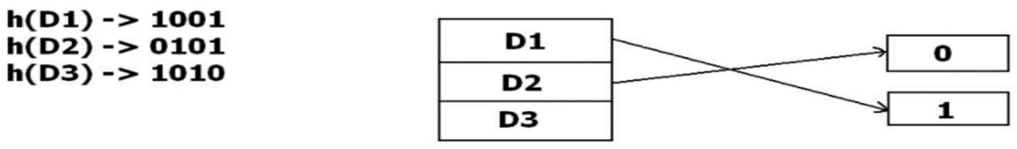
For example, we have to insert a new record D3 into the tables. The static hash function generates the data bucket address as 105. But this bucket is full to store the new data. In this case a new data bucket is added at the end of 105 data bucket and is linked to it. Then new record D3 is inserted into the new bucket.



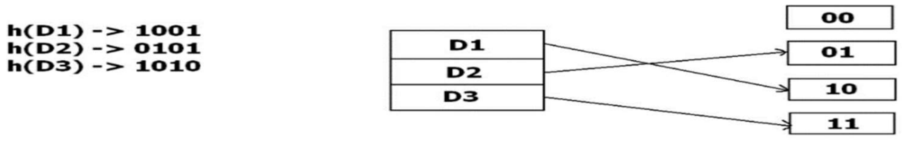
- **Quadratic probing:** Quadratic probing is very much similar to open hashing or linear probing. Here, the only difference between old and new bucket is linear. Quadratic function is used to determine the new bucket address.
- **Double Hashing:** Double Hashing is another method similar to linear probing. Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function. That's why the name is double hashing.

Dynamic Hashing: The drawback of static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. In Dynamic hashing, data buckets grow or shrinks (added or removed dynamically) as the records increases or decreases. Dynamic hashing is also known as extended hashing.

In dynamic hashing, the hash function is made to produce a large number of values. For Example, there are three data records D1, D2 and D3. The hash function generates three addresses 1001, 0101 and 1010 respectively. This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at address 0 and 1.



But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2-bit address. Then it tries to accommodate D3.



Pros and Cons of Heap File Organization –

Pros –

- Hash is a good storage structure in the following situations:

  When tuples are retrieve based on an exact match on the hash field value, particularly if the access order is random. For example, if the STUDENT relation is hashed on Name then retrieval of the tuple with Name equal to "Rahat Bhatia" is efficient.

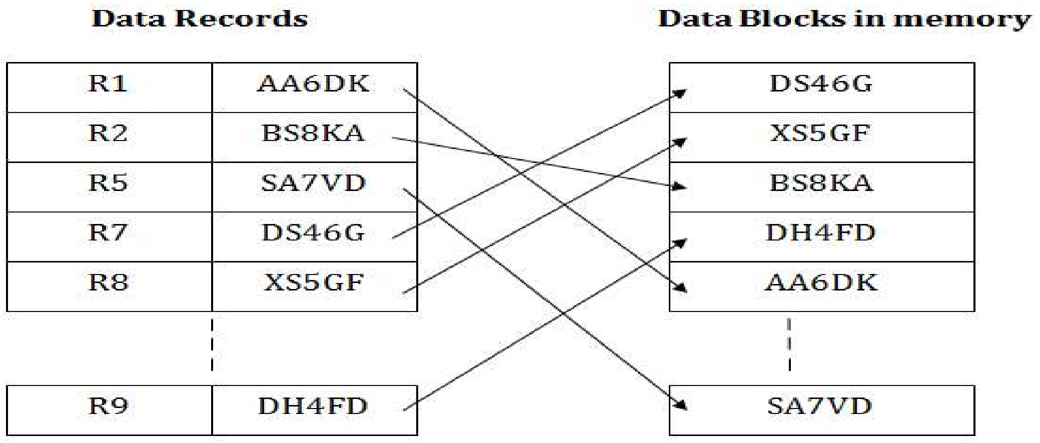
Cons –

- Hash is not a good storage structure in the following situations:

  When tuples are retrieved based on a range of values for the hash field. For example, retrieve all students whose name begins with the "R".

  When tuples are retrieved based on a range of values for the hash field. For example, if STUDENT relation has hash filed Roll Number and the query is to retrieve all students with roll numbers in the range of 3000-5000.

<u>4. Indexed Sequential Access Method:</u> ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros and Cons of ISAM–

Pros –

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons –

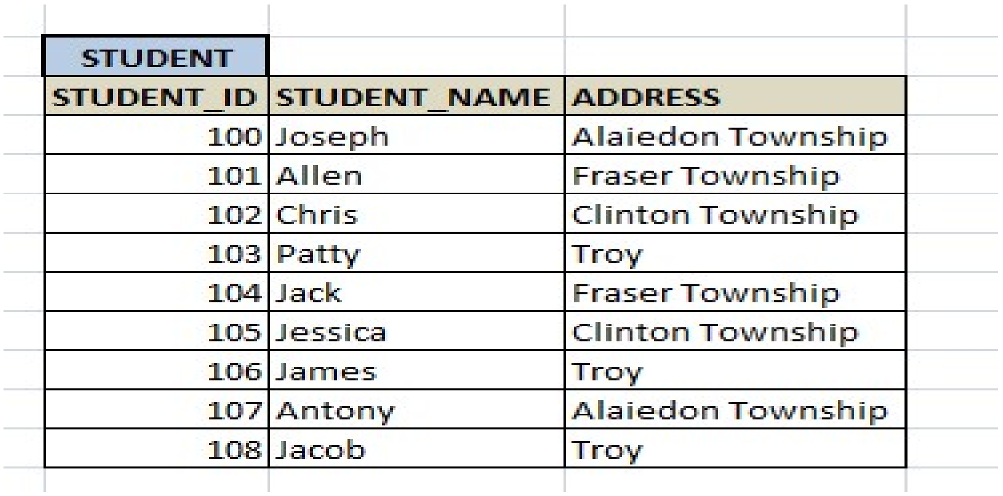
- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.

5. B+ Tree File Organization: B+ Tree, as the name suggests, it uses a tree like structure to store records in File.

It uses the concept of Key indexing where the primary key is used to sort the records. For each primary key, an index value is generated and mapped with the record. An index of a record is the address of record in the file.

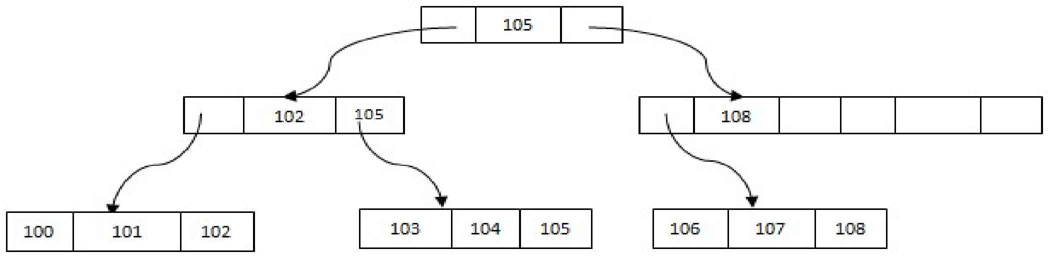
B+ Tree is very much similar to binary search tree, with the only difference that instead of just two children, it can have more than two. All the information is stored in leaf node and the intermediate nodes acts as pointer to the leaf nodes. The information in leaf nodes always remain a sorted sequential linked list.

Consider a student table below. The key value here is STUDENT_ID. And each record contains the details of each student along with its key value and the index/pointer to the next value. In a B+ tree it can be represented as below.



STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS
100	Joseph	Alaiedon Township
101	Allen	Fraser Township
102	Chris	Clinton Township
103	Patty	Troy
104	Jack	Fraser Township
105	Jessica	Clinton Township
106	James	Troy
107	Antony	Alaiedon Township
108	Jacob	Troy



From the above B+ tree structure, it is evident that

- There is one main node called root of the tree – 105 is the root here.
- There is an intermediary layer with nodes. They do not have actual records stored. They are all pointers to the leaf node. Only the leaf node contains the data in sorted order.
- The nodes to the left of the root nodes have prior values of root and nodes to the right have next values of the root. i.e.; 102 and 108 respectively.
- There is one final node, called leaf node, which has only values. i.e.; 100, 101, 103, 104, 106 and 107
- All the leaf nodes are balanced – all the leaf nodes at same distance from the root node. Hence searching any record is easier.
- Searching any record is linear in this case. Any record can be traversed through single path and accessed easily.
- Since the intermediary nodes have only pointers to the leaf node, the tree structure is of shorter height. Shorter the height, faster is the traversal and hence the retrieval of records.

Pros and Cons of B+ Tree –

Pros –

- Since all records are stored only in the leaf node and are sorted sequential linked list, searching is becomes very easy.
- Using B+, we can retrieve range retrieval or partial retrieval. Traversing through the tree structure makes this easier and quicker.
- There is no restriction on B+ tree size, like we have in ISAM.
- Since it is a balance tree structure, any insert/ delete/ update does not affect the performance.

Cons –

- This method is less efficient for static tables.

6. Cluster File Organization: When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.

This method reduces the cost of searching for various records in different files.

The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization: Cluster file organization is of two types

1. Indexed Clusters: In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. Hash Clusters: It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Pros and Cons of Cluster File Organization –

Pros –

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

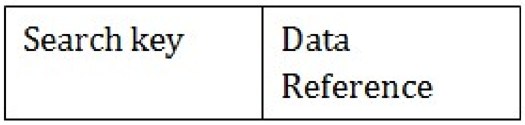
Cons –

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining, then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.

<u>Indexing in DBMS:</u> Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index Structure: Indexes can be created using some database columns.

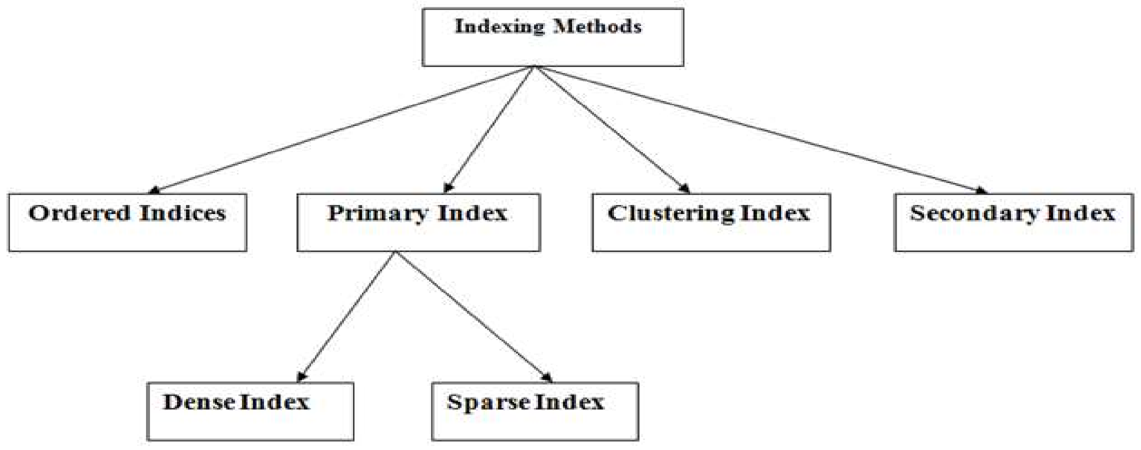


Search key	Data Reference

Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods:



Ordered Indices: The indices are usually sorted so that the searching is faster. The indices which are sorted are known as ordered indices.

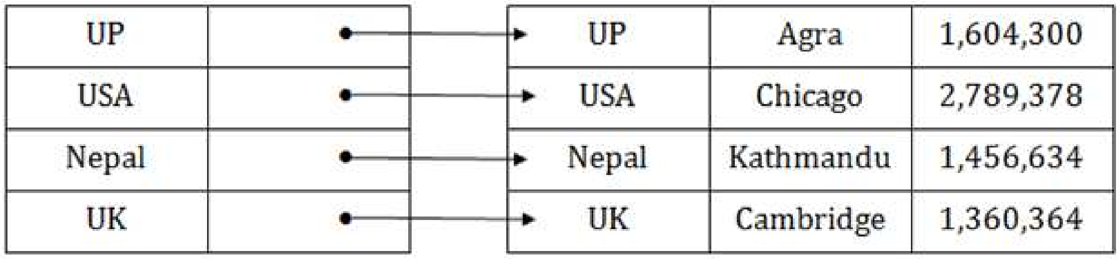
- If the search key of any index specifies same order as the sequential order of the file, it is known as primary index or clustering index.
- **Note:** The search key of a primary index is usually the primary key, but it is not necessarily so.
- If the search key of any index specifies an order different from the sequential order of the file, it is called the secondary index or non-clustering index.

Primary Index:

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

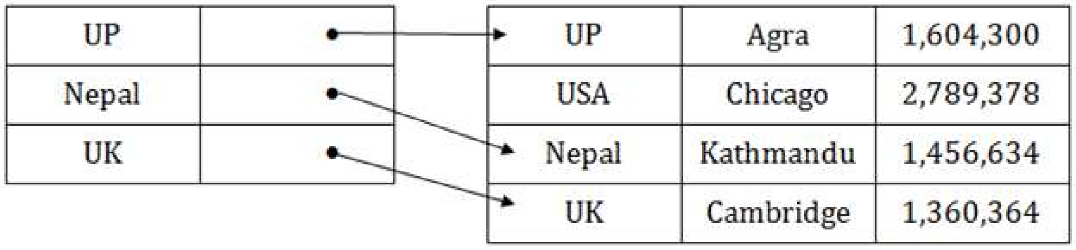
Dense Index:

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



Sparse Index:

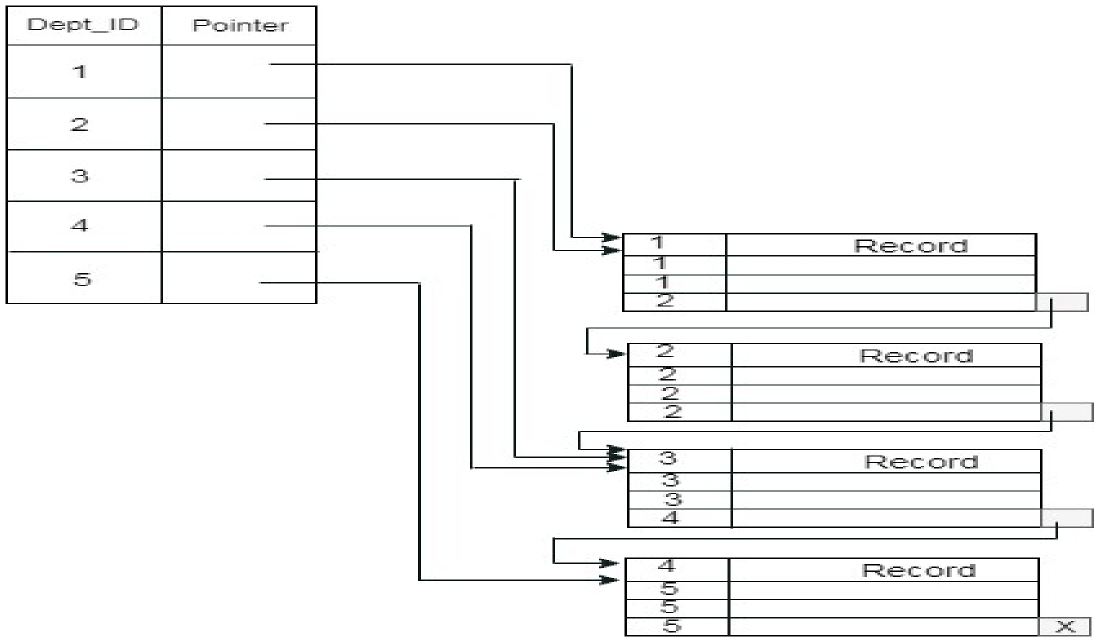
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



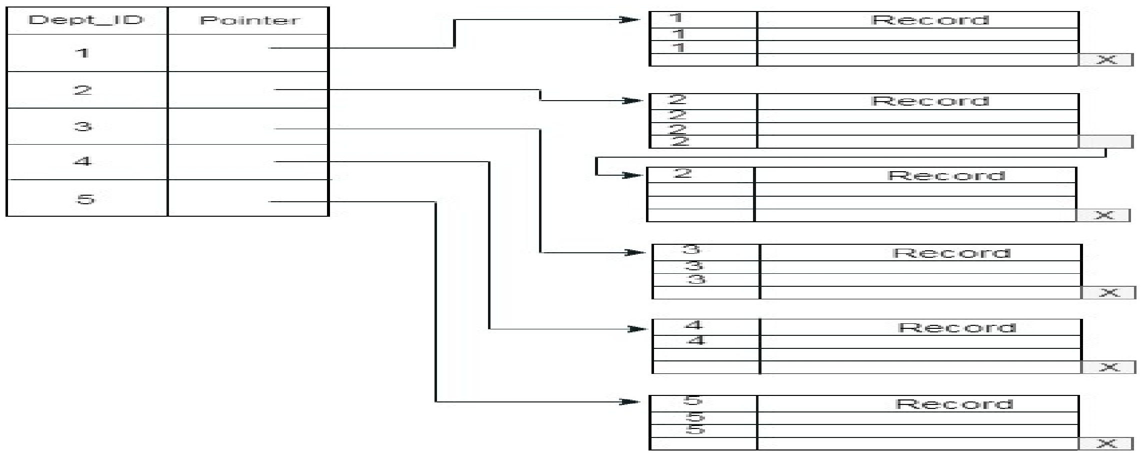
Clustering Index:

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.

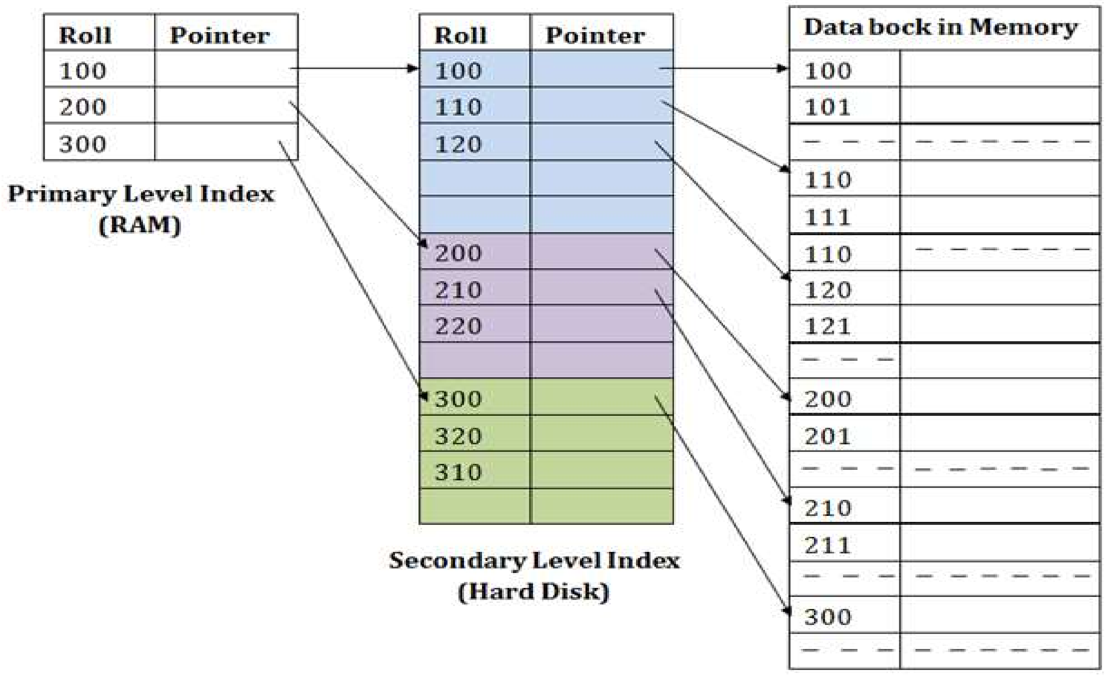


The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index: In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows, then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



B+ Tree indexing: An Index can be simply defined as an optional structure associated with a table cluster that enables the speed access of data. One can reduce the disk I/O by this. Creating an index, a small set of randomly distributed rows from the table can be retrieved.

There are many reasons for using B-trees such as

- Provides the best way to retrieve the wide range of queries.
- Speeds up the data access.
- Excellent for highly selective indexes and primary keys.
- Retrieves the data sorted by indexed columns, used as concatenated indexes.
- Provides universal applicability.
- They are storage-friendly, work on any layer of the storage hierarchy.

Data Dictionary: It is a dictionary about the data that we store in the database. It contains all the information about the data objects. It is like storing all up-to-date information about the objects like tables, columns, index, constraints, functions etc.

In the case of a table, data dictionary provides information about

- Its name
- Security information like who is the owner of the table, when was it created, and when it was last accessed.
- Physical information like where is the data stored for this table
- Structural information like its attribute names and its datatypes, constraints and indexes.

Types of Data Dictionary

1. Active Data Dictionary: Any changes to the database object structure via DDLs will have to be reflected in the data dictionary. But updating the data dictionary tables for the changes are responsibility of database in which the data dictionary exists. If the data dictionary is created in the same database, then the DBMS software will automatically update the data dictionary. Hence there will not be any mismatch between the actual structure and the data dictionary details. Such data dictionary is called active data dictionary.

2. Passive Data Dictionary: In some of the databases, data dictionary is created separately from the current database as entirely new database to store only data dictionary information's. Sometimes it is stored as xml, excels or in any other file format. In such case, an effort is required to keep data dictionary in sync with the database objects. This kind of data dictionary is called passive data dictionary. In this case, there is a chance of mismatch with the database objects and the data dictionary. This kind of DD has to be handled with utmost care.