

File Structures: Physical Storage Media File Organization, Organization of records into Blocks, Sequential Files, Indexing and Hashing, Primary indices, Secondary indices, B+ Tree index Files, B Tree index Files, Indexing and Hashing Techniques and their Comparisons.

## What is File?

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

**There are two important features of file:**

1. File Activity
2. File Volatility

**File activity** specifies percent of actual records which proceed in a single run.

**File volatility** addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

## File Organization

File organization ensures that records are available for processing. It is used to determine an efficient file organization for each base relation.

For example, if we want to retrieve employee records in alphabetical order of name. Sorting the file by employee name is a good file organization. However, if we want to retrieve all employees whose marks are in a certain range, a file is ordered by employee name would not be a good file organization.

## Types of File Organization

**There are three types of organizing the file:**

1. Sequential access file organization
2. Direct access file organization
3. Indexed sequential access file organization

## **1. Sequential access file organization**

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.

### **Advantages of sequential file**

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

### **Disadvantages of sequential file**

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

## **2. Direct access file organization**

- Direct access file is also known as random access or relative file organization.
- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk. The records are randomly placed throughout the file.

- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

#### **Advantages of direct access file organization**

- Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.
- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately.
- It updates several files quickly.
- It has better control over record allocation.

#### **Disadvantages of direct access file organization**

- Direct access file does not provide back up facility.
- It is expensive.
- It has less storage space as compared to sequential file.

### **3. Indexed sequential access file organization**

- Indexed sequential access file combines both sequential file and direct access file organization.
- In indexed sequential access file, records are stored randomly on a direct access device such as magnetic disk by a primary key.
- This file have multiple keys. These keys can be alphanumeric in which the records are ordered is called primary key.
- The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

### **Advantages of Indexed sequential access file organization**

- In indexed sequential access file, sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- The records can be inserted in the middle of the file.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

### **Disadvantages of Indexed sequential access file organization**

- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.
- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

## **File Organization**

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

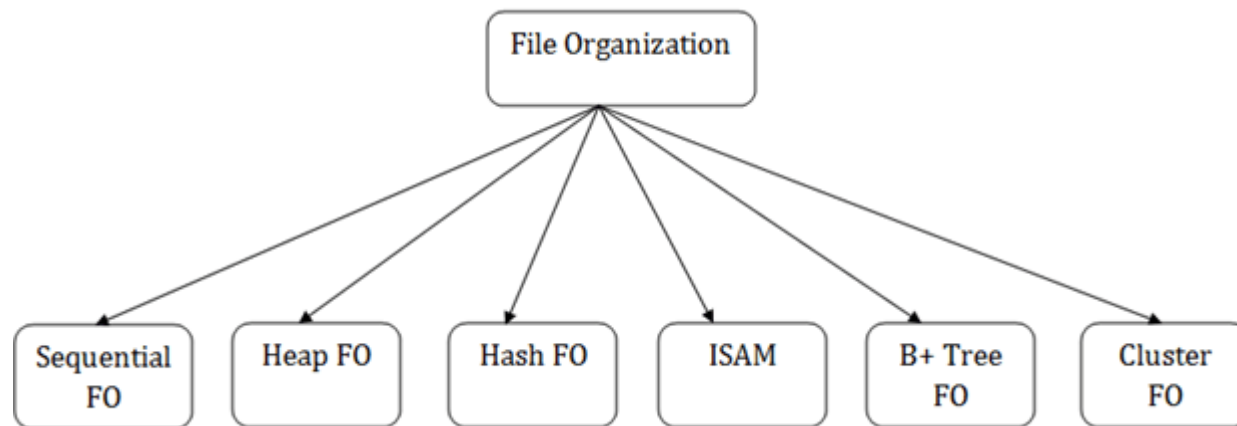
## Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

## Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



- [Sequential file organization](#)
- [Heap file organization](#)

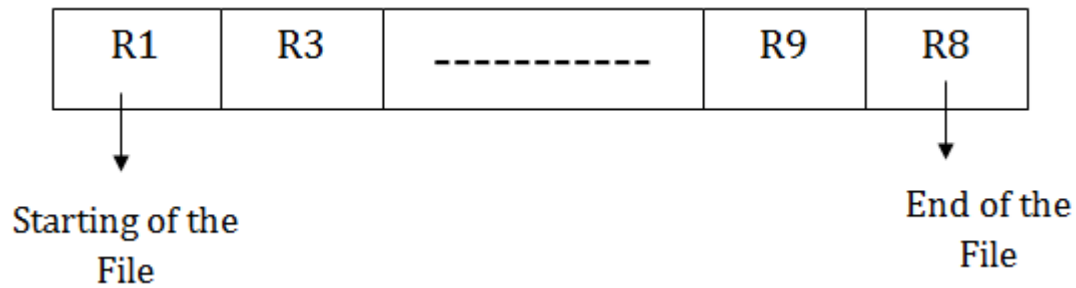
- [Hash file organization](#)
- [B+ file organization](#)
- [Indexed sequential access method \(ISAM\)](#)
- [Cluster file organization](#)

## Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

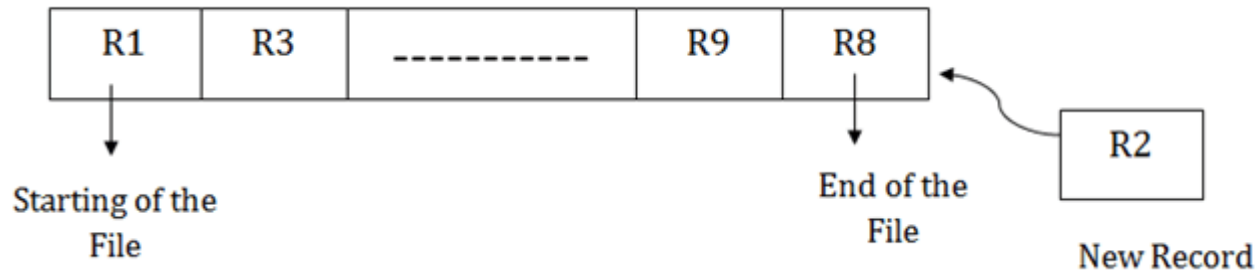
### 1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



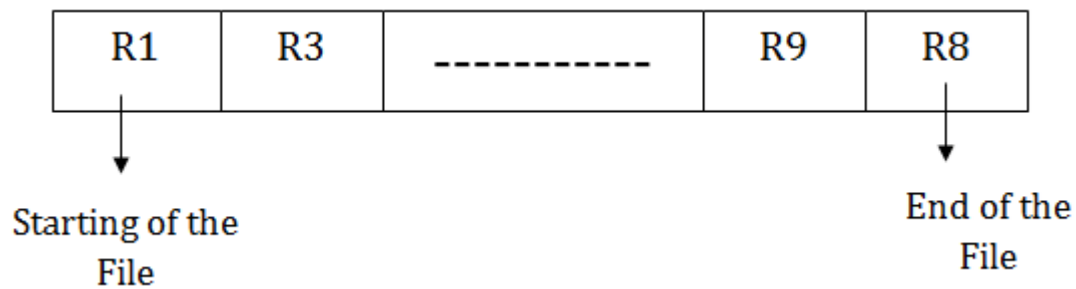
Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



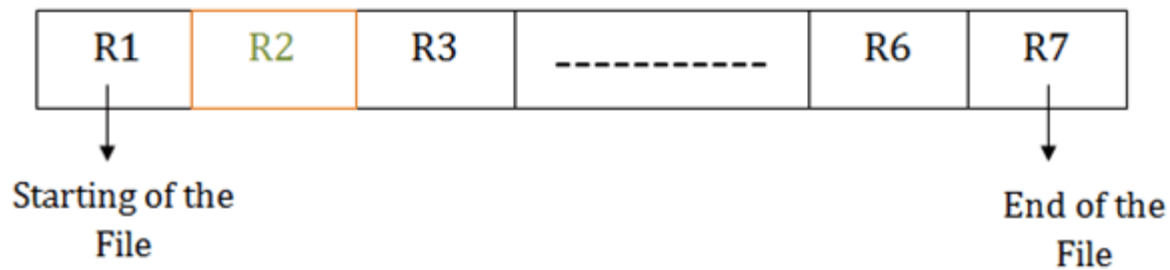
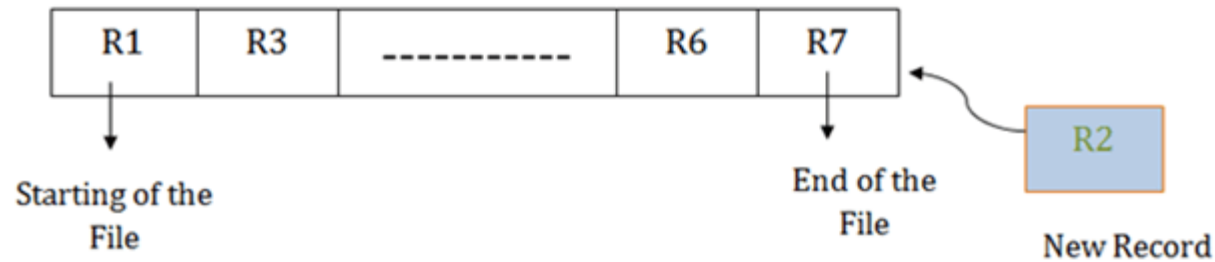
## 2. Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. 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 of sequential file organization

- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

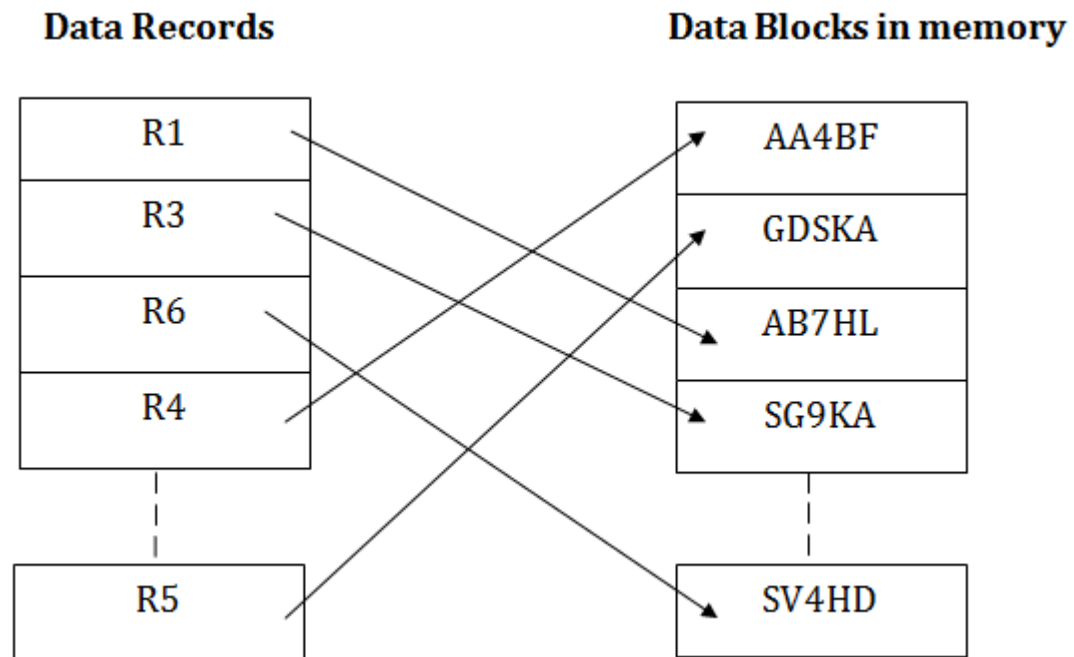


## Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

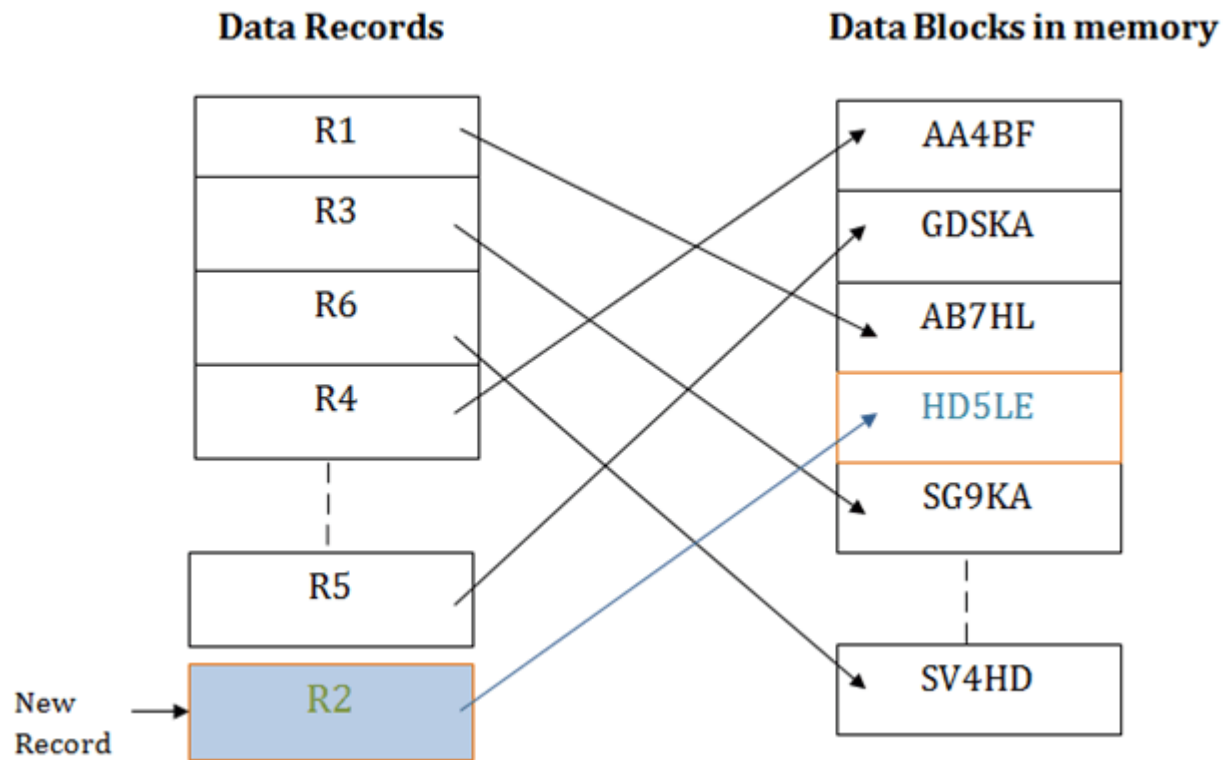
## Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



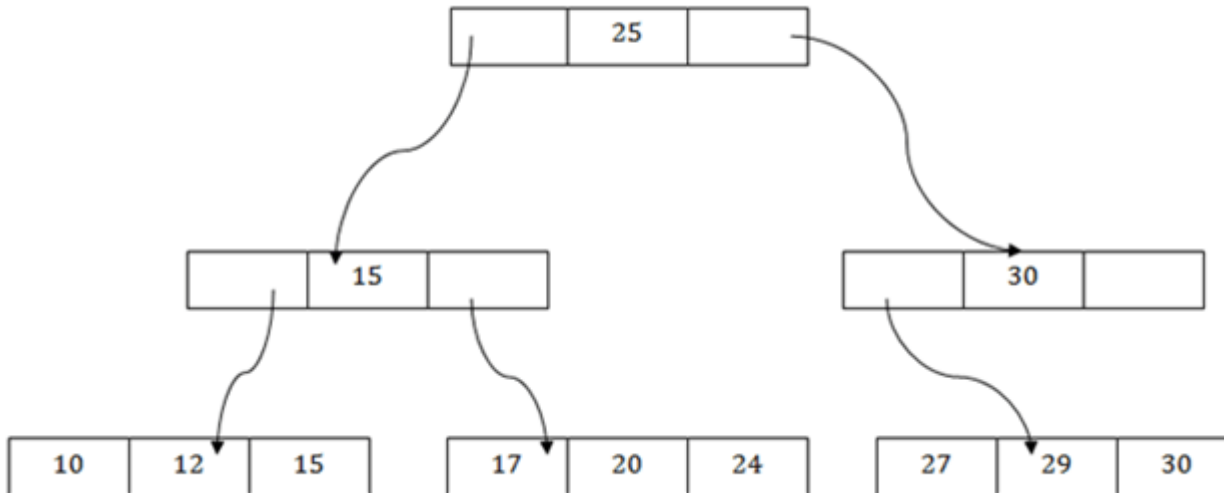
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



## B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.

- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

## Pros of B+ tree file organization

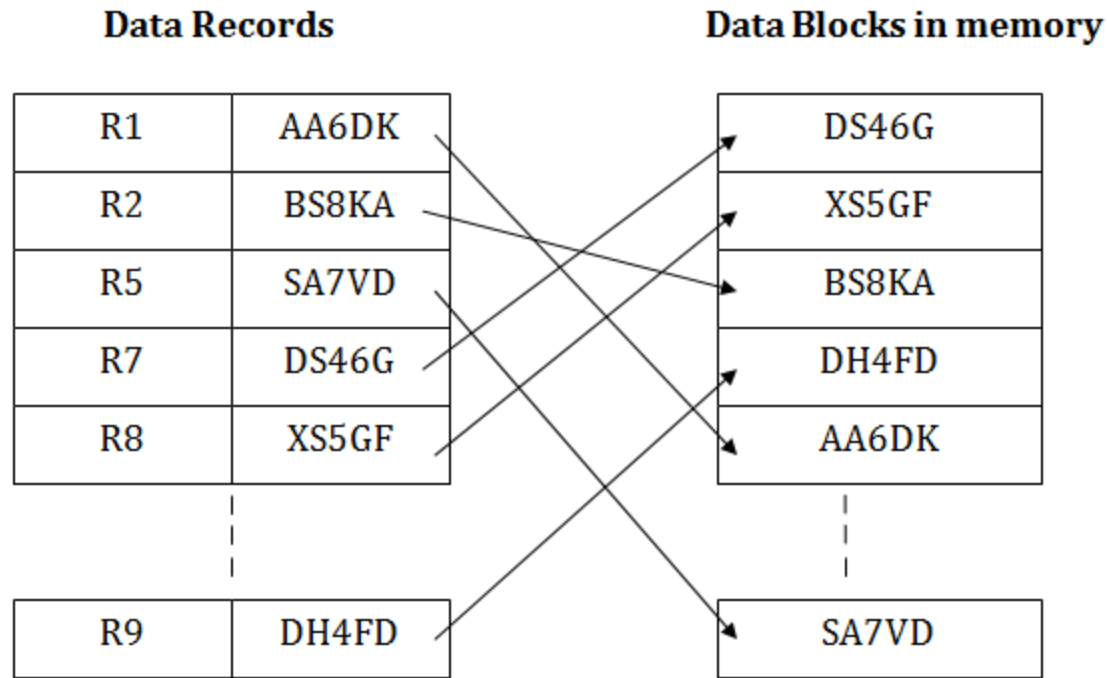
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

## Cons of B+ tree file organization

- This method is inefficient for the static method.

## Indexed sequential access method (ISAM)

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 of ISAM:

- 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 of ISAM

- 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.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

## 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 of Cluster file organization

- 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 of Cluster file organization

- 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.

## B+ Tree

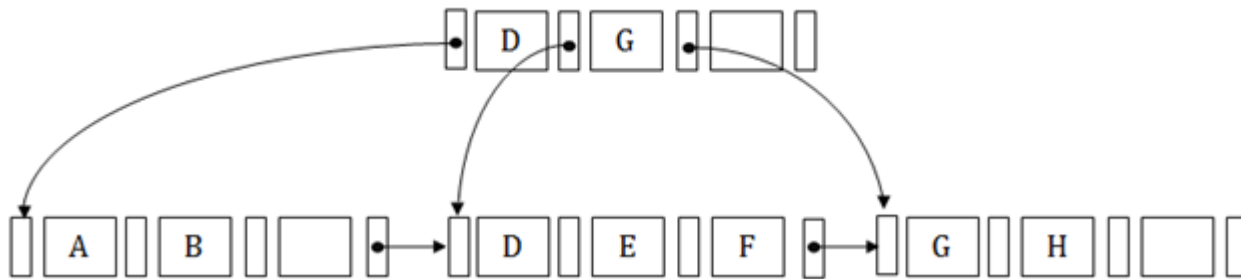
- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.



- o In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

## Structure of B+ Tree

- o In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- o It contains an internal node and leaf node.



## Internal node

- o An internal node of the B+ tree can contain at least  $n/2$  record pointers except the root node.
- o At most, an internal node of the tree contains n pointers.

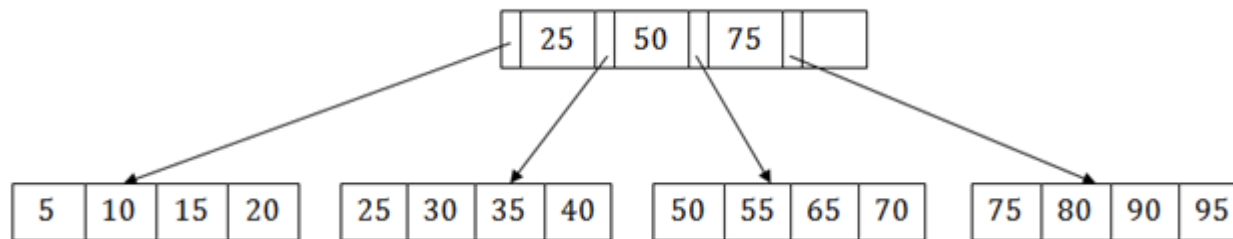
## Leaf node

- o The leaf node of the B+ tree can contain at least  $n/2$  record pointers and  $n/2$  key values.
- o At most, a leaf node contains n record pointer and n key values.
- o Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

## Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

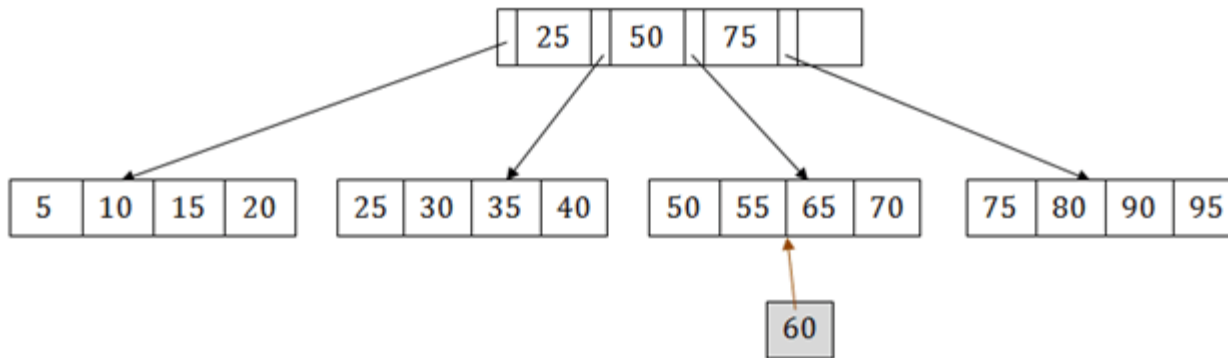
So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.



## B+ Tree Insertion

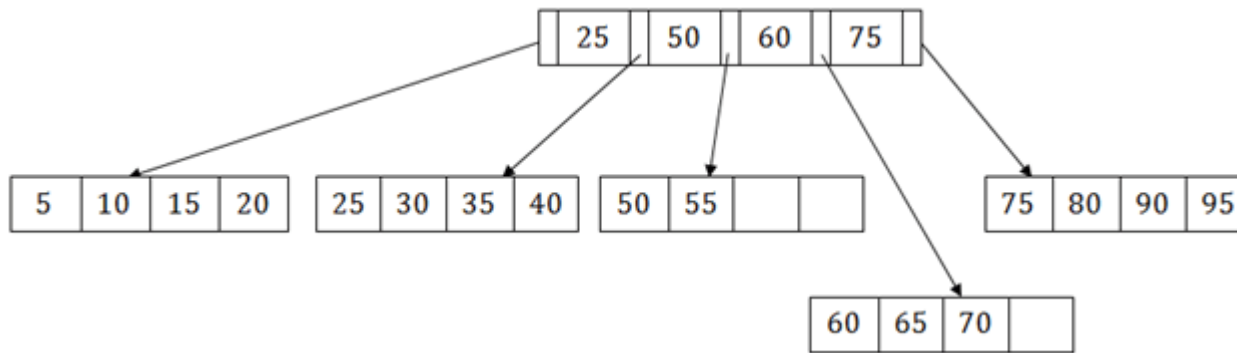
Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.

In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.



The 3<sup>rd</sup> leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.

If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.

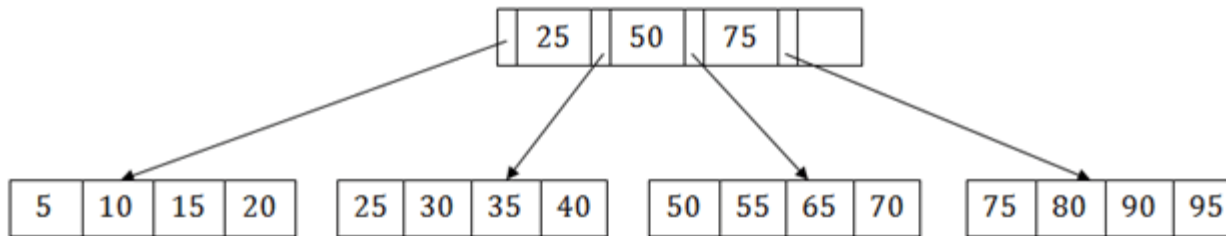


This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

## B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:

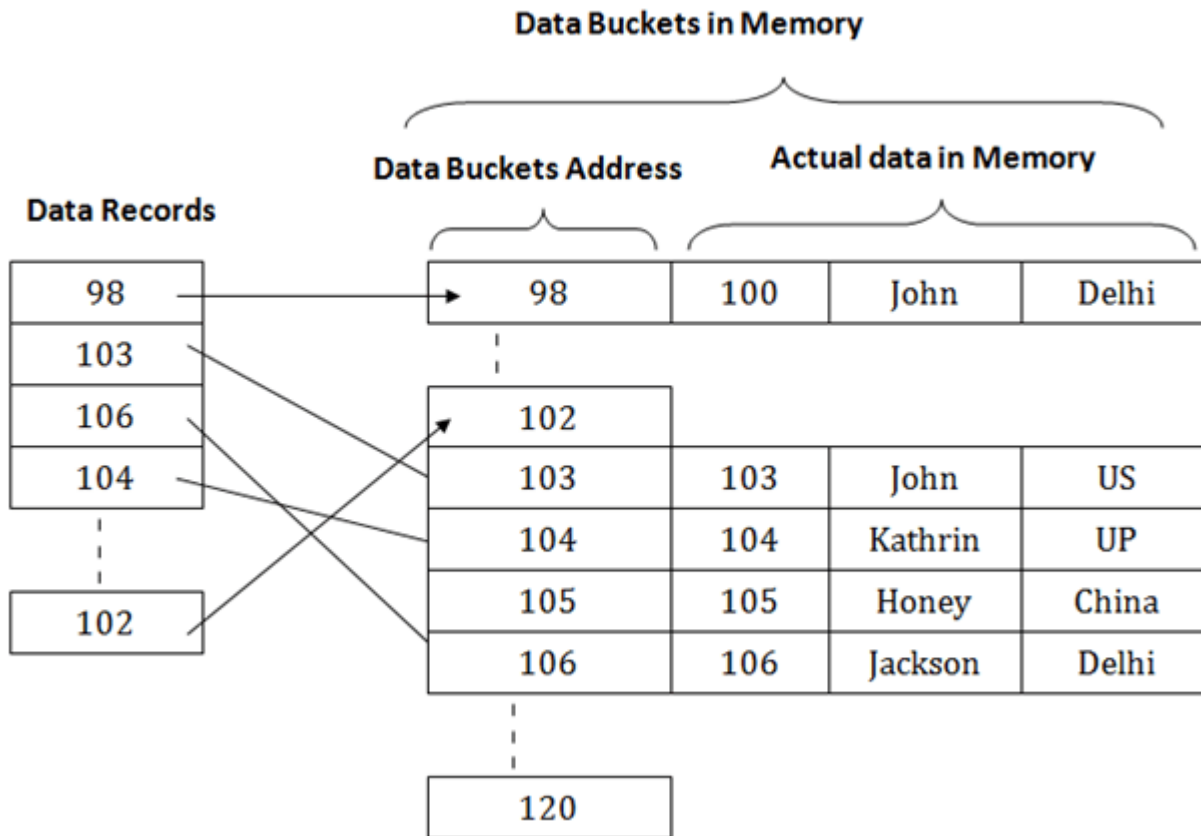


## Hashing

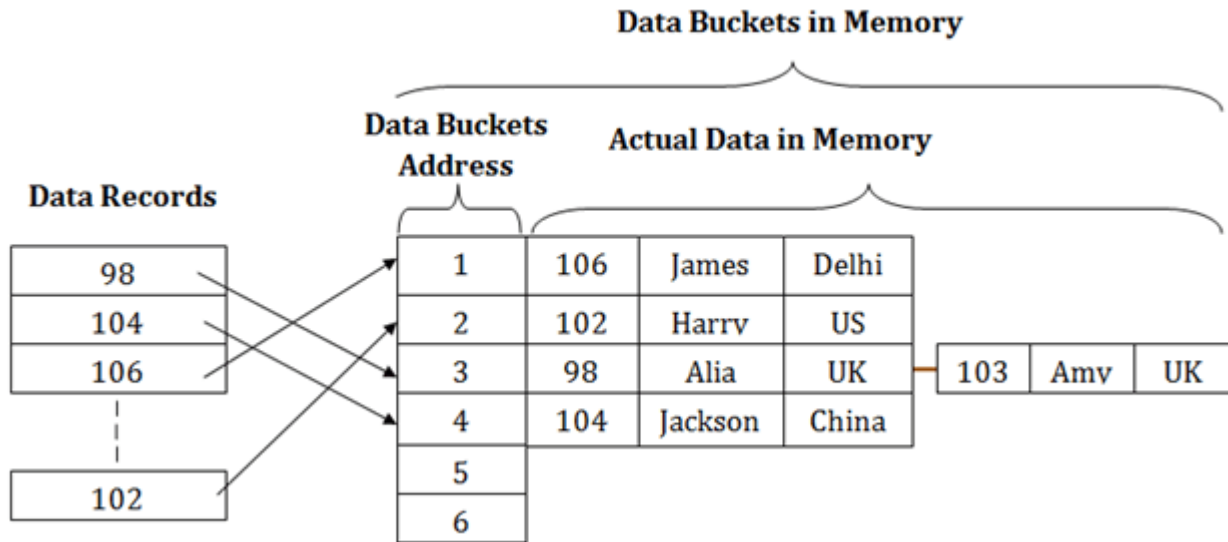
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

In this technique, data is stored at the data blocks whose address is generated by using the hashing function. The memory location where these records are stored is known as data bucket or data blocks.

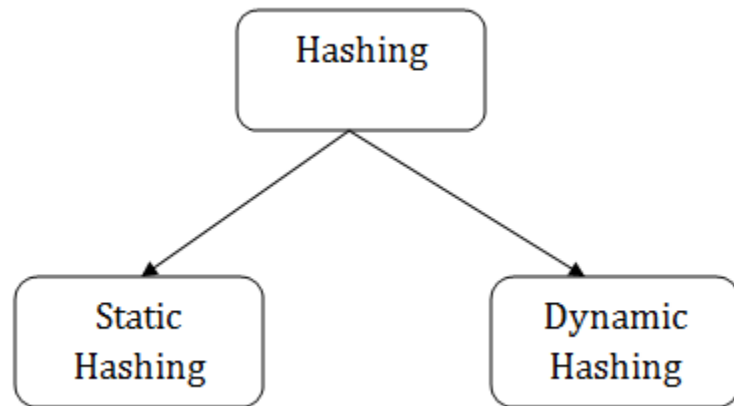
In this, a hash function can choose any of the column value to generate the address. Most of the time, the hash function uses the primary key to generate the address of the data block. A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block. That means each row whose address will be the same as a primary key stored in the data block.



The above diagram shows data block addresses same as primary key value. This hash function can also be a simple mathematical function like exponential, mod, cos, sin, etc. Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.



### Types of Hashing:



- [Static Hashing](#)
- [Dynamic Hashing](#)

## What is Data Structure?

- Data structure is an arrangement of data in computer's memory. It makes the data quickly available to the processor for required operations.
- It is a software artifact which allows data to be stored, organized and accessed.
- It is a structure program used to store ordered data, so that various operations can be performed on it easily. **For example**, if we have an employee's data like name 'ABC' and salary 10000. Here, 'ABC' is of String data type and 10000 is of Float data type.

We can organize this data as a record like Employee record and collect & store employee's records in a file or database as a data structure like 'ABC' 10000, 'PQR' 15000, 'STU' 5000.

- Data structure is about providing data elements in terms of some relationship for better organization and storage.
- It is a specialized format for organizing and storing data that can be accessed within appropriate ways.

## Why is Data Structure important?

- Data structure is important because it is used in almost every program or software system.
- It helps to write efficient code, structures the code and solve problems.
- Data can be maintained more easily by encouraging a better design or implementation.

- Data structure is just a container for the data that is used to store, manipulate and arrange. It can be processed by algorithms.

**For example**, while using a shopping website like Flipkart or Amazon, the users know their last orders and can track them. The orders are stored in a database as records.

However, when the program needs them so that it can pass the data somewhere else (such as to a warehouse) or display it to the user, it loads the data in some form of data structure.

## Types of Data Structure

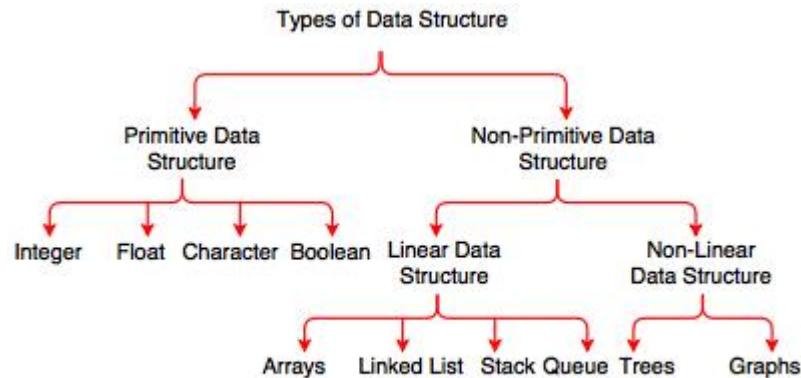


Fig. Types of Data Structure

### A. Primitive Data Type

- Primitive data types are the data types available in most of the programming languages.
- These data types are used to represent single value.
- It is a basic data type available in most of the programming language.

| Data type | Description |
|-----------|-------------|
|-----------|-------------|



|           |                                                        |
|-----------|--------------------------------------------------------|
| Integer   | Used to represent a number without decimal point.      |
| Float     | Used to represent a number with decimal point.         |
| Character | Used to represent single character.                    |
| Boolean   | Used to represent logical values either true or false. |

## B. Non-Primitive Data Type

- Data type derived from primary data types are known as Non-Primitive data types.
- Non-Primitive data types are used to store group of values.

**It can be divided into two types:**

1. Linear Data Structure
2. Non-Linear Data Structure

### 1. Linear Data Structure

- Linear data structure traverses the data elements sequentially.
- In linear data structure, only one data element can directly be reached.
- It includes array, linked list, stack and queues.

| Types       | Description                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arrays      | Array is a collection of elements. It is used in mathematical problems like matrix, algebra etc. each element of an array is referenced by a subscripted variable or value, called subscript or index enclosed in parenthesis. |
| Linked list | Linked list is a collection of data elements. It consists of two parts: Info and Link. Info gives information and Link is an address of next node. Linked list can be implemented by using pointers.                           |

|       |                                                                                                                                                                                                                                                                                                     |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Stack | Stack is a list of elements. In stack, an element may be inserted or deleted at one end which is known as Top of the stack. It performs two operations: Push and Pop. Push means adding an element in stack and Pop means removing an element in stack. It is also called Last-in-First-out (LIFO). |
| Queue | Queue is a linear list of element. In queue, elements are added at one end called rear and the existing elements are deleted from other end called front. It is also called as First-in-First-out (FIFO).                                                                                           |

## 2. Non-Linear Data Structure

- Non-Linear data structure is opposite to linear data structure.
- In non-linear data structure, the data values are not arranged in order and a data item is connected to several other data items.
- It uses memory efficiently. Free contiguous memory is not required for allocating data items.
- It includes trees and graphs.

| Type  | Description                                                                                                                                                                                                                                                               |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tree  | Tree is a flexible, versatile and powerful non-linear data structure. It is used to represent data items processing hierarchical relationship between the grandfather and his children & grandchildren. It is an ideal data structure for representing hierarchical data. |
| Graph | Graph is a non-linear data structure which consists of a finite set of ordered pairs called edges. Graph is a set of elements connected by edges. Each elements are called a vertex and node.                                                                             |

## Abstract Data type (ADT)

### What is ADT?

- ADT stands for **Abstract Data Type**.
- It is an abstraction of a data structure.

- Abstract data type is a mathematical model of a data structure.
- It describes a container which holds a finite number of objects where the objects may be associated through a given binary relationship.
- It is a logical description of how we view the data and the operations allowed without regard to how they will be implemented.
- ADT concerns only with what the data is representing and not with how it will eventually be constructed.
- It is a set of objects and operations. For example, List, Insert, Delete, Search, Sort.

**It consists of following three parts:**

1. Data
2. Operation
3. Error

**1. Data** describes the structure of the data used in the ADT.

**2. Operation** describes valid operations for the ADT. It describes its interface.

**3. Error** describes how to deal with the errors that can occur.

### **Advantages of ADT**

- ADT is reusable and ensures robust data structure.
- It reduces coding efforts.
- Encapsulation ensures that data cannot be corrupted.
- ADT is based on principles of Object Oriented Programming (OOP) and Software Engineering (SE).
- It specifies error conditions associated with operations.

## Introduction of B+ Tree

In order, to implement dynamic multilevel indexing, [B-tree](#) and B+ tree are generally employed. The drawback of B-tree used for indexing, however is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree. Thus, the structure of leaf nodes of a B+ tree is quite different from the structure of internal nodes of the B+ tree. It may be noted here that, since data pointers are present only at the leaf nodes, the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to access them. Moreover, the leaf nodes are linked to provide ordered access to the records. The leaf nodes, therefore form the first level of index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the searching of a record.

From the above discussion it is apparent that a B+ tree, unlike a B-tree has two orders, 'a' and 'b', one for the internal nodes and the other for the external (or leaf) nodes.

### **The structure of the internal nodes of a B+ tree of order 'a' is as follows:**

1. Each internal node is of the form :  
 $\langle P_1, K_1, P_2, K_2, \dots, P_{c-1}, K_{c-1}, P_c \rangle$   
where  $c \leq a$  and each  $P_i$  is a tree pointer (i.e points to another node of the tree) and, each  $K_i$  is a key value (see diagram-1 for reference).
2. Every internal node has :  $K_1 < K_2 < \dots < K_{c-1}$
3. For each search field values 'X' in the sub-tree pointed at by  $P_i$ , the following condition holds :  
 $K_{i-1} < X \leq K_i$ , for  $1 < i < c$  and,

$K_{i-1} < X$ , for  $i = c$

(See diagram I for reference)

- Each internal nodes has at most 'a' tree pointers.
- The root node has, at least two tree pointers, while the other internal nodes have at least  $\lceil a/2 \rceil$  tree pointers each.
- If any internal node has 'c' pointers,  $c \leq a$ , then it has 'c - 1' key values.

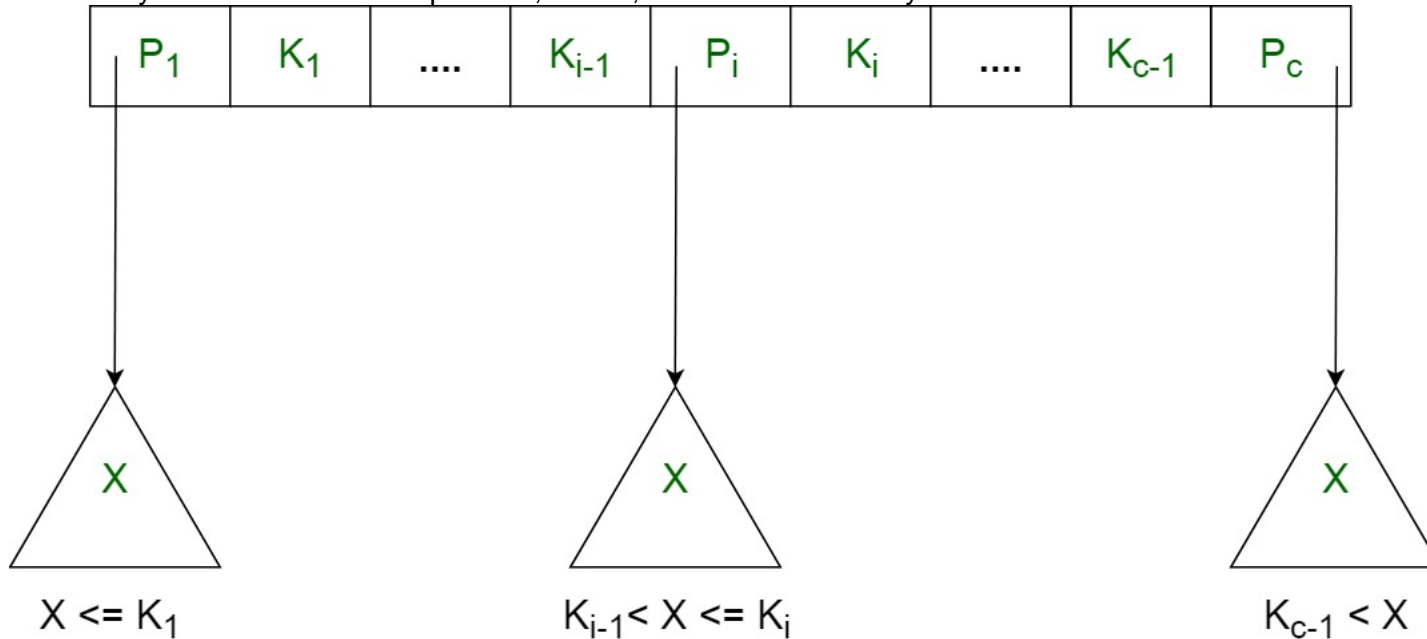


Diagram-I

**The structure of the leaf nodes of a B+ tree of order 'b' is as follows:**

- Each leaf node is of the form :  
 $\langle \langle K_1, D_1 \rangle, \langle K_2, D_2 \rangle, \dots, \langle K_{c-1}, D_{c-1} \rangle, P_{next} \rangle$   
where  $c \leq b$  and each  $D_i$  is a data pointer (i.e points to actual record in the disk whose key value is  $K_i$  or to a disk file block containing that record) and, each  $K_i$  is a key value and,  $P_{next}$  points to next leaf node in the B+ tree (see diagram II for reference).
- Every leaf node has :  $K_1 < K_2 < \dots < K_{c-1}$ ,  $c \leq b$
- Each leaf node has at least  $\lceil b/2 \rceil$  values.
- All leaf nodes are at same level.

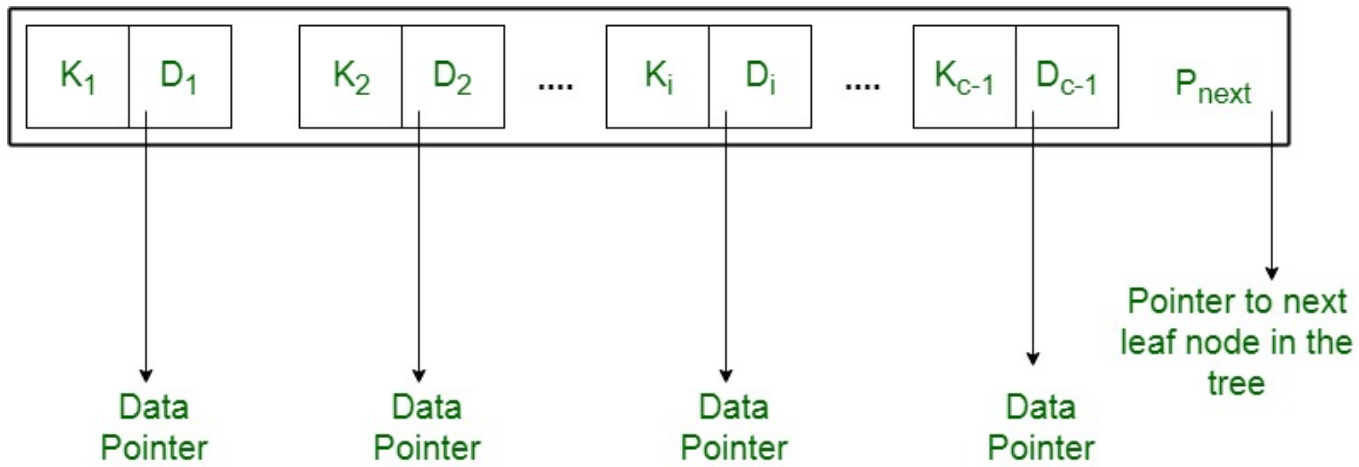
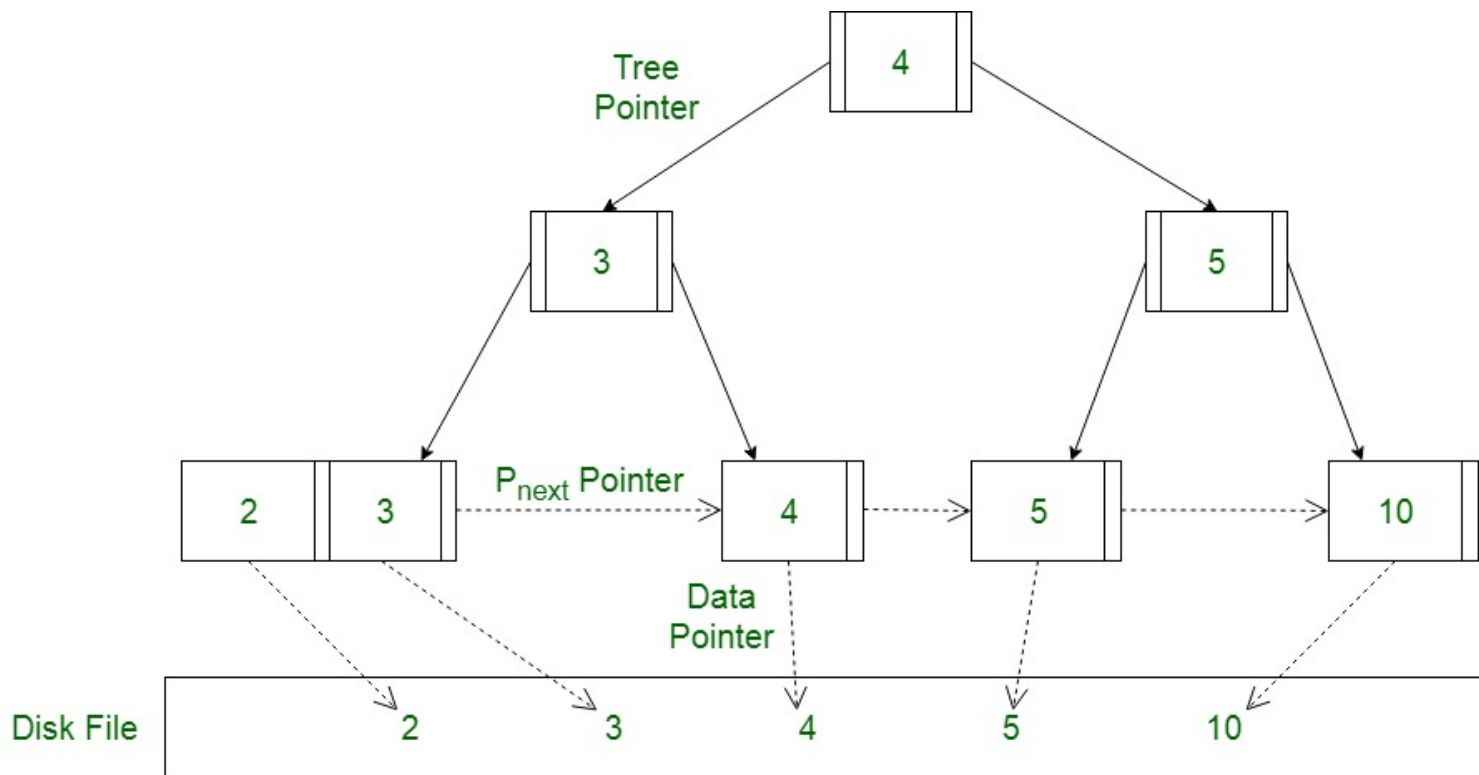


Diagram-II

Using the  $P_{next}$  pointer it is viable to traverse all the leaf nodes, just like a linked list, thereby achieving ordered access to the records stored in the disk.

**A Diagram of B+ Tree –**



**Advantage –**

A B+ tree with 'l' levels can store more entries in its internal nodes compared to a B-tree having the same 'l' levels. This accentuates the significant improvement made to the search time for any given key. Having lesser levels and presence of P<sub>next</sub> pointers imply that B+ tree are very quick and efficient in accessing records from disks.