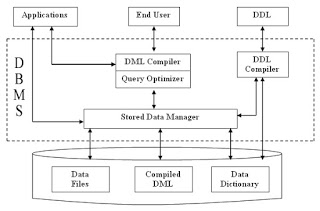
UNIT -V

File and System Structure

1. **Overall Structure**



DBMS acts as an interface between user and the database. DBMS are very large and typically divided into modules :-

**DDL Compiler -**Converts DDL statements to a set of tables containing metadata stored in a data dictionary. Metadata information can be the name of files, data items, storage details of each file, mapping information and constraints, etc.

**DML Compiler and Query Optimizer -**DML compiler translates the Data Manipulation Languages into query Engine instructions. It might also do optimization for query.  
 Query processor/optimizer translates statements in a query language into low-level instructions the database manager understands. (It is used to find an equivalent but more efficient form).  
**Data Manager -**The data manager is the central software component of the DBMS. It is sometimes referred to as the *database control system.* One of the functions of the data manager is to convert operations in the user's queries coming directly via the query processor or\ indirectly via an application program from the user's logical view to a physical file system.

The data manager is responsible for interfacing with the file system as show. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data, as well as its security, are also performed by the data manager. It is also the responsibility of the Data. Manager to provide the synchronization in the simultaneous operations performed by concurrent users and to maintain the backup and recovery operations.

**Data Dictionary** - Data Dictionary is a repository of description of data in the database. A data dictionary contains a list of all files in the database, the number of records in each file and the names and types of each field. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its content.

*Functions of the Data Dictionary-*

1. Defines the data element.
2. Helps in the scheduling.
3. Helps in the control.
4. Permits the various users who know which data is available and how can it be obtained.
5. Helps in the identification of the organizational data irregularity.
6. Acts as a very essential data management tool.
7. Provides with a good standardization mechanism.
8. Acts as the corporate glossary of the ever growing information resource.
9. Provides the report facility, the control facility along with the excerpt facility.

**Data Files** - It stores the database.

**Compiled DML -** The DML complier converts the high level Queries into low level file access commands known as compiled DML.

**End Users**- End Users are the people who interact with the database through applications or utilities. The various categories of end users are:

1.*Casual End Users* - These Users occasionally access the database but may need different information each time. They use sophisticated database Query language to specify their requests. For example: High level Managers who access the data weekly or biweekly.

2. *Native End Users* - These users frequently query and update the database using standard types of Queries. The operations that can be performed by this class of users are very limited and effect precise portion of the database. For example: - Reservation clerks for airlines/hotels check availability for given request and make reservations. Also, persons using Automated Teller Machines (ATM's) fall under this category as he has access to limited portion of the database.

3.*Standalone end Users/On-line End Users -* Those end Users who interact with the database directly via on-line terminal or indirectly through Menu or graphics based Interfaces. Example:-Library Management System.

**II. File Organization**

File organization refers to the relationship of the key of the record to the physical location of that record in the computer file. File organization may be either physical file or a logical file.

**2.1 Physical File Organization:-** A physical file is a physical unit, such as magnetic tape or a disk. **It** contains the actual data that is stored on the system, and a description of **how** data is to be presented to or received from a program. They contain only one record format, and one or more members. A physical file can have a keyed sequence access path. This means that data is presented to a program in a sequence based on one or more key fields in the file.

* Occupies the portion of memory. It contains the original data.
* A physical file contains one record format
* Can exist even without LF(Logical File)
* If there is a logical file for a PF (Physical File), the PF can’t be deleted until and unless we delete the LF.
* **CRTPF (create physical file)** command is used to create such object

**2.2 Logical File Organization :-** A logical file on the other hand is a complete set of records for a specific application or purpose. It do not contain data. They contain a description of records found in one or more physical files. A logical file is a *view or representation of one or more physical files*. Logical files that contain more than one format are referred to as **multi-format** logical files.

* Does not occupy any memory space. Does not contain any data. It loads itself at run time as per the defined access path.
* A logical file can contain up to 32 record formats.
* Can’t exist without PF
* If there is a logical file for a PF, the LF can be deleted without deleting the PF.
* **CRTLF (create logical file)** command is used to create such type object

**Difference between physical file and logical file**

|  |  |
| --- | --- |
| **Physical file** | **Logical file** |
| 1. Occupies the portion of memory. It contains the original data. | 1. Does not occupy any memory space. Does not contain any data. It loads itself at run time as per the defined access path. |
| 2. A physical file contains one record format | 2. A logical file can contain up to 32 record formats. |
| 3.Can exist even without LF | 3. Can’t exist without PF |
| 4. If there is a logical file for a PF, the PF can’t be deleted until and unless we delete the LF. | 4. If there is a logical file for a PF, the LF can be deleted without deleting the PF. |
| 5.**CRTPF** command is used to create such object | **CRTLF** command is used to create such type object |

Typical DBMS applications need a small subset of the DB at any given time. when a portion of the data is needed it must be located on disk, copied to memory for processing and rewritten to disk if the data was modified

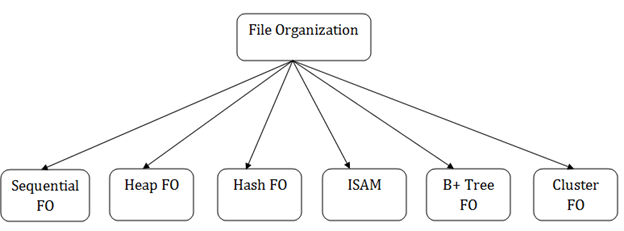
* 1. **Types of File Organization**

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.**

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.

Different types of File Organizations are :



* **Sequential File Organization**
* **Heap File Organization**
* **Random or Direct File Organization/Hash File Organization**
* **Indexed File Organization**
* **B+ Tree File Organization**
* **Clustered File Organization**

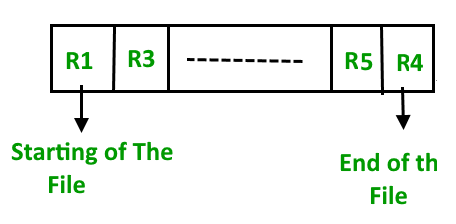
The description is as follow:-

**1. Sequential File Organization –**

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

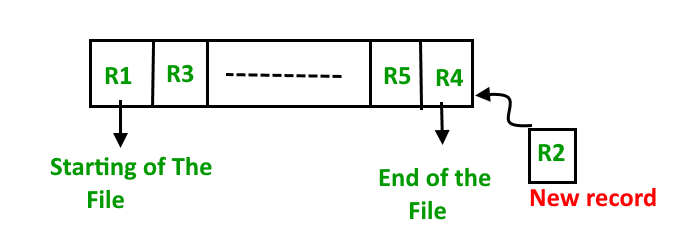
**1.2 Pile File Method** –In this method we store the records in a sequence i.e one after other in

**the order in which they are inserted** into the tables.

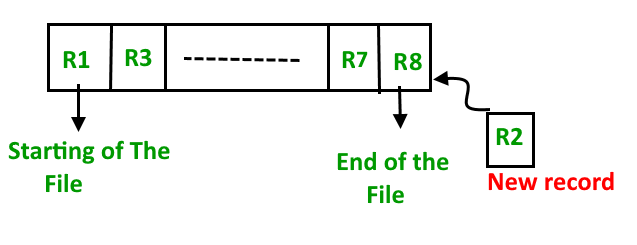


**(i) Insertion of new record –**

Let the R1, R3 and so on upto 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.

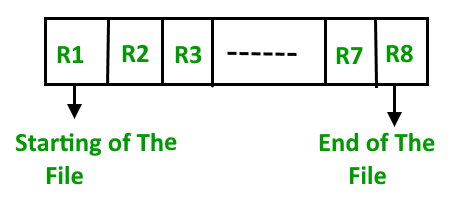


**1.2 Sorted File Method** –In this method, whenever a new record has to be inserted, it is always inserted in a sorted **(ascending or descending)** manner. Sorting of records may be based on any primary key or any other key.



**(i)Insertion of new record –**

Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto 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** –

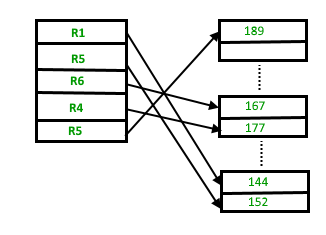
* Fast and efficient method for huge amount of data.
* Simple design.
* Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

**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.

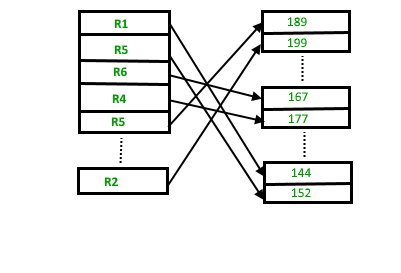
**2 Heap File Organization –**

Heap File Organization works with **data bloc**ks. In this method, records are inserted at the **end of the file**, into the data blocks. **No Sorting or Ordering** is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.



**2.1 Insertion of new record –**

Suppose we have four records in the heap R1, R5, R6, R4 and R3 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, lets say data block 1.



If we want to search, delete or update data in heap file Organization then 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. Indexed File Organization :-**

**3. Random Files**

In random file organization, records are stored in random order within the file.  Though there is no sequencing to the placement of the records, there is however, a pre-defined relationship between the key of the record and its location within the file.

 In other words, the value of the record key is **mapped** by an established function to the address within the file where it resides.  Therefore, any record within the file can be directly accessed through the **mapping function** in roughly the same amount of time.  The location of the record within the file therefore is not a factor in the access time of the record.  As such, random files are also known as **direct access files.**

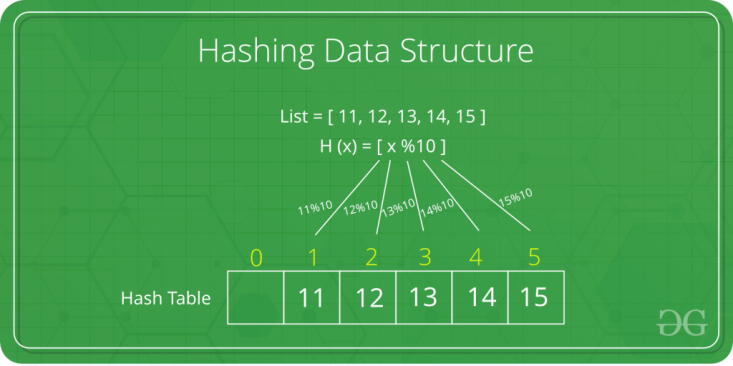
To create and maintain a random file, a mapping function must be established between the record key and the address where the record is held.  If M is the mapping function, then

M(value of record key) => address of record

**Hashing Data Structure**

Hashing is an important Data Structure which is designed to use a special function called the **Hash function** which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

Eg :-Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



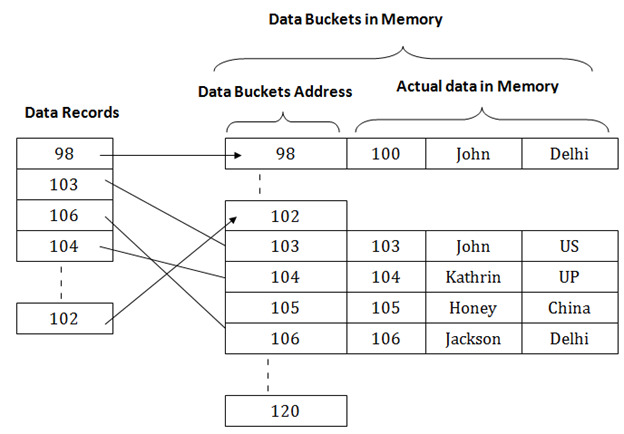
**Hashing**

There are various mapping techniques.  Some involve using the key field to directly map to the location with the file, while others refer to some lookup table for the location.  However, the more common method is to employ a **hash function** to derive the address.

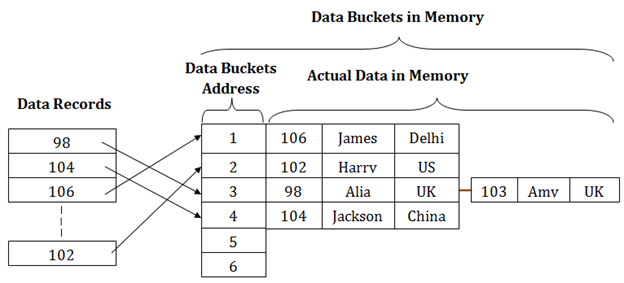
In a huge database structure, it is very inefficient to search all the index values and reach the desired data. Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.

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.

****

 A **hash function** generates the record address by performing some simple operations on the key or parts of the key.  A good hashing function should be

• quick to calculate

• cover the complete range of the address space.

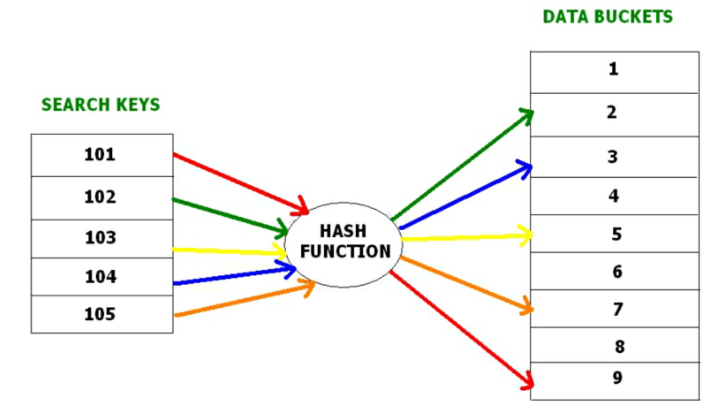
• give an even distribution

• not generate addresses that tend to cluster within a few locations, thus resulting in frequent collisions.

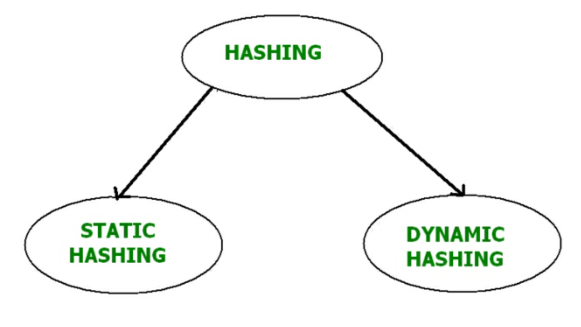
**3.1 Hash File Organization:**

* **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:



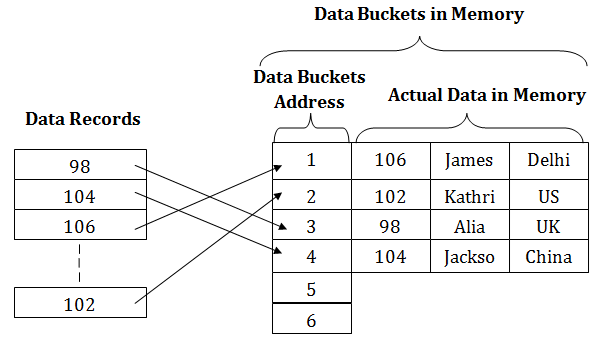
Hashing is further divided into two sub categories :



**(i) Static Hashing –**

In static hashing, the resultant data bucket address will always be the same. That means if we generate an address for EMP\_ID =103 using the hash function mod (5) then it will always result in same bucket address 3. Here, there will be no change in the bucket address.

Hence in this static hashing, the number of data buckets in memory remains constant throughout. In this example, we will have five data buckets in the memory used to store the data.



**Operations –**

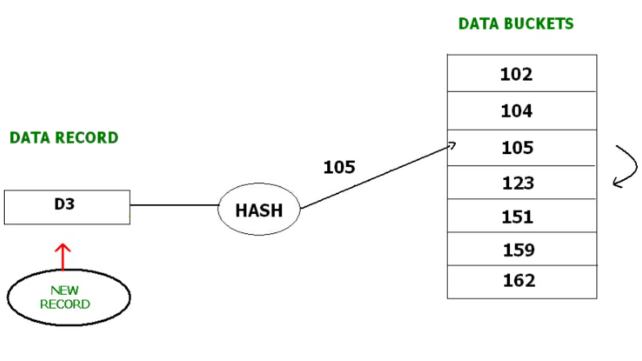
* **Insertion –**When a new record is inserted into the table, The hash function h generate a bucket address for the new record based on its hash key K. Bucket address = h(K)
* **Searching –**When a record needs to be searched, The same hash function is used to retrieve the bucket address for the record. For Example, if we want to retrieve whole record for ID 76, and if the hash function is mod (5) on that ID, the bucket address generated would be 1. Then we will directly got to address 1 and retrieve the whole record for ID 104. Here ID acts as a hash key.
* **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.
* **Updation –**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?

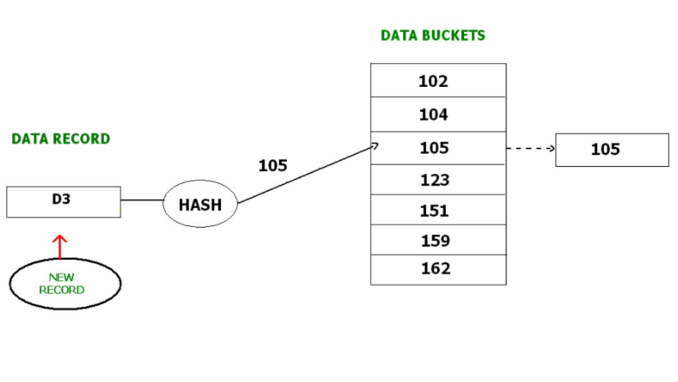
There are several methods provided to overcome this situation. Some commonly used methods are discussed below:

**a. Open Hashing or Linear Probing –**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.



**b. 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 is 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.



**c. 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.

**d. 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.

**(ii) Dynamic Hashing**

* The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
* In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
* This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.

**How to search a key**

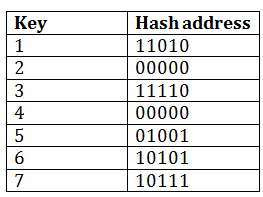
* First, calculate the hash address of the key.
* Check how many bits are used in the directory, and these bits are called as i.
* Take the least significant i bits of the hash address. This gives an index of the directory.
* Now using the index, go to the directory and find bucket address where the record might be.

**How to insert a new record**

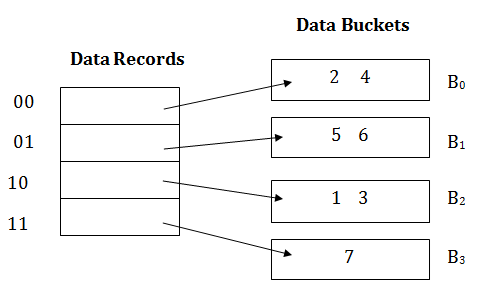
* Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
* If there is still space in that bucket, then place the record in it.
* If the bucket is full, then we will split the bucket and redistribute the records.

**For example:**

Consider the following grouping of keys into buckets, depending on the prefix of their hash address:

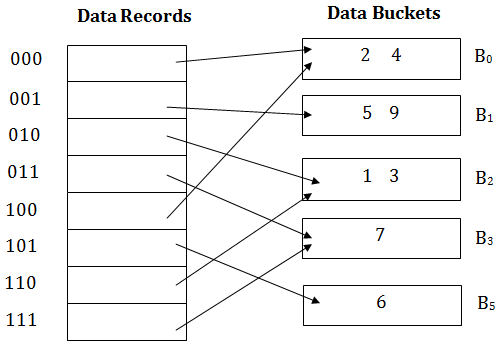
****

The last two bits of 2 and 4 are 00. So it will go into bucket B0. The last two bits of 5 and 6 are 01, so it will go into bucket B1. The last two bits of 1 and 3 are 10, so it will go into bucket B2. The last two bits of 7 are 11, so it will go into B3.

****

**Insert key 9 with hash address 10001 into the above structure:**

* Since key 9 has hash address 10001, it must go into the first bucket. But bucket B1 is full, so it will get split.
* The splitting will separate 5, 9 from 6 since last three bits of 5, 9 are 001, so it will go into bucket B1, and the last three bits of 6 are 101, so it will go into bucket B5.
* Keys 2 and 4 are still in B0. The record in B0 pointed by the 000 and 100 entry because last two bits of both the entry are 00.
* Keys 1 and 3 are still in B2. The record in B2 pointed by the 010 and 110 entry because last two bits of both the entry are 10.
* Key 7 are still in B3. The record in B3 pointed by the 111 and 011 entry because last two bits of both the entry are 11.

****

**Advantages of dynamic hashing**

* In this method, the performance does not decrease as the data grows in the system. It simply increases the size of memory to accommodate the data.
* In this method, memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.
* This method is good for the dynamic database where data grows and shrinks frequently.

**Disadvantages of dynamic hashing**

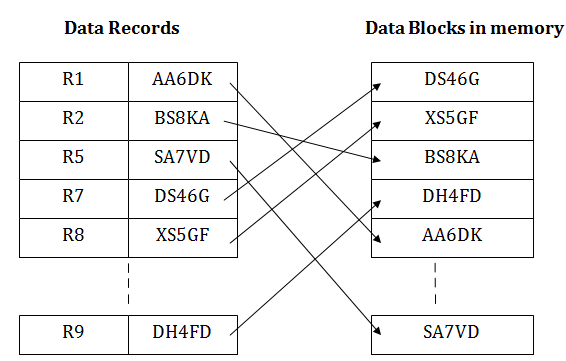
* In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as buckets grow and shrink. If there is a huge increase in data, maintaining the bucket address table becomes tedious.
* In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than static hashing.

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexity than indexing. All hash operations are done in constant time.

**4. 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.

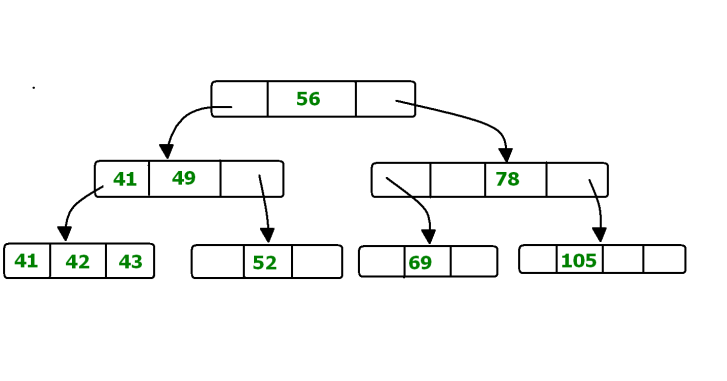
**Cons of ISAM**

* 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.

**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.



In the above diagram 56 is the root node which is also called the main node of the tree.  
The intermediate nodes here, just consist the address of leaf nodes. They do not contain any actual record. Leaf nodes consist of the actual record. All leaf nodes are balanced.

**Pros and Cons of B+ Tree File Organization –**

**Pros –**

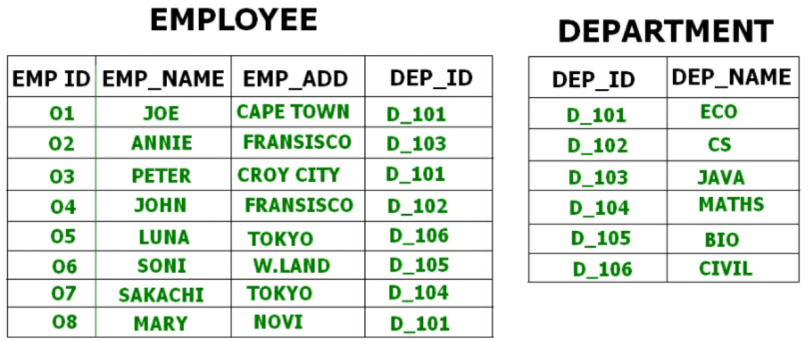
* Tree traversal is easier and faster.
* Searching becomes easy as all records are stored only in leaf nodes and are sorted sequential linked list.
* There is no restriction on B+ tree size. It may grows/shrink as the size of data increases/decreases.

**Cons –**

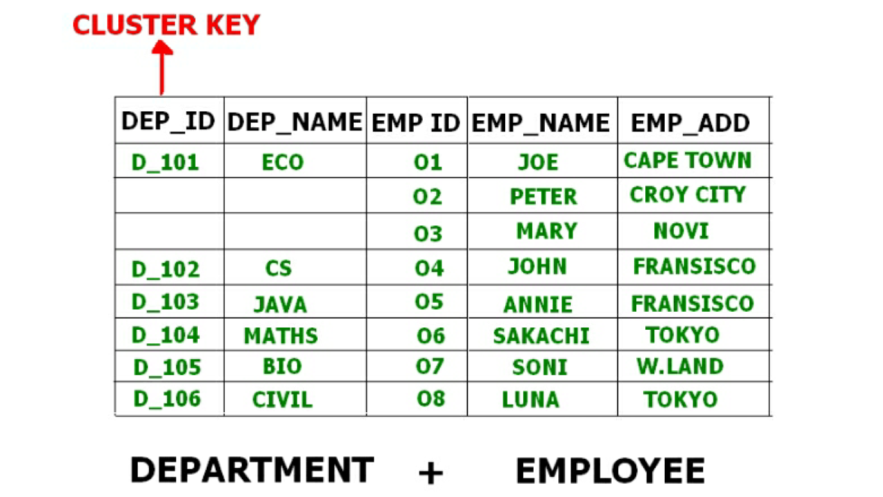
* Inefficient for static tables.

### 6. Cluster File Organization –In cluster file organization, two or more related tables/records are stored within same file known as *clusters*. These files will have two or more tables in the same data block and the key attributes which are used to map these table together are stored only once.

Thus it lowers the cost of searching and retrieving various records in different files as they are now combined and kept in a single cluster. For example we have two tables or relation Employee and Department. These table are related to each other.



Therefore these table are allowed to combine using a join operation and can be seen in a cluster file.



If we have to insert, update or delete any record we can directly do so. Data is sorted based on the primary key or the key with which searching is done. **Cluster key** is the key with which joining of the table is performed.

**Types of Cluster File Organization –** There are two ways to implement this method:

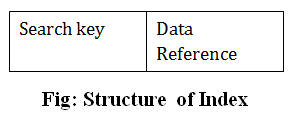
1. **Indexed Clusters –** In Indexed clustering the records are group based on the cluster key and stored together. The above mentioned example of Employee and Department relationship is an example of Indexed Cluster where the records are based on the Department ID.
2. **Hash Clusters –** This is very much similar to indexed cluster with only difference that instead of storing the records based on cluster key, we generate hash key value and store the records with same hash key value.

III. Indexing

* 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.

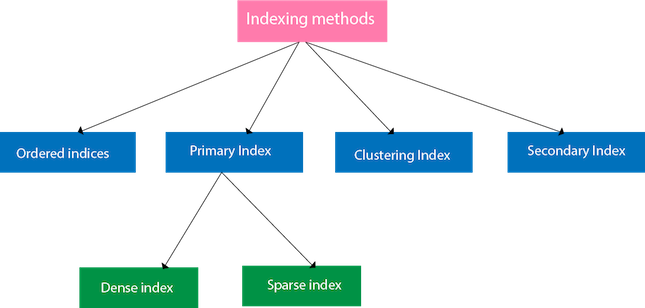
**1. Index structure:**

Indexes can be created using some database columns.

* 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.

**2. Indexing Methods**



**2.1 Ordered indices**

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

**Example**: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

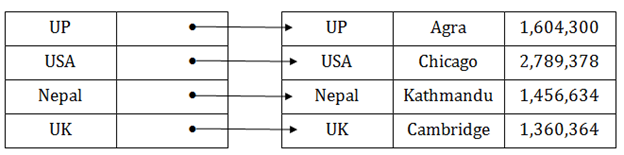
* In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543\*10=5430 bytes.
* In the case of an index, we will search using indexes and the DBMS will read the record after reading 542\*2= 1084 bytes which are very less compared to the previous case.

**2.2 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.*

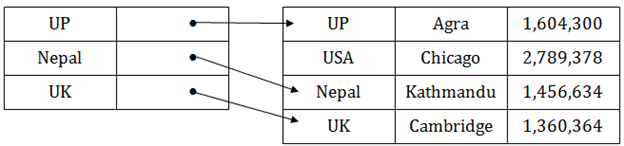
**2.2.1 Dense index**

* 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.



**2.2.2 Sparse index**

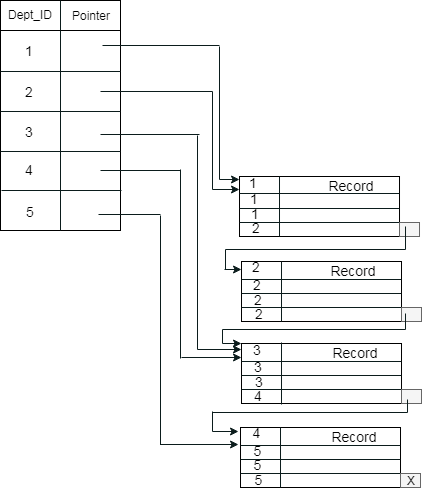
* In the data file, index record appears only for a few items. Each item points to a block.
* In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



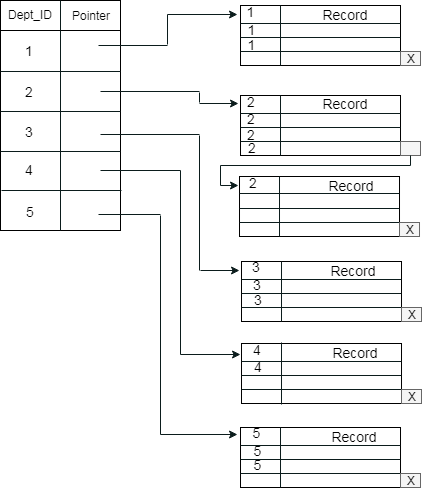
**3. Clustering Index**

* 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.

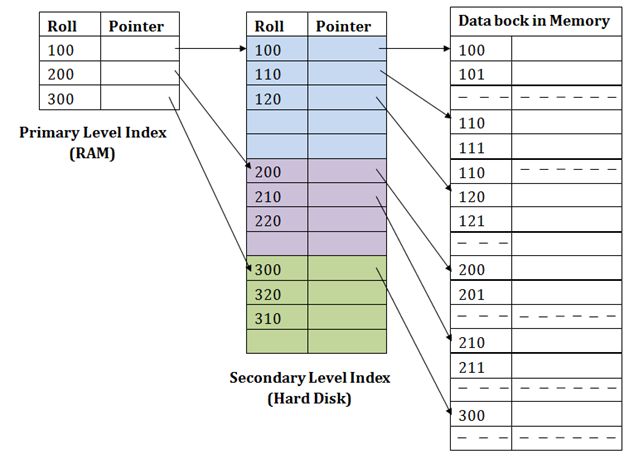
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



**4.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).



**For example:** If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.

* Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
* This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

**IV. B-Tree :-**

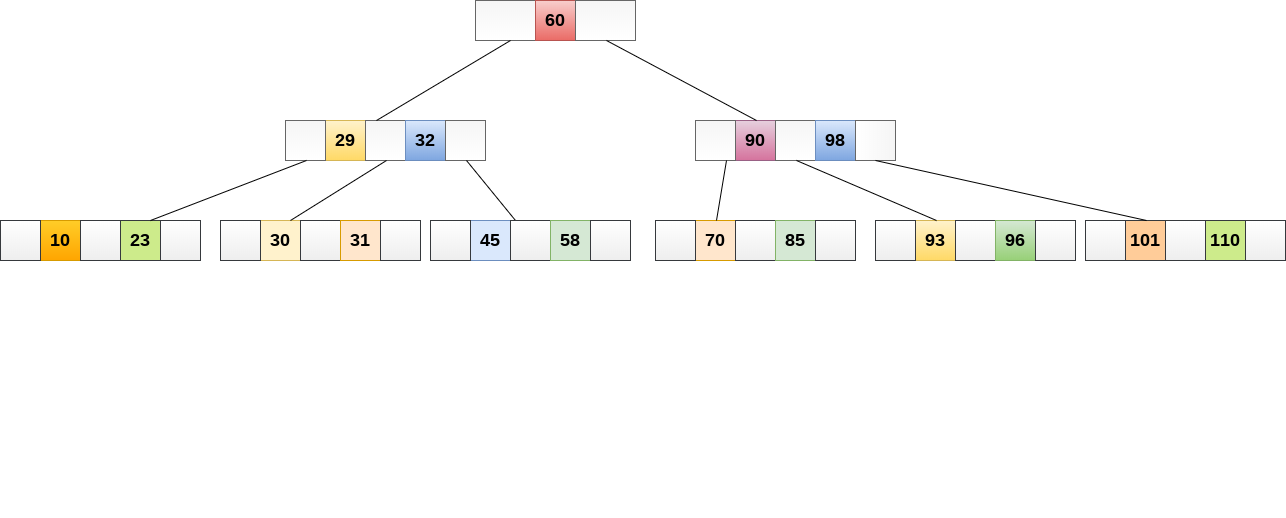
B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

A B tree of order 4 is shown in the following image.



While performing some operations on B Tree, any property of B Tree may violate such as number of minimum children a node can have. To maintain the properties of B Tree, the tree may split or join.

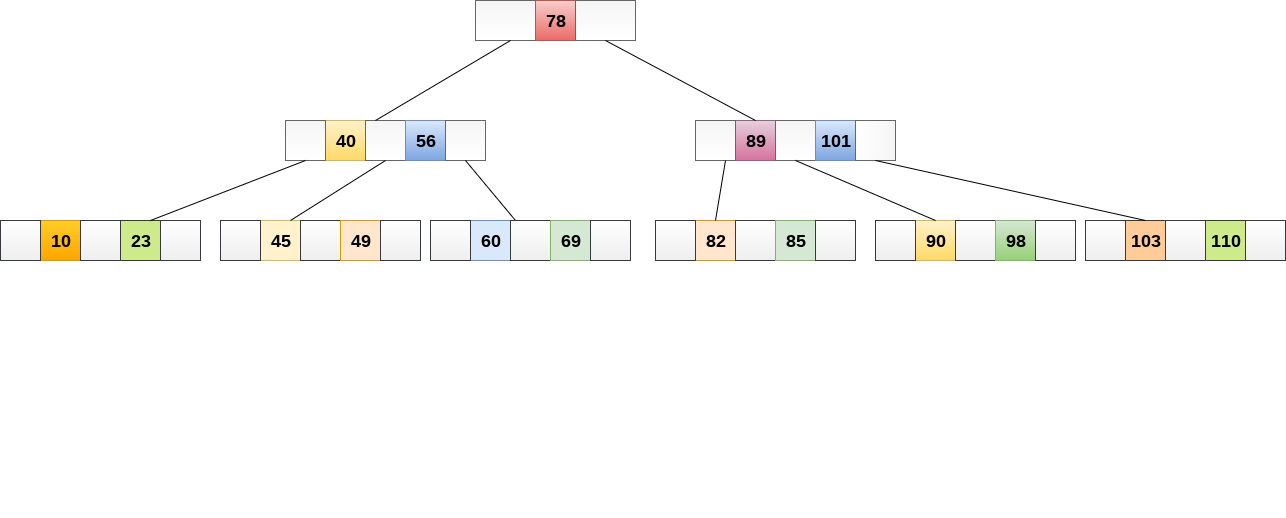
**Operations**

**Searching :**

Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following :

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Searching in a B tree depends upon the height of the tree. The search algorithm takes O(log n) time to search any element in a B tree.



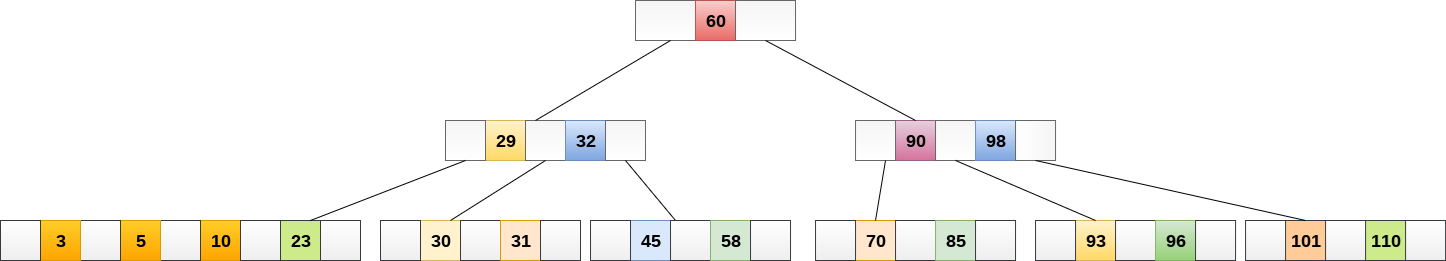
**Insertion**

Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.

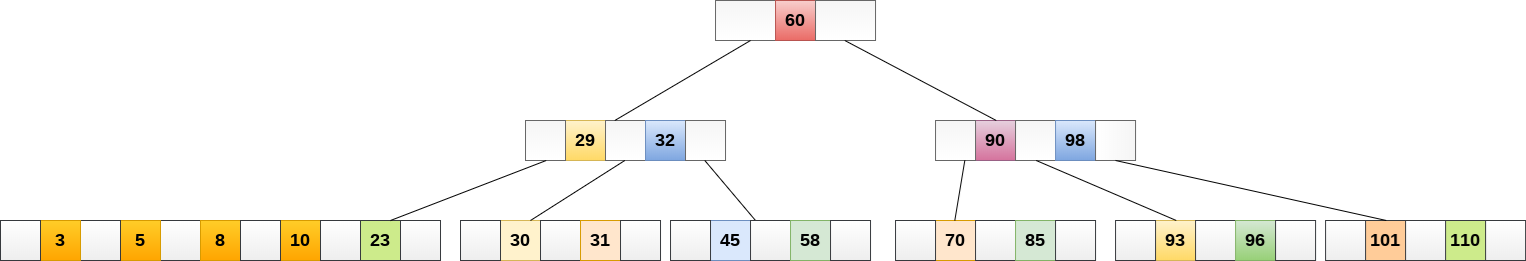
1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
3. Else, if the leaf node contains m-1 keys, then follow the following steps.
   * Insert the new element in the increasing order of elements.
   * Split the node into the two nodes at the median.
   * Push the median element upto its parent node.
   * If the parent node also contain m-1 number of keys, then split it too by following the same steps.

**Example:**

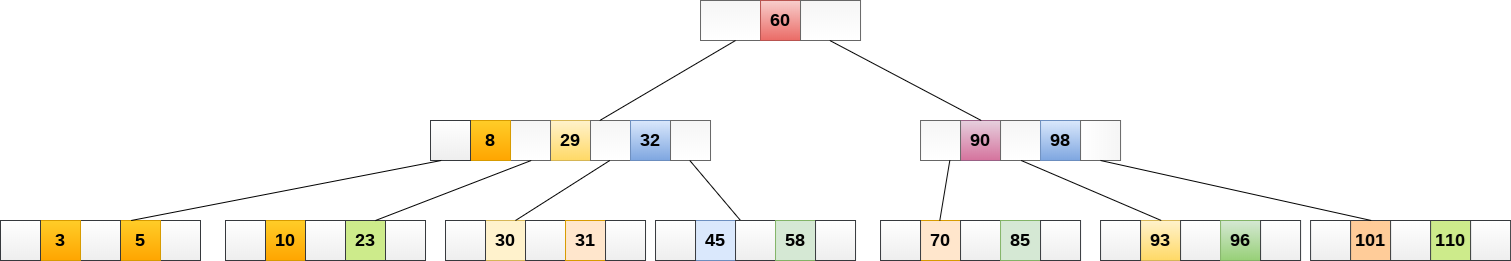
Insert the node 8 into the B Tree of order 5 shown in the following image.



8 will be inserted to the right of 5, therefore insert 8.



The node, now contain 5 keys which is greater than (5 -1 = 4 ) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.



**Deletion**

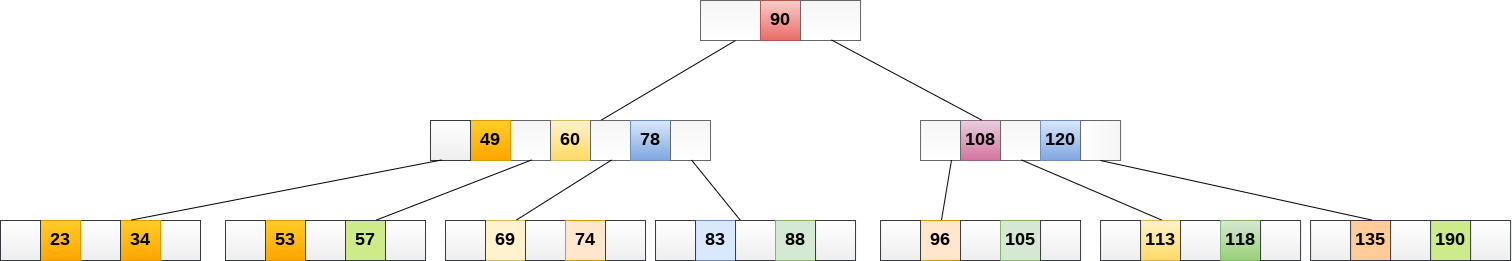
Deletion is also performed at the leaf nodes. The node which is to be deleted can either be a leaf node or an internal node. Following algorithm needs to be followed in order to delete a node from a B tree.

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.
   * If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
   * If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.
4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

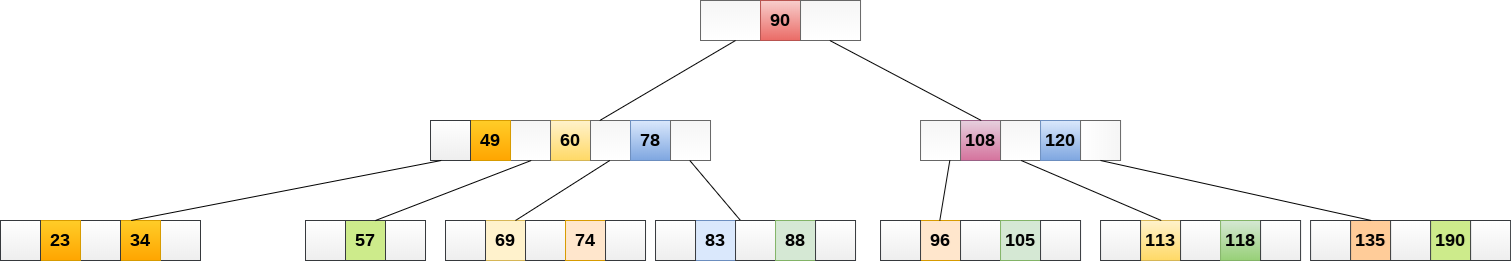
If the the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

**Example 1**

Delete the node 53 from the B Tree of order 5 shown in the following figure.

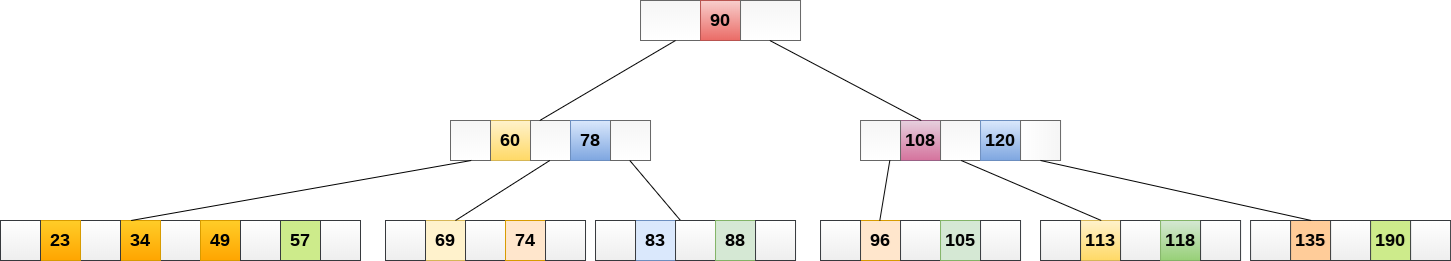


53 is present in the right child of element 49. Delete it.



Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2. it is less than that, the elements in its left and right sub-tree are also not sufficient therefore, merge it with the left sibling and intervening element of parent i.e. 49.

The final B tree is shown as follows.



**Application of B tree**

* B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.
* Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

## B Tree VS B+ Tree

|  |  |  |
| --- | --- | --- |
| **SN** | **B Tree** | **B+ Tree** |
| 1 | Search keys can not be repeatedly stored. | Redundant search keys can be present. |
| 2 | Data can be stored in leaf nodes as well as internal nodes | Data can only be stored on the leaf nodes. |
| 3 | Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes. | Searching is comparatively faster as data can only be found on the leaf nodes. |
| 4 | Deletion of internal nodes are so complicated and time consuming. | Deletion will never be a complexed process since element will always be deleted from the leaf nodes. |
| 5 | Leaf nodes can not be linked together. | Leaf nodes are linked together to make the search operations more efficient. |