

File Structure & Hashing

6.1 Record Organization

Database storage structure

<u>Physical</u>	<u>Logical</u>
It can be seen and operated from the OS.	It can be created and recognized by oracle database.
Physical files gets stored in the disk data present on your system & how those files are stored & managed in the disk handled by physical database storage.	OS has no interaction with the logical database.
	Relationship between data are seen by logical db struct.

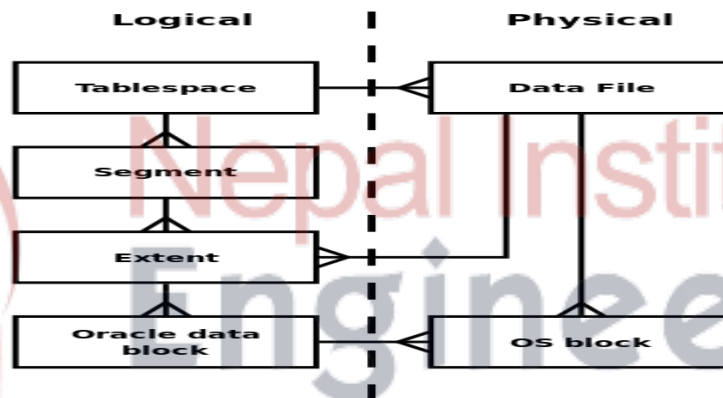


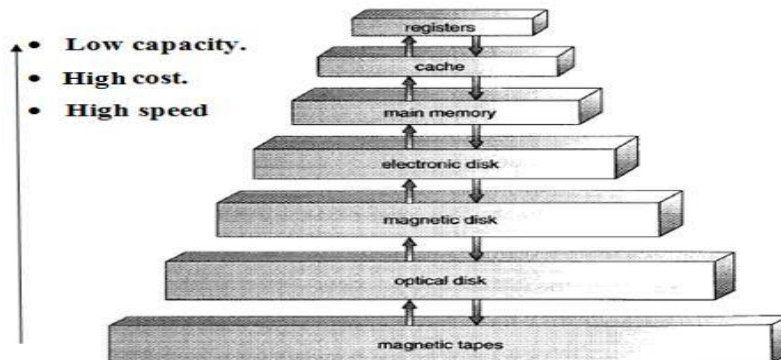
Figure : Logical & Physical storage

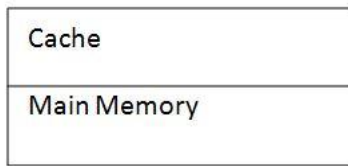
Physical Storage Structure

Classified into different types based on :

- accessing speed
- cost per unit of data
- reliability

Storage device hierarchy (according to speed and cost)





Primary memory fastest, most expensive and computer system hardware, volatile

Flash -

Secondary memory non-volatile , read is fast, write/erase is slow in flash memory.

Magnetic Disk -

Non-volatile, large amount of data is stored, data must be moved from disk to main memory for access & written back to storage

Magnetic Tape -

Tertiary memory. It is used mainly for backups. It holds large volume of data.

Optical Disk -

Non-volatile. Data is read optically from a spinning disk using a laser.

CD/DVD most popular form of optical disk.

Record Organization

Databases are stored as a collection of files.

Each file is a sequence of records.

Records can be of 2 types mainly :

Fixed Length Record : Those records which has fixed sizes.

Let us consider a file of "Account records" for our bank database, which has the attributes -
type . deposit = record

```
account_no :char(10);
branch_name : char(22);
balance : real;
```

end

Let us assume

1 char = 1 Byte

1 real = 8 Byte

So our one records becomes

account_no = 10*1 = 10 Byte

branch_name = 22*1 = 22 Byte

balance = 8*1 = 8 Byte

Total = 40 Bytes

Table

<u>A/c No</u>	<u>Branch_name</u>	<u>Balance</u>
A_101	Kathmandu	1800
A_102	Bhaktpur	2000
A_103	Kalimati	8100
A_104	Kalanki	6900
A_105	Balkhu	7000

Then we follow an approach :

- use a first 40 bytes for the first record
- the next 40 bytes for the second records

There are two problems with this approach


1. Deleting the record from this will create an empty block which may occur waste of memory space.

So to overcome this problem we must fill this block with record of the file, or we must have way of marking deleted records so that can be ignored.

If block size is less than 40 bytes.

So we have to store records in 2 or more block according to block size. But for this kind of write & read operations become tedious.

To save the space of deleted records.



<u>A/c No</u>	<u>Branch_name</u>	<u>Balance</u>
A_101	Kathmandu	1800
A_102	Kalimati	2000
A_103	Bhaktpur	3000
A_105	Kalanki	3500
A_106	Balkhu	4500
A_109	Kupondole	3500
A_110	Tripureshwor	8100

First Approach

- Move every record up to the deleted record block
- It takes more time & effort to move every record till end of file


Second Approach

- Move the last record to the deleted record block
- Need additional block access

At the beginning of the file , we allocate a certain number of bytes as a 'file header'. this file header contains a variety of information about the file with the address of the records whose contents are deleted.

We use this first record to store the address of the second available record & so on.

	<u>A/c No</u>	<u>Branch_name</u>	Balance
Header			
record 0	A_101		
1	A_103		
2	A_105		
3	A_106		
4			
5	A_108		
6			
7	A_109		



We can assume these stored addresses as pointer as they point to the location of a record.

The deleted records thus form a linked list or a free list.

On insertion of new record we use record pointed to by the header and we change the header pointer to point next available records.

If no space is available, we add new record to the end of the file.

[Fixed Length Record : All records in the file are of same size. It can lead to memory wastage.

Access of record is easier and faster. Exact location of the record can be determined , location of i_{th} record would be $n*(i-1)$, where n is the size of every record.]

Variable Length Record : In this case

- Different records in the file have different sizes.
- Memory efficient
- Access of the record is slower

Variable length records arise in database system in different ways

- When multiple record types have to be stored in a file.
- Storing records which allow variable length for one or more fields.
- Records which allow repeating fields.

Let us take an example for variable length records

```

type deposit-list = record
    branch-name : char(20);
    account-info : array[1...∞] of record;
    account# : char(20);
    customer-name : char(20);
    balance : real;
end

```

Account info is an array with an arbitrary number of element. While deleting any records it becomes difficult to find size of record in table.

2 Approaches

1. Byte string representation

- Uses the technique of attaching a special end of symbol(\perp) to the end of each record.

Then we can store each record as a string of successive bytes.

Ktm	A_101	1000	A_551	3500	A_108	\perp
Bhaktpur	A_102	2000	A_681	4000	\perp	
Kalanki	A_216	3500	\perp			
Kalimati	A_300	4500	\perp			
Balkhu	A_301	6000	\perp			
Siphal	A_258	7000	\perp			

Byte string representation has several disadvantages:

- It is not easy to re-use space left by a deleted record
- In general, there is no space for records to grow longer. (Must move to expand, and record may be pinned.)

So this method is not usually used.

To overcome this problem in byte string representation, a modified byte string representation called slotted page structure is used.



Header contains

- number of records entries
- end of free space in block
- location and size of each record

- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.

- Pointers should not point directly to record - instead they should point to the entry for the record in header.

There is another way to implement variable length records efficiently in a file system.

By using one or more fixed length records

We have two Approaches of doing this

1. Reserved Space : We can use fixed-length records large enough to accommodate the largest variable-length record. Here unused space in shorter records are filled with **null or end of record symbol**.


The reserved space method requires the selection of some maximum record length.

- If most records are of near-maximum length this method is useful.
- Otherwise, space is wasted.

Kathmandu	A_892	1800	A_601	3000	A_804	1800
Kalanki	A_365	2000	A_701	3500	⊥	⊥
Balkhu	A_542	3000	⊥	⊥	⊥	⊥
Siphal	A_691	3500	⊥	⊥	⊥	⊥
Bhaktpur	A_881	4100	⊥	⊥	⊥	⊥

2. List representation (Pointer method)

- A variable length record is represented by a list of fixed length record, chained together via pointers.
- It can be used even if the maximum record length is not known.



0	Perryridge	A-102	400	
1	Round Hill	A-305	350	
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4	Redwood	A-222	700	
5		A-201	900	
6	Brighton	A-217	750	
7		A-110	600	
8		A-218	700	

Disadvantage is that space is wasted in successive records in a chain as non-repeating fields are still present.

To overcome this disadvantage, we can split records in to 2 blocks.

- **Anchor block** - contains first records of a chain
- **Overflow block** - contains records other than first in the chain.

Now all records in a block have the same length, and there is no wasted space.



File Organization

Storing the files in certain order is called file organization.

Main objectives of file organization is :

- Records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should not harm other records.
- No duplicate records should be induced as a result of insert, delete or update.
- Records should be stored efficiently so that soct storage is minimal.

Some of the file organization are :

1. Sequential file organization : Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization records are placed in the file in some sequential order based on the unique key of search key. Practically, it is not possible to store all the records sequentially in physical form.

Advantages

Cheaper

Fast & efficient when there is large volumes of data, report generation, statistical calculations etc.

Disadvantages

Sorting of data each time for insert/delete/update takes time and makes system slow.

2. Heap file organization : When a file is created using heap file organization, the os allocated memory area to that file without any further according details. File records can be replaced any where in that memory area. It is the responsibility of the software to manage the records. Heap file does not support any ordering, sequencing or indexing on its own. It has simplest design.

Advantages

Storage cost is cheap

Best suited for bulk insertion and small files/tables

Disadvantages

Not suitable for large tables

Proper memory management is needed

Records are scattered in the memory and they are inefficiently used. Hence increases the memory size.

3. Hash file organization

It uses hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

Advantages

Faster access

No need to sort

Handles multiple transactions

Suitable for online transactions

Disadvantages

Accidental deletion or updation of data

Use of memory is inefficient

4. Clustered file organization

It is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block that is, the ordering of records is not based on primary key or search key.

Advantages

Best suited for frequently joined tables

Suitable for 1:m mapping

Disadvantages

Not suitable for large database

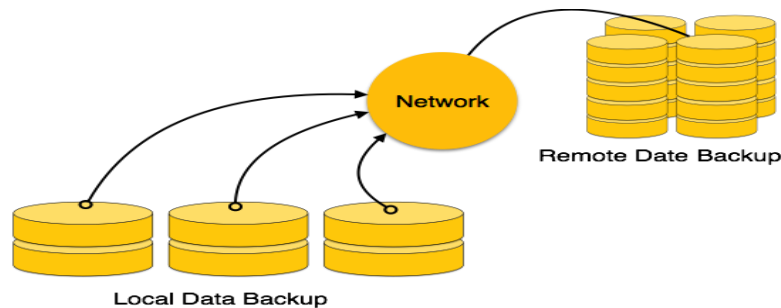
Suitable only for the joins on which clustering is done

Difference among sequential, heap/direct, hash, ISAM, B+ Tree and Cluster File Organization

	Sequential	Heap/Direct	Hash	ISAM	B+ tree	Cluster
Method of storing	Stored as they come or sorted as they come	Stored at the end of the file. But the address in the memory is random.	Stored at the hash address generated	Address index is appended to the record	Stored in a tree like structure	Frequently joined tables are clubbed into one file based on cluster key
Types	Pile file and sorted file Method		Static and dynamic hashing	Dense, Sparse, multilevel indexing		Indexed and Hash
Design	Simple Design	Simplest	Medium	Complex	Complex	Simple
Storage Cost	Cheap (magnetic tapes)	Cheap	Medium	Costlier	Costlier	Medium
Advantage	Fast and efficient when there is large volumes of data, Report generation, statistical calculations etc	Best suited for bulk insertion, and small files/tables	Faster Access No Need to Sort Handles multiple transactions Suitable for Online transactions	Searching records is faster. Suitable for large database. Any of the columns can be used as key column. Searching range of data & partial data are efficient.	Searching range of data & partial data are efficient. No performance degrades when there is insert / delete / update. Grows and shrinks with data. Works well in secondary storage devices and	Best suited for frequently joined tables. Suitable for 1:M mappings

					<p>hence reducing disk I/O.</p> <p>Since all data are at the leaf node, searching is easy.</p> <p>All data at leaf node are sorted sequential linked list.</p>	
Disadvantage	<p>Sorting of data each time for insert/delete/ update takes time and makes system slow.</p>	<p>Records are scattered in the memory and they are inefficiently used.</p> <p>Hence increases the memory size.</p> <p>Proper memory management is needed. Not suitable for large tables.</p>	<p>Accidental Deletion or updation of Data</p> <p>Use of Memory is inefficient</p> <p>Searching range of data, partial data, non-hash key column, searching single hash column when multiple hash keys present or frequently updated column as hash key are inefficient.</p>	<p>Extra cost to maintain index.</p> <p>File reconstruction is needed as insert/update/delete.</p> <p>Does not grow with data.</p>	<p>Not suitable for static tables</p>	<p>Not suitable for large database.</p> <p>Suitable only for the joins on which clustering is done.</p> <p>Less frequently used joins and 1:1 Mapping are inefficient.</p>

Remote Backup System



In computer system, we have primary and secondary memory storage. Primary memory storage device i.e. RAM is a volatile memory which stores disk buffer, active logs and other related data of a database. It stores all the recent transformations and the results too. When a query is fired, the database first fetches in the primary memory for data, if it does not exist there, then it moves to the secondary memory to fetch the record. Fetching the record from primary memory is always faster than secondary memory. What happens if the primary memory crashes? All the data in the primary memory is lost and we can not recover the database.

In such cases, we can follow any one of the following steps so that data in the primary memory are not lost.

- We can create a copy of primary memory in the database with all logs and buffers and are copied periodically into database. So in case of any failure, we will not lose all the data. We can recover the data till the point it is last copied to the database.
- We can have checkpoints created at several places so that data is copied to the database.

Suppose the secondary memory itself crashes. What happens to the data stored in it? All the data are lost and we can not recover. We have to think of some alternative solution for this because we can not afford for loss of data in huge database.

There is a method used to back up the data in the secondary memory, so that it can be recovered if there is any failure.

Remote Backup : Database copy is created and stored in the remote network. This database is periodically updated with the current database so that it will be sync with data and other details. This remote database can be updated manually called offline backup. It can be backed up online where the data is updated at current and remote database simultaneously. In this case, as soon as there is a failure of current database, system automatically switched to the remote database and starts functioning. The user will not know that there was a failure.

6.4 Hashing Concepts, Static and Dynamic Hashing

In database management, For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data.

Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure. Hashing uses hash functions with search keys as parameters to

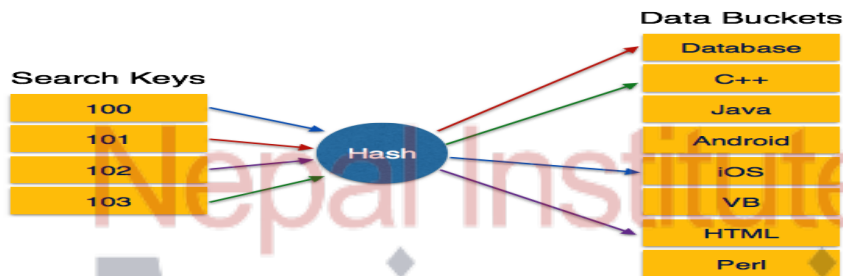
generate the address of a data record.

Hash Organization

- **Data Bucket** – A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function** – A hash function, **h**, is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Static Hashing : 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.



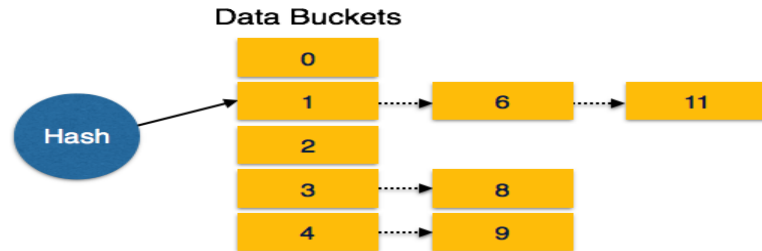
Operation

- **Insertion** – When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.
Bucket address = $h(K)$
- **Search** – When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.
- **Delete** – This is simply a search followed by a deletion operation.

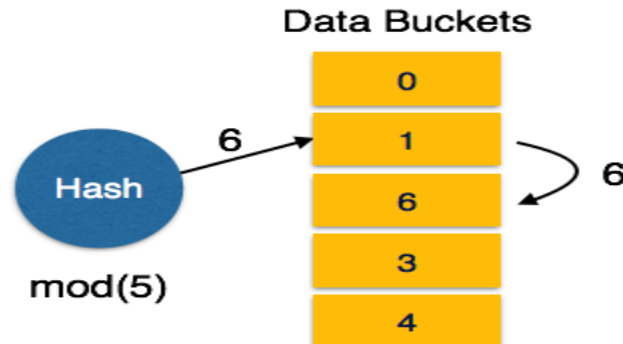
Bucket Overflow

The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- **Overflow Chaining** – When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.

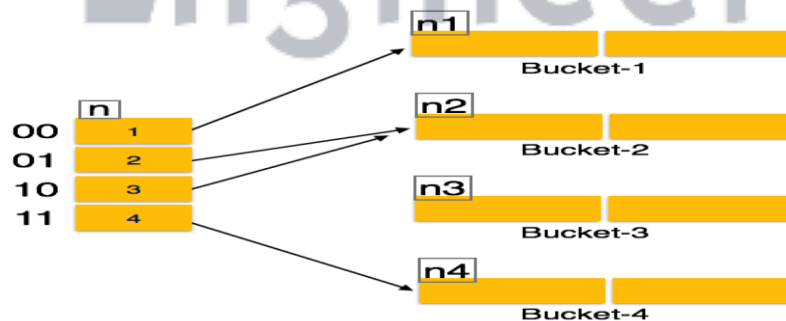


- **Linear Probing** – When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.



Dynamic Hashing : The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



Organization

The prefix of an entire hash value is taken as a hash index. Only a portion of the hash value is used for computing bucket addresses. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2^n buckets. When all these bits are consumed – that is, when all the buckets are full – then the depth value is increased linearly and twice the buckets are allocated.

Operation

- **Querying** – Look at the depth value of the hash index and use those bits to compute the bucket address.
- **Update** – Perform a query as above and update the data.

- **Deletion** – Perform a query to locate the desired data and delete the same.
- **Insertion** – Compute the address of the bucket
 - If the bucket is already full.
 - Add more buckets.
 - Add additional bits to the hash value.
 - Re-compute the hash function.
 - Else
 - Add data to the bucket,
 - If all the buckets are full, perform the remedies of 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.

6.5 Order Indices

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books. Indexing is defined based on its indexing attributes. Indexing can be of the following types

- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types –

- Dense Index
- Sparse Index

Dense Index : In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

China	→	China	Beijing	3,705,386
Canada	→	Canada	Ottawa	3,855,081
Russia	→	Russia	Moscow	6,592,735
USA	→	USA	Washington	3,718,691

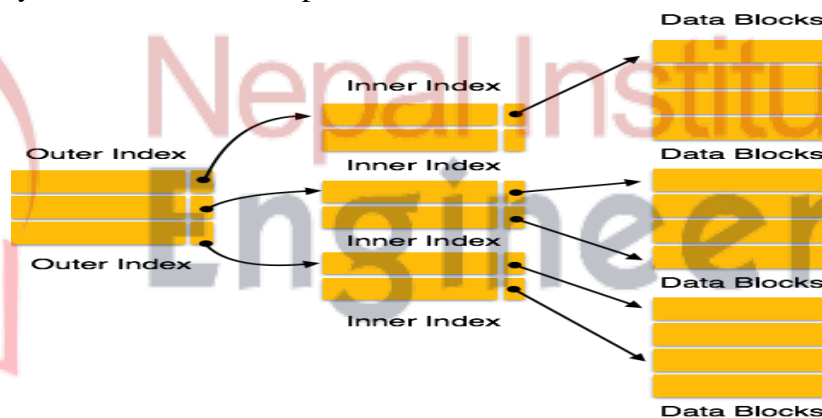
Sparse Index : In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data

we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

China	→	China	Beijing	3,705,386
Russia	→	Canada	Ottawa	3,855,081
USA	→	Russia	Moscow	6,592,735
	→	USA	Washington	3,718,691

Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

6.6 B+ Tree

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

thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B⁺ tree can support random access as well as sequential access.

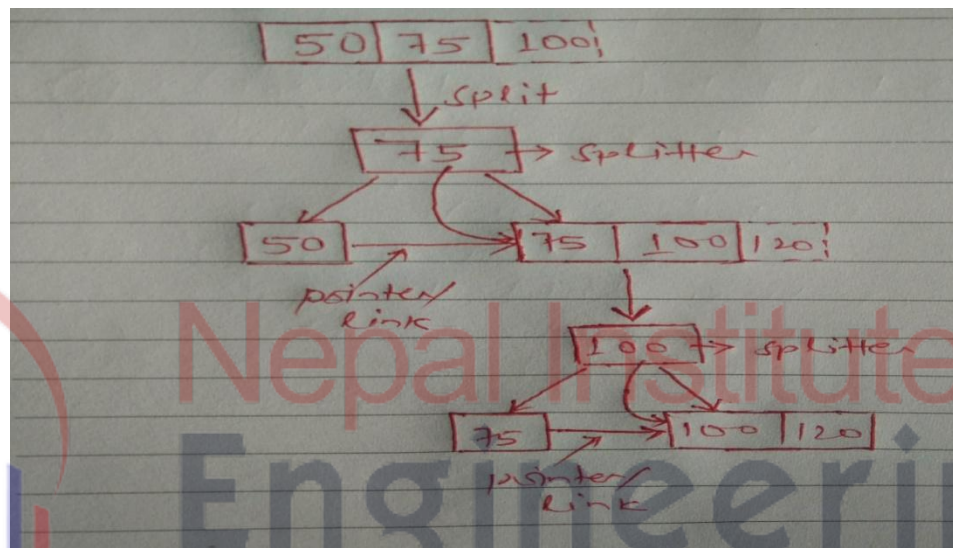
It is best suited for range queries in database. All data are stored in leaf node. Every leaf is at the same level. All leaves have pointer link with each other. Root can have 2.....n number of children.

Example : Threshold level (m)=maximum number of elements that can be placed at a single leaf node.

If we have m=2, 2 elements at the node.

Suppose we have elements : 50, 75, 100, 120,and so on

Here m = 2



Here smaller value always exist on left side. Starting with 50 and 75, 50 is on left and 75 is on right at same level due to threshold value m=2. Next 100 should be on right of 75, so here splitting occurs and now 75 and 100 will be on right and 50 single will be on left. But 50 will point to 75. Again to place 120, now 100 will be splitter and 75 comes on left of 100 and 100 and 120 will be on right where 75 will point to 100.

B⁺ Tree Insertion

- B⁺ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows –
 - Split node into two parts.
 - Partition at $i = \lfloor (m+1)/2 \rfloor$.
 - First i entries are stored in one node.
 - Rest of the entries ($i+1$ onwards) are moved to a new node.
 - i^{th} key is duplicated at the parent of the leaf.
- If a non-leaf node overflows –
 - Split node into two parts.
 - Partition the node at $i = \lfloor (m+1)/2 \rfloor$.
 - Entries up to i are kept in one node.

- Rest of the entries are moved to a new node.

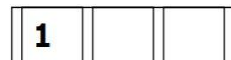
B⁺ Tree Deletion

- B⁺ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
 - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
 - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
 - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
 - Merge the node with left and right to it.

Insertion Example

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

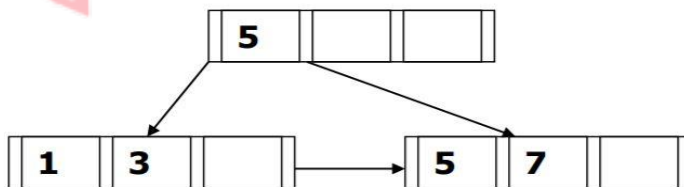
Insert 1



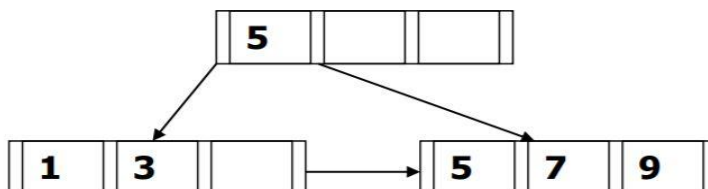
Insert 3,5



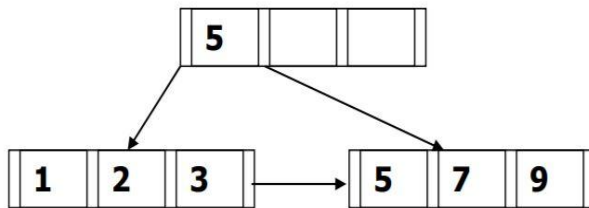
Insert 7



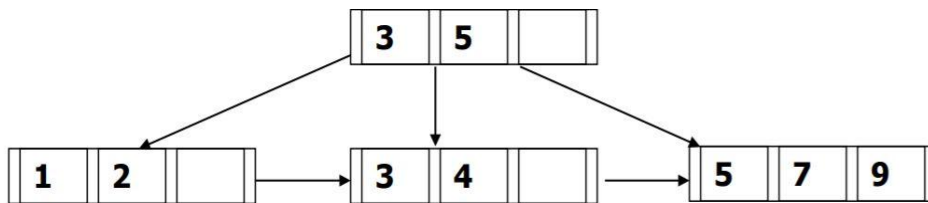
Insert 9



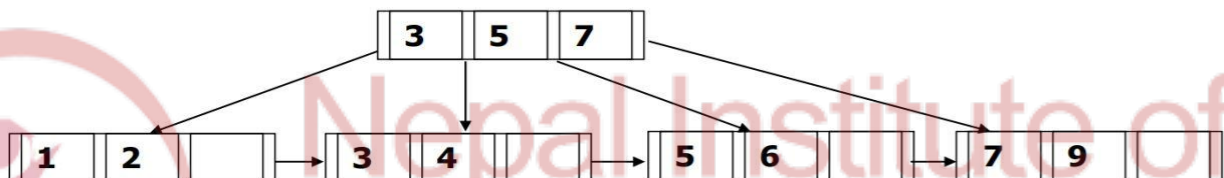
Insert 2



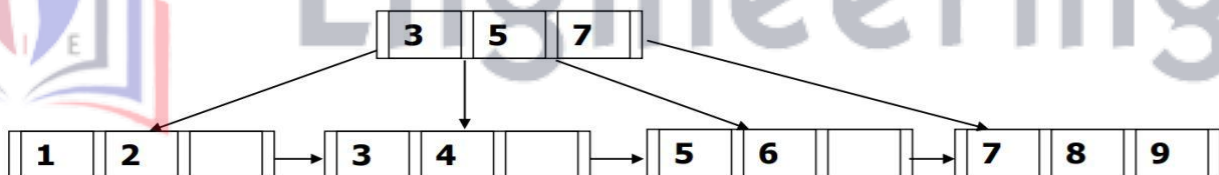
Insert 4



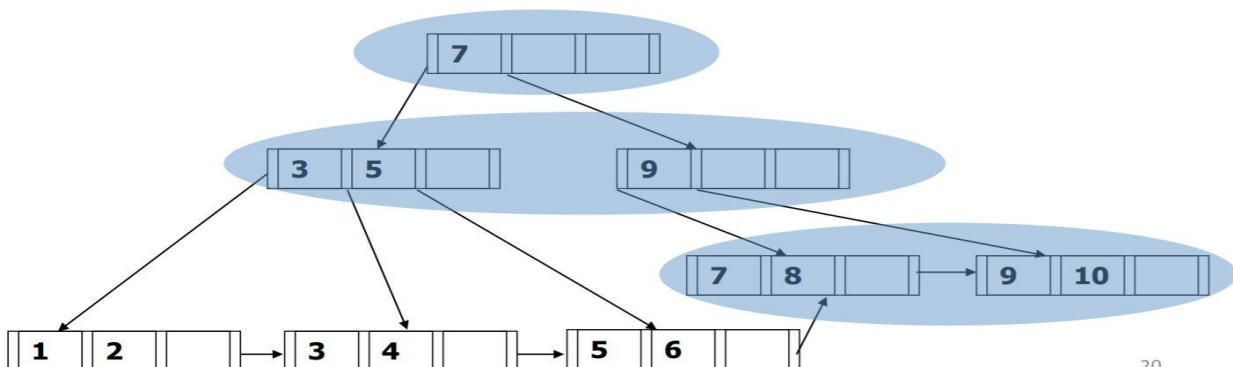
Insert 6



Insert 8



Insert 10

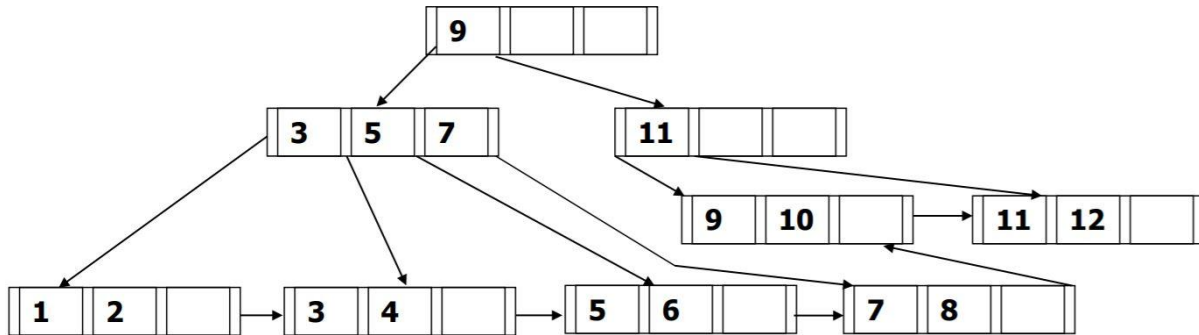


20

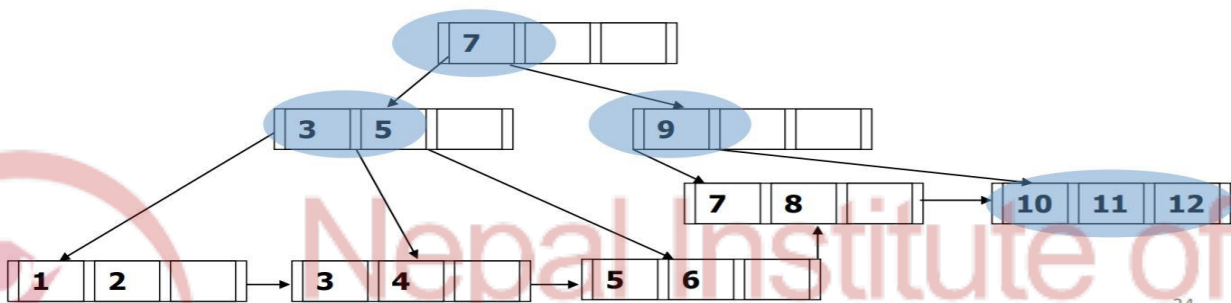
DELETION in B+ Tree

By : Er. Deepak Kumar Singh

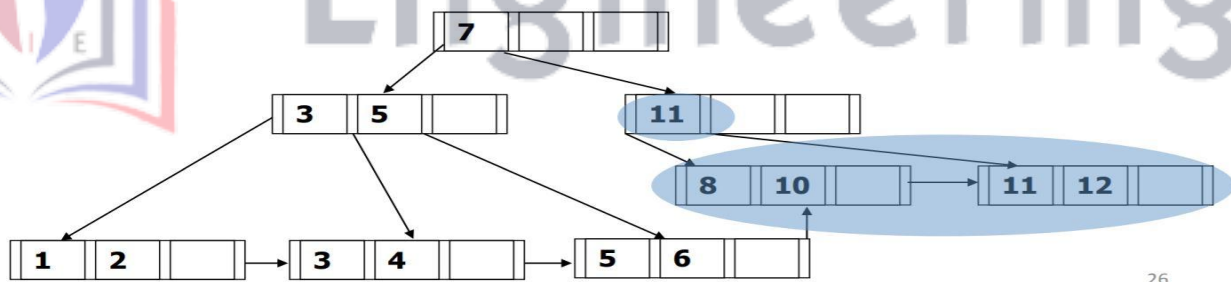
Show the tree after deletion



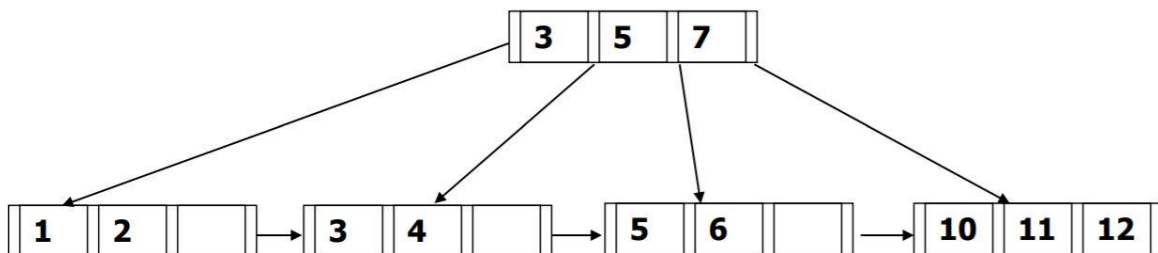
Remove 9, 7, 8
After removing 9



After removing 7



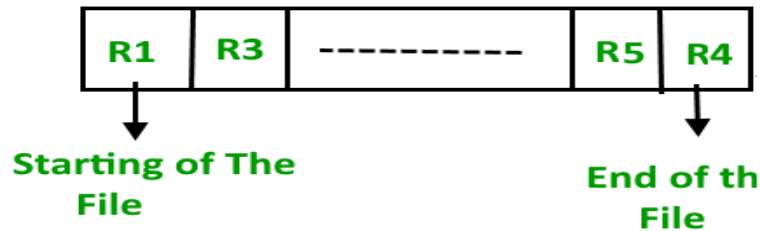
After removing 8



Sequential File Organization –

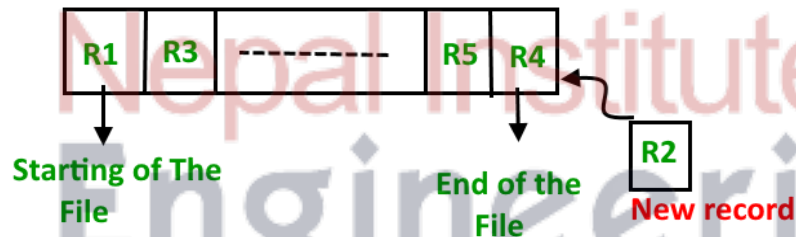
The easiest method for file Organization is Sequential method. In this method the the file are stored one after another in a sequential manner. There are two ways to implement this method:

- **File 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.

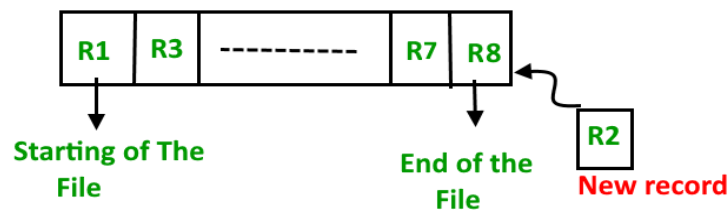


Insertion

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.

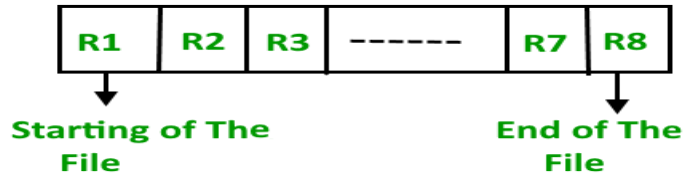


- **Sorted File Method** –In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.



Insertion

Let us assume that there is a preexisting 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.



Advantages and Disadvantages

Advantages

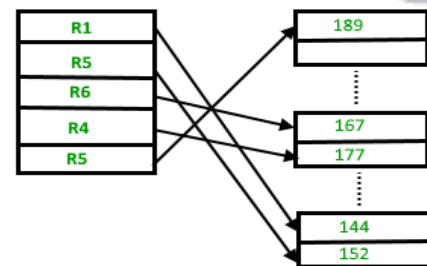
- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e. cheaper storage mechanism.

Disadvantages

- 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.

Heap File Organization

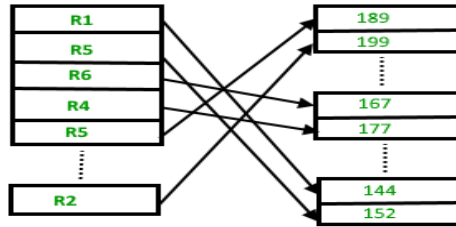
Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next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



and manage the new records.

Insertion

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, let's 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.

Advantages and Disadvantages

Advantages

- 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.

Disadvantages

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

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.

EMPLOYEE

EMP ID	EMP_NAME	EMP_ADD	DEP_ID
01	JOE	CAPE TOWN	D_101
02	ANNIE	FRANSISCO	D_103
03	PETER	CROY CITY	D_101
04	JOHN	FRANSISCO	D_102
05	LUNA	TOKYO	D_106
06	SONI	W.LAND	D_105
07	SAKACHI	TOKYO	D_104
08	MARY	NOVI	D_101

DEPARTMENT

DEP_ID	DEP_NAME
D_101	ECO
D_102	CS
D_103	JAVA
D_104	MATHS
D_105	BIO
D_106	CIVIL

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

CLUSTER KEY



DEP_ID	DEP_NAME	EMP ID	EMP_NAME	EMP_ADD
D_101	ECO	01	JOE	CAPE TOWN
		02	PETER	CROY CITY
		03	MARY	NOVI
D_102	CS	04	JOHN	FRANSISCO
D_103	JAVA	05	ANNIE	FRANSISCO
D_104	MATHS	06	SAKACHI	TOKYO
D_105	BIO	07	SONI	W.LAND
D_106	CIVIL	08	LUNA	TOKYO

DEPARTMENT + EMPLOYEE

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:

- **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.
- **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.

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.

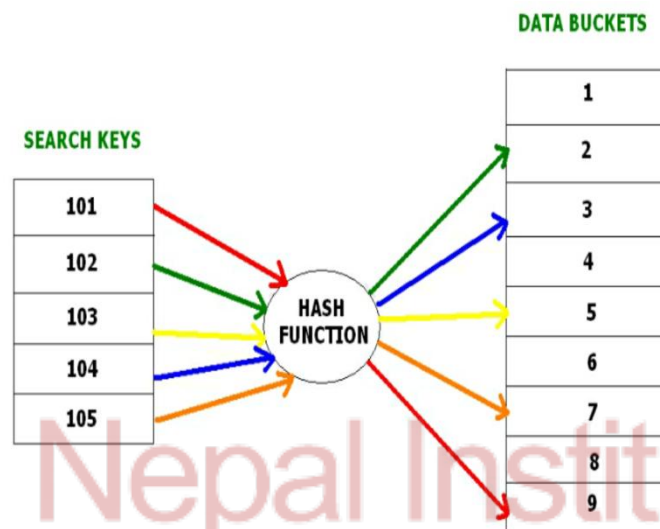
Hash File Organization :

- **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

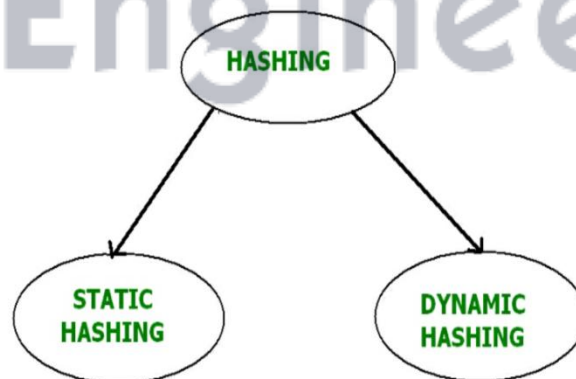
hash index – address of the data block. Hash function can be simple mathematical function to any complex mathematical function.

- **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 2^n 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 :



Static Hashing –

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if we want to generate address for STUDENT_ID = 76 using mod (5) hash function, it always result in the same bucket address 4. There will not be any changes to the bucket address here. Hence number of data buckets in the memory for this static hashing remains constant throughout.

Operations –

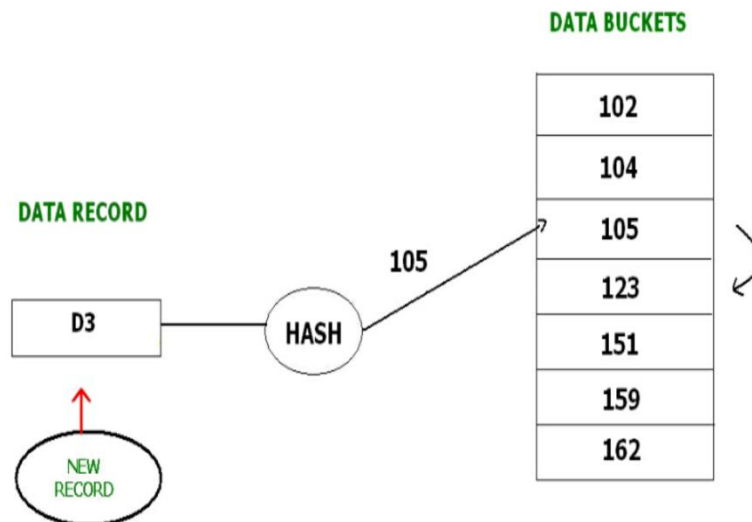
- **Insertion** – When a new record is inserted into the table, The hash function h generate a bucket address for the new record based on its hash key K .
Bucket address = $h(K)$
- **Searching** – When a record needs to be searched, The same hash function is used to retrieve the bucket address for the 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 4. Then we will directly got to address 4 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:

Open Hashing

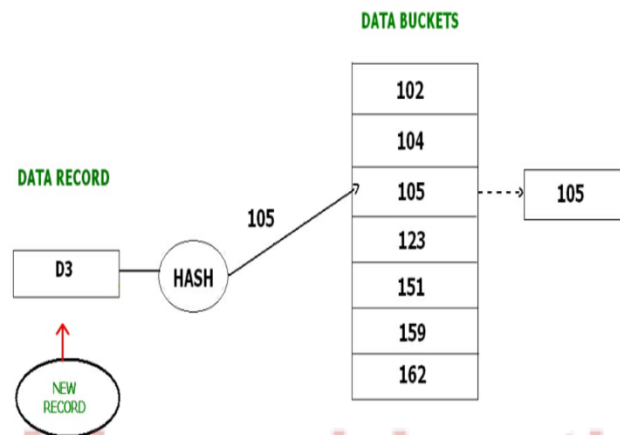
In Open hashing method, next available data block is used to enter the new record, instead of overwriting older one. This method is also called linear probing.

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.



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.



- **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**

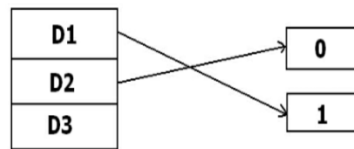
Double Hashing is another method similar to linear probing. Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function. That's why the name is double hashing.

Dynamic Hashing

The drawback of static hashing is that that it does not expand or shrink dynamically as the size of the database grows or shrinks. In Dynamic hashing, data buckets grows 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.

$h(D1) \rightarrow 1001$
 $h(D2) \rightarrow 0101$
 $h(D3) \rightarrow 1010$



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.

$h(D1) \rightarrow 1001$
 $h(D2) \rightarrow 0101$
 $h(D3) \rightarrow 1010$

