International Islamic University Chittagong (IIUC)

Department of Computer and Communication Engineering

Program: **B.sc (Eng.)**
Course Code: **CCE–3505**
Type: **Theory**
Segment: **Final Term-Segment 6**

Baizid MD Ashadzzaman
Designation: **Adjunct Lecturer**
Email: **baizid.md.ashadzzaman@gmail.com**
Phone: **8801862420119**

Course Contents:

Contents	Note

Indexing and Hashing

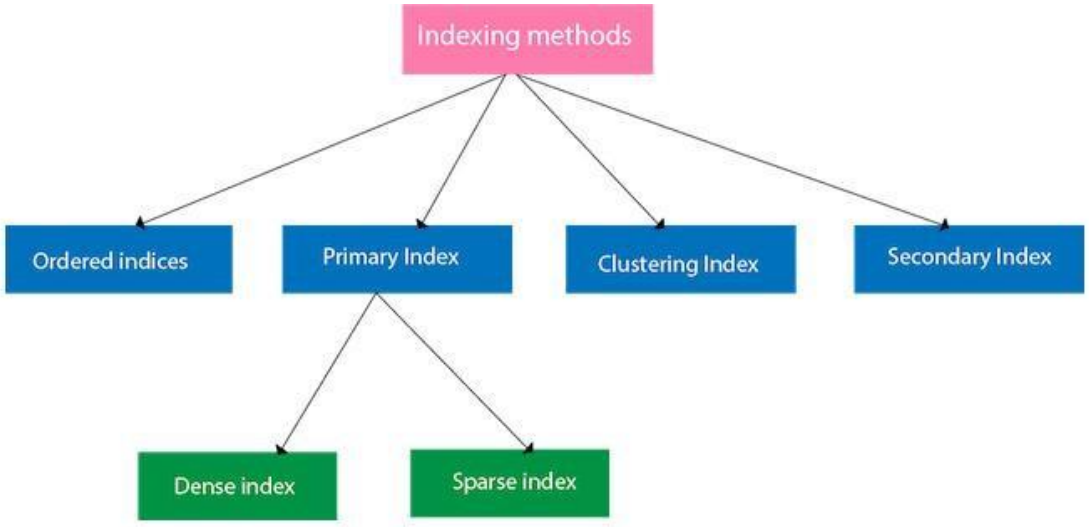
Indexing:
It is a technique that allows us to quickly retrieve records from database file.Indexing is a technique or mechanism generally used to speed up access of data. Index is basically a type of data structure that is used to locate and access data in database table quickly.

Structure of Index

Search key	**Data reference**

An index consist of a small table that has two columns.
The first column is the search key (comprises the copy of the primary key or candidate key of a table), while the second column stores the set of pointers that holds the address of the disk block (or reference to it) where that specific key value is stored. An index takes a search key as input and returns a collection of matching records.

Indexing Methods



Ordered Indices

Ordered indices are indices that have been ordered. These indices have been sorted, which makes searching easier and faster. To further understand ordered indices, let's look at an example.

Consider a table of Coding Ninjas workers with lac of records, each of which is 5 bytes. If the employee ID 1024 is among those lac of records

In the event of a database without an index, we must search the disc block starting at zero and continuing until it reaches 1024. The DBMS will read the record once 1024*5=5120 bytes have been read.

If there is an index, we will conduct the search using the index, and the DBMS will read the record after reading 1024*2 = 2048 bytes, which is a lot fewer bytes than in the previous example.

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.

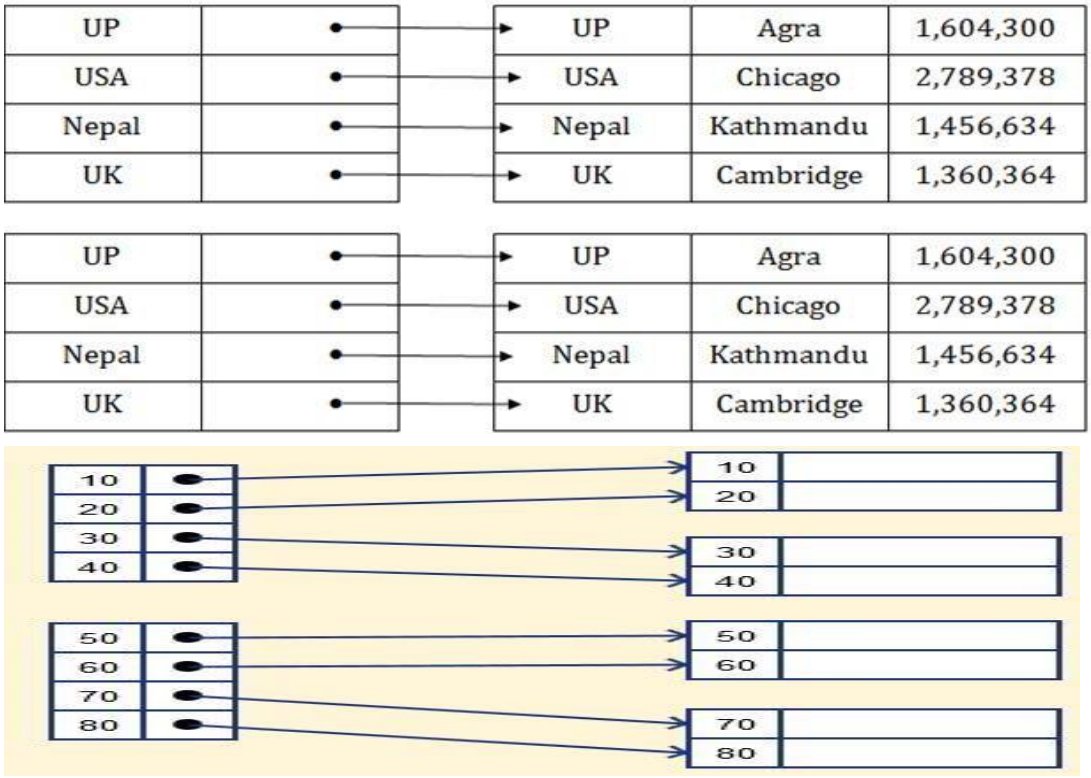
Types of Primary Index

The primary index can be classified into two types:

- Dense index
- Sparse index.

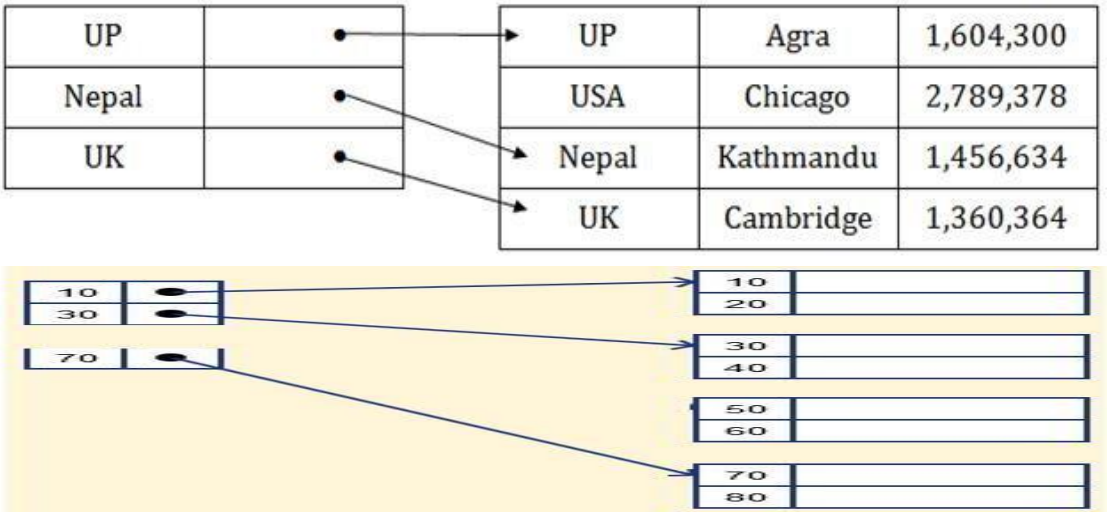
Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.
- In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records. In this Indexing, method records contain search key value and points to the real record on the disk.



Sparse index

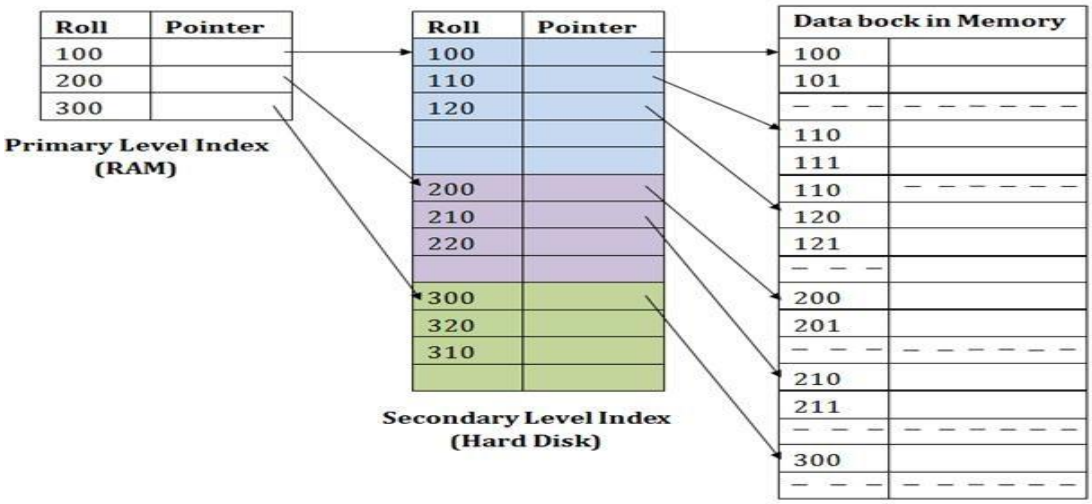
- In the data file, index record appears only for a few items. Each item points to a block.
- In this indexing, instead of pointing to each record in the main table, the index points to the records in the main table in a gap
- In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched. o However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.



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, 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)

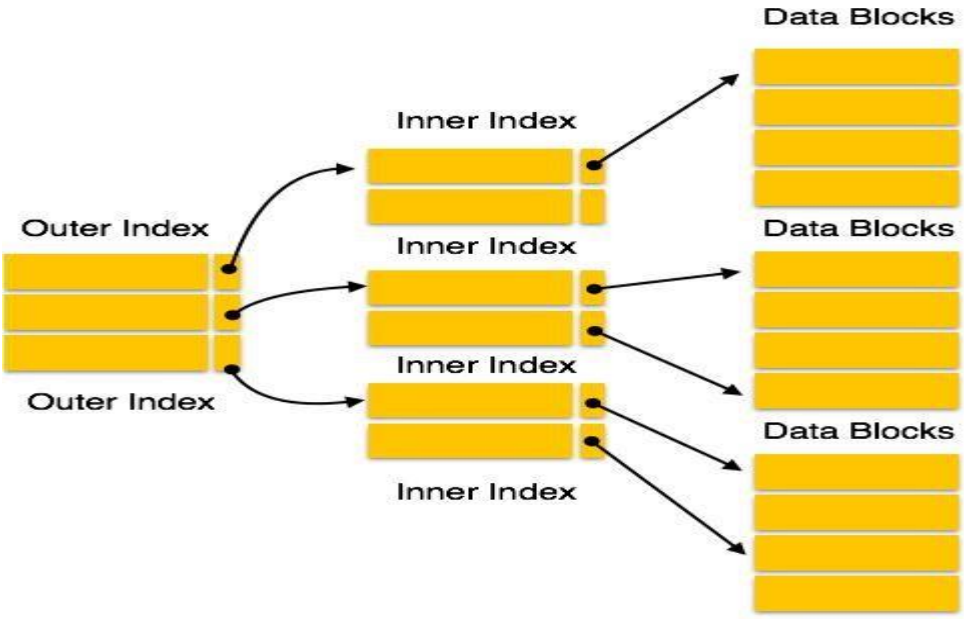


Multilevel Indexing:

The multilevel indexing segregates the main block into various smaller blocks so that the same data can be stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.

- Multilevel indexing is a technique used in database systems to efficiently index large amounts of data.

- It involves creating multiple levels of indexes to navigate and access the data.
- The first level index, also known as the primary index, provides an entry for each block or page of data.
- The primary index is usually based on the primary key of the table.
- Each entry in the primary index points to a block or page containing a secondary index.
- The secondary index is created on a secondary key or non-key attribute of the table.
- The secondary index provides a way to locate specific records within a block or page.

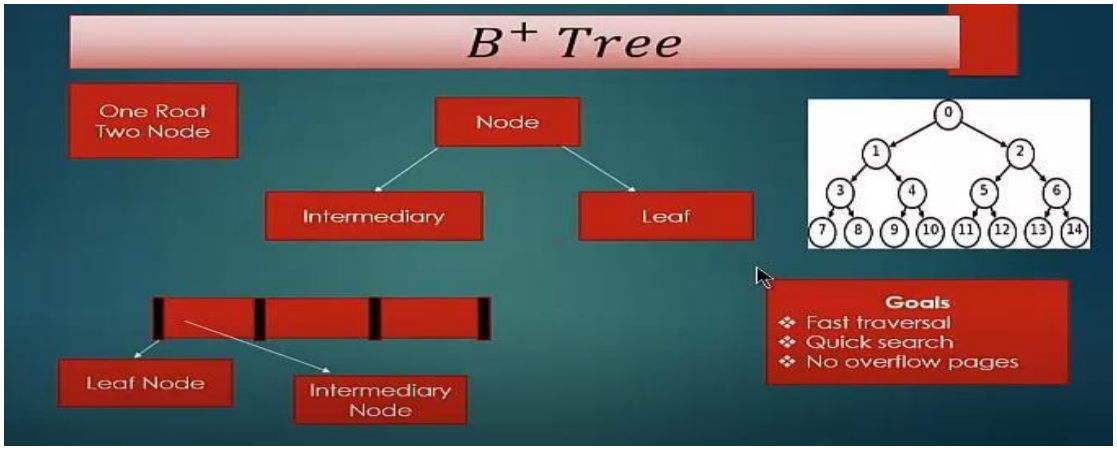


B+ tree index files

- 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.
- 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

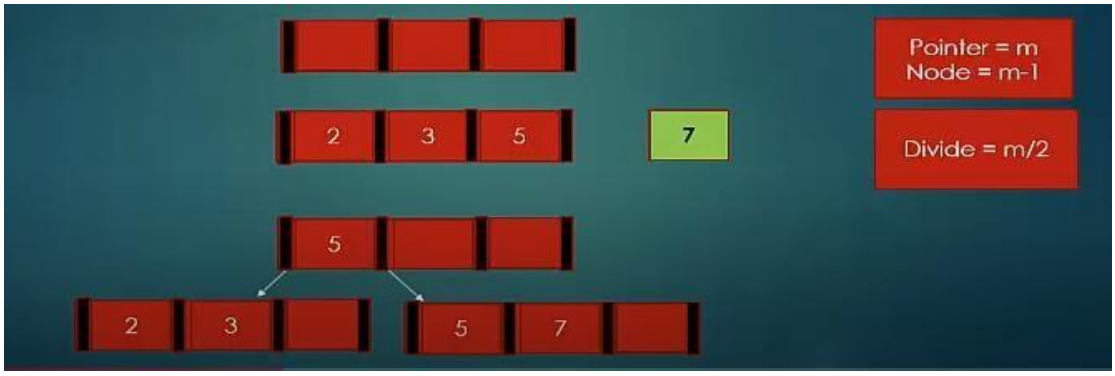
o In the B+ tree, every leaf node is at equal distance from the root node. o It contains an internal node and leaf node.



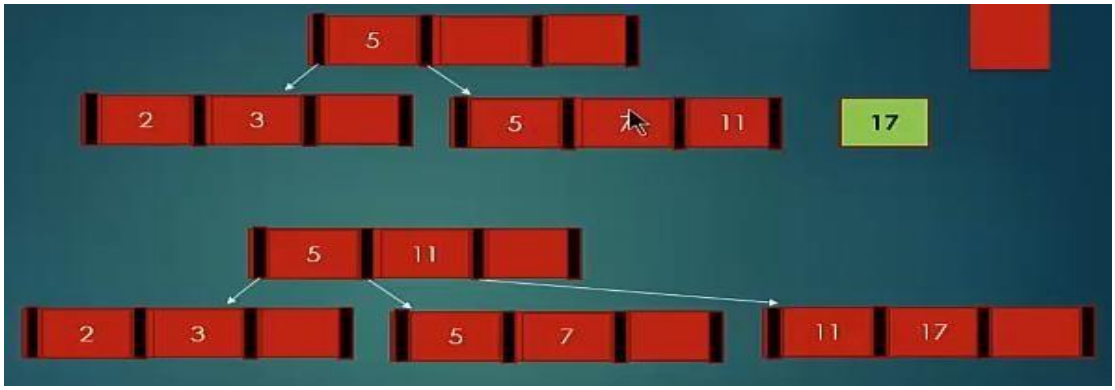
Example

Construct a B+-tree for the following set of values
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)
Assume that the tree is initially empty and values are inserted in ascending order.
1) Construct B+-trees for the cases where the number *m* of pointers that will fit a node is as follows:
a. Four

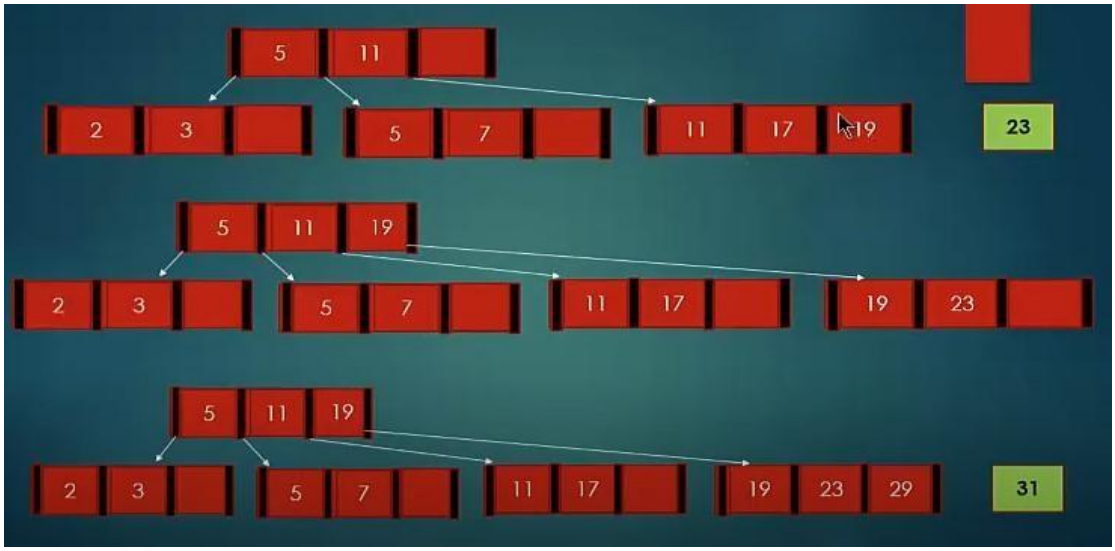
Step:1



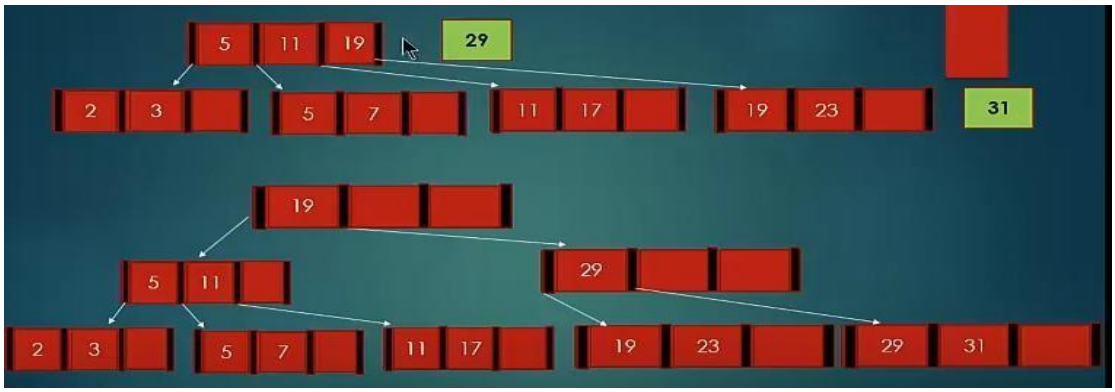
Step:2



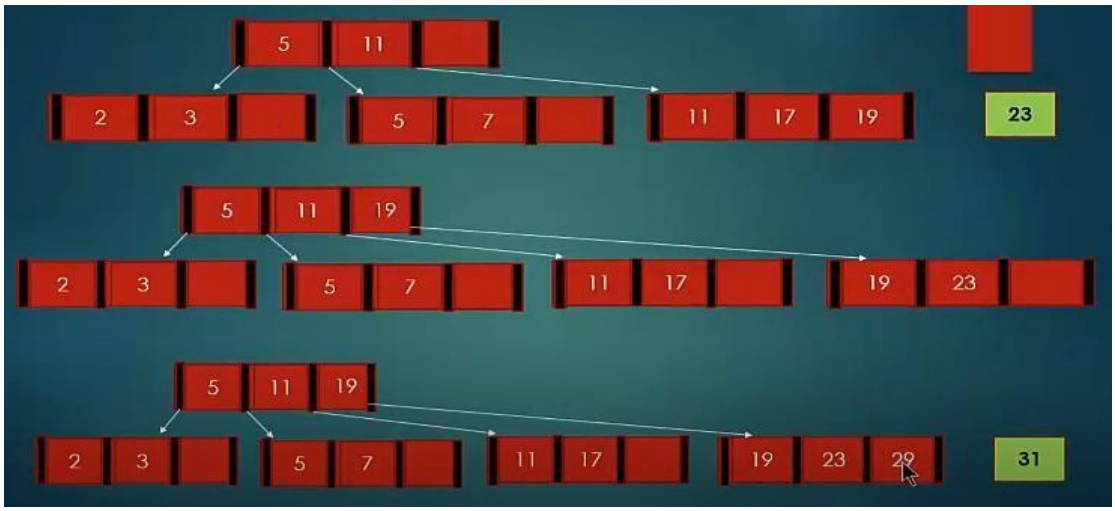
Step:3



Step:4



Step:5



See for insertion: https://www.youtube.com/watch?v=2hre_Jd1i_Y

See for Deletion: https://www.youtube.com/watch?v=-2cu6GteyWI&t=323s **Hashing**
Hashing in DBMS is a technique used to directly search the location of data without using index structure. It uses a hash function (a mathematical function) to find the exact location of a record in the minimum amount of time.

Important Terminologies

Data Bucket

Data buckets are the memory locations where the actual data records are stored.

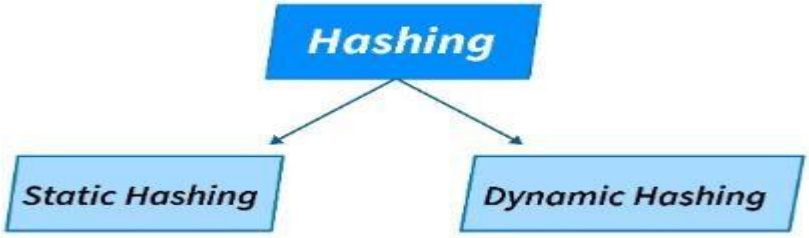
Hash function

A hash function is a mathematical function that returns the value of the address of the desired data block. Mostly, it uses the primary key to generate the address of a data block.

Hash Index

It denotes the address of a data block generated by the hash function.
Types of Hashing in DBMS

Hashing techniques can be divided into two types, hashing. static hashing and dynamic



Static Hashing

Whenever a search-key value is given in static hashing, the hash algorithm always returns the same address. If the mod-4 hash function is employed, for example, only 5 values will be generated. For this function, the output address must always be the same.

Static hashing can also be further divided to open and closed hashing.

A. Open Hashing

In open hashing, whenever a collision occurs, we probe for the next empty bucket to enter the new record. The order of checking are majorly of two types -

- Linear Probing
- Quadratic Probing

Linear Probing

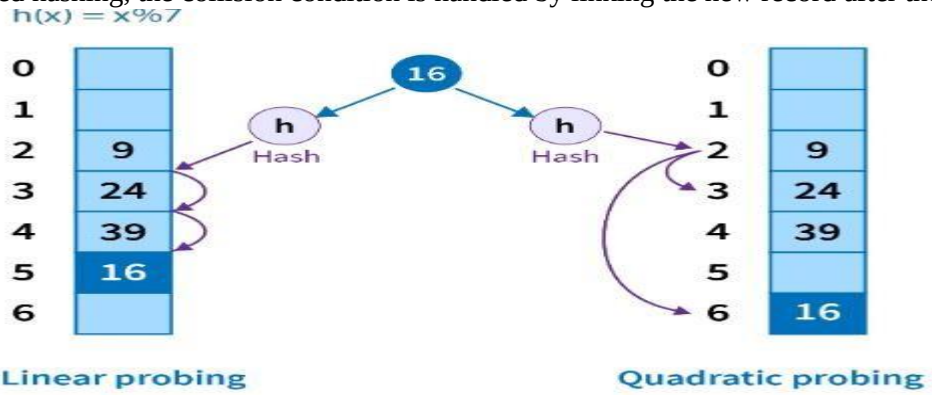
It is used when a record already exists on the memory location calculated by Hash Function. It linearly searches for an empty bucket until it is found.

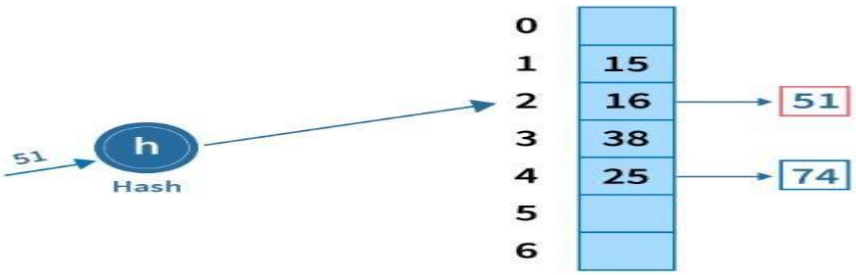
Quadratic Probing

It is similar to linear probing with the only difference as it uses an arbitrary polynomial function to search for the next empty bucket instead of searching linearly

B. Closed Hashing

In closed hashing, the collision condition is handled by linking the new record after the previous one, due to which is also termed as "Hashing with separate chaining". For example -





Dynamic hashing

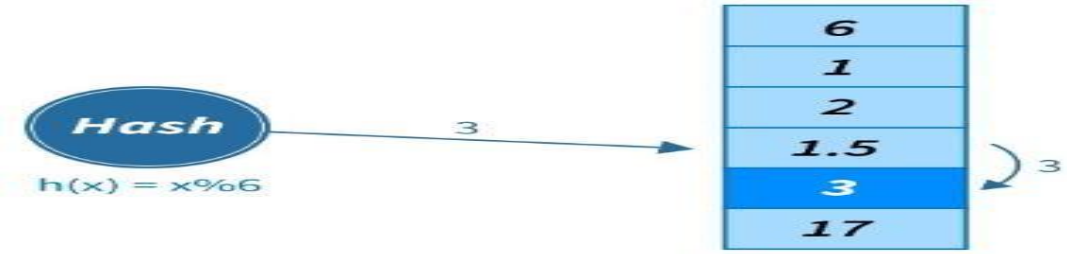
Dynamic hashing is a mechanism for dynamically adding and removing data buckets on demand. The hash function aids in the creation of a huge number of values in this hashing.

Bucket

A bucket is a unit of storage that contain one or more records. Data buckets are the memory locations where the actual data records are stored.In a hash file organization, we obtain the bucket of a record directly from its search-key value using a hash function.

Bucket Overflow

When the hash function generate the memory address that already contains some data record is known as bucket overflow.



Reason of Bucket Overflows

Bucket overflow can occur because of

1. Insufficient buckets
2. Unequal distribution of the records.

This can occur due to two reasons:

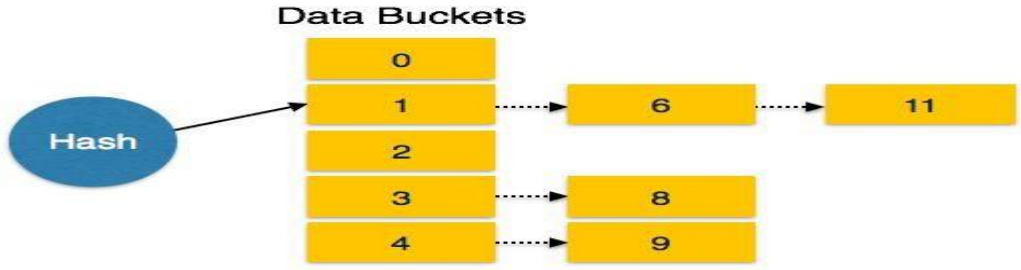
1. multiple records have same search-key value.
2. Chosen hash function produces non-uniform distribution of key value

Handling of bucket overflows

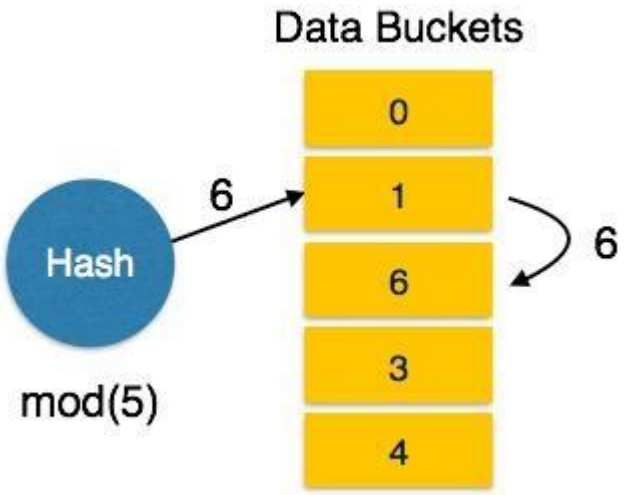
Open hashing − Open hashing occurs where records are stored in different buckets. Compute the hash function and search the corresponding bucket to find a record.

Closed hashing − Closed hashing occurs where all records are stored in **one** bucket. Hash function computes addresses within that bucket. It is not used much in database applications.

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**.


Data Buckets
0
1
6
3
4
Hash
6
6
mod(5)