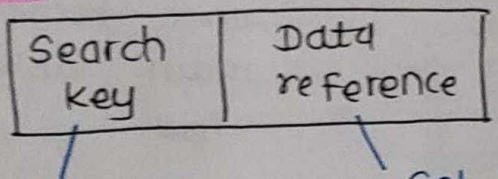
Indexing and Hashing

Indexing :

- used to optimize the DB performance by minimizing no. of disk access required while query processing
- Index is type of data-structure
- used to locate & access data in DB table quickly.

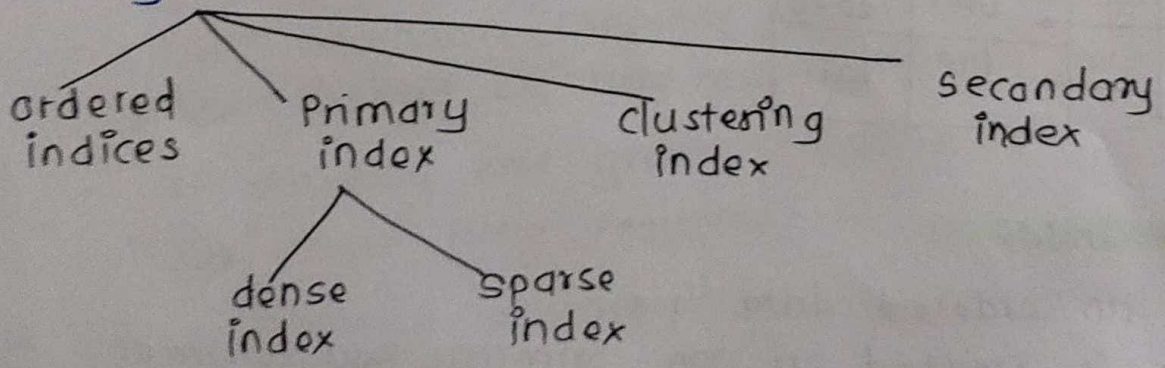
Index structure :



- Copy of primary key or candidate key
- In sorted form

Set of pointers holding the address of the disk block where value of particular key can be found.

Indexing methods :



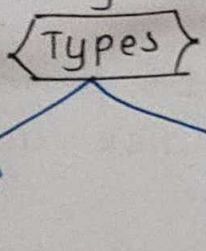
① Ordered Indices :

- are sorted indices
- which makes searching faster

② Primary index :

- Index is created on the basis of primary key of the table

- As primary keys are unique and stored in sorted order the searching will be faster.



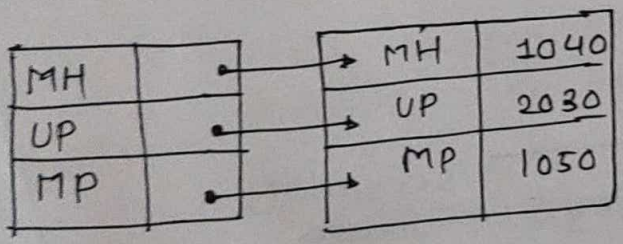
I] Contains an index record for every search key value in the data file.

II] Thus searching is faster.

III] no. of record in index table = no. of record in main table.

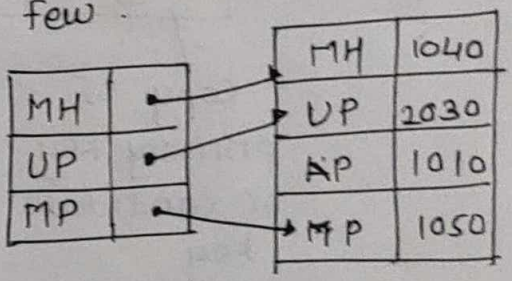
IV] Needs more spare for storing record itself.

V] Index records have search key & a pointer to actual record.



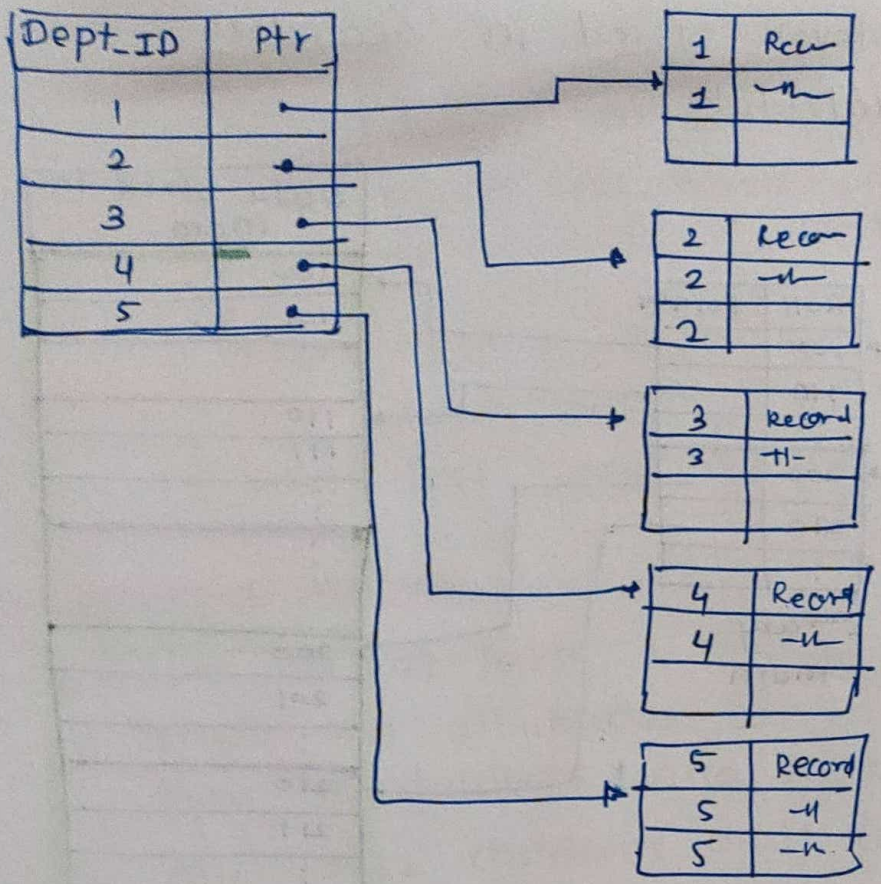
I] In data file, index record appears only for a few items.

II] Instead of pointing to each record in the main table, points to few.



3) Clustering Index :

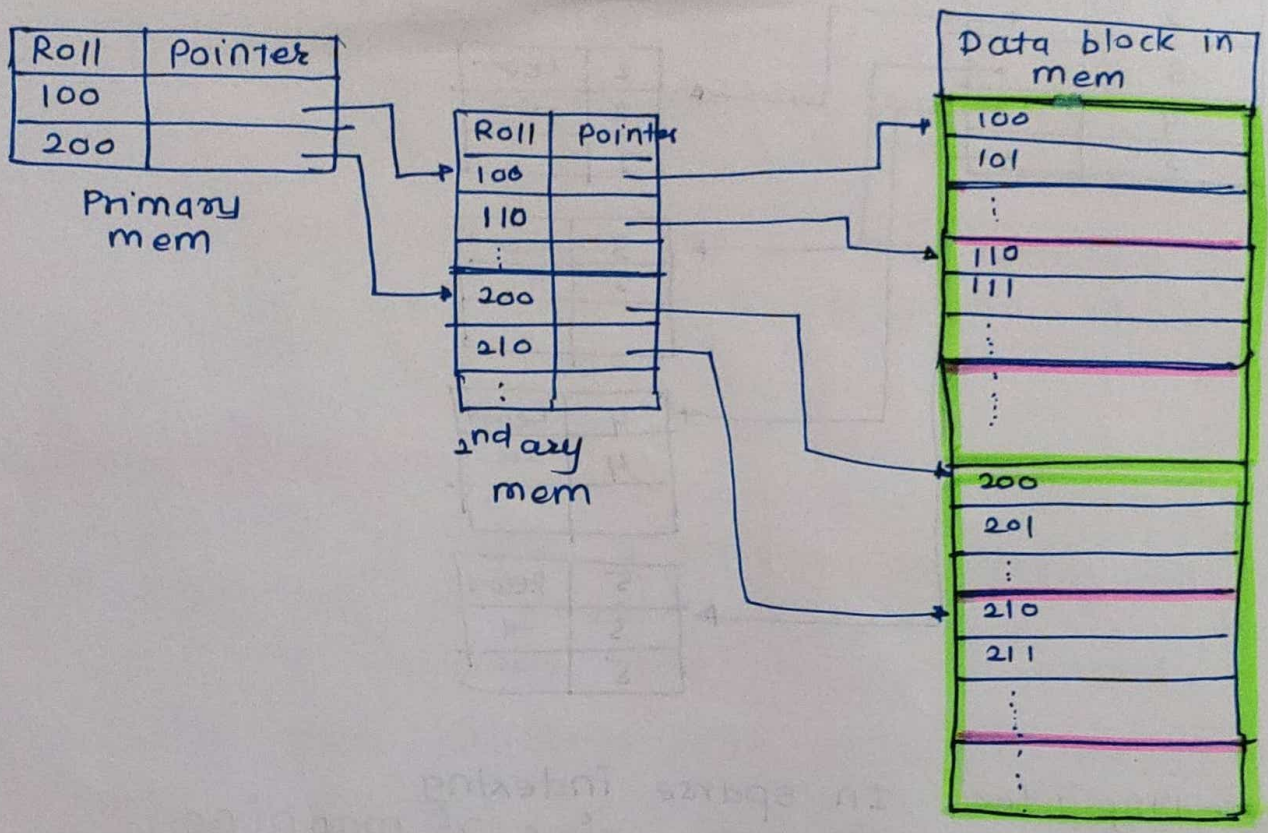
- It is an ordered data file
- Index is created on non-primary key columns which may not be unique.
- To identify record faster, we will group two or more columns to get unique value & create index from them.



4) Secondary index : In sparse indexing
- Size of table grows size of mapping also grows
- These mappings are kept in primary memory so that address fetch should be faster.
- Then secondary memory searches actual data based on address got from mapping
- If mapping size grows fetching time increases. (ie more time requires)

To overcome these prblms
- In this method
  Large no of columns are selected initially so that mapping size of first level becomes small.
- Then each range is further divided into small ranges.
- mapping of first level stored in primary memory.

- mapping of 2nd level stored in secondary memory with actual data.



B + Tree :

- B+ tree is a balanced binary search-tree
- follows multilevel index format
  - leaf nodes = actual data pointers
  - B+ ensure that all leaf nodes remain at same height.
  - leaf nodes are linked using linkedlist.
  - B+ support random and sequential access.

Structure of B+ Tree :

- every leaf node is at equal distance from root node
- It contains internal node & leaf node.

Internal node —— internal node can contain at least n/2 record pointers except root node

—— At most internal node of tree contains n pointers.

leaf node — contain at least $n/2$ record pointers & $n/2$ key values

- At most n records and n key pointers values.
- Every leaf node of B+ tree contains one block pointer P to point to next leaf node.

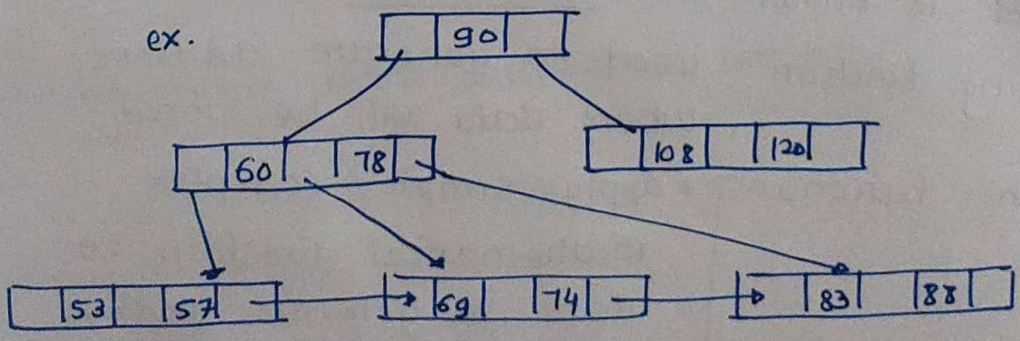
Properties of B+ tree :

- There are at least 2 children for root
- except root all nodes can have max m childrens and max 'm-1' keys
- min $m/2$ childrens and $(m/2 - 1)$ keys

In B+ tree

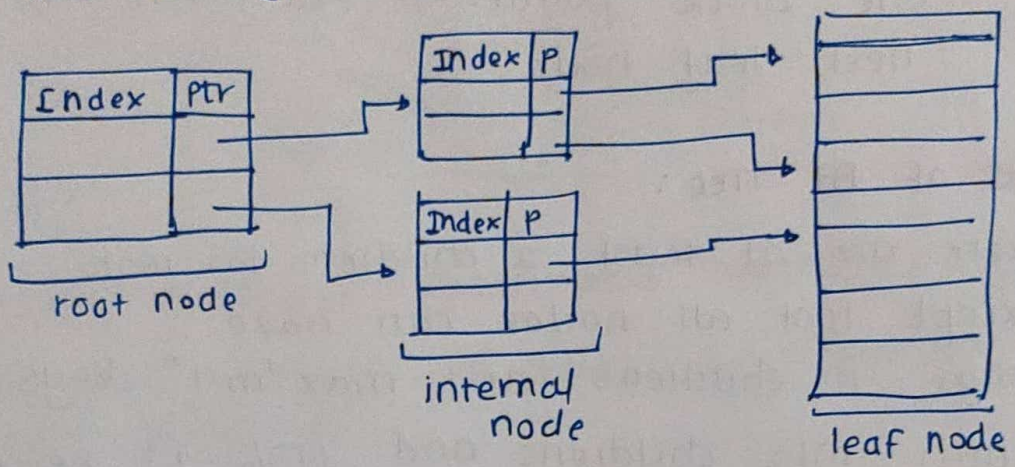
- key value are placed in internal node
- Record and data placed in leaf node.
- leaf nodes are connected as singly linked list for fast accessment
- internal nodes → In main memory
  Leaf nodes → In secondary memory

ex.



* Multilevel Indexing :

- refers to a hierarchical structure of Indexes.
- Each level of index provides a more detailed reference to the data.
- allows faster data retrieval.
- B+ tree is type of multilevel Indexing



Hashing

- In huge DB it is very difficult to / inefficient to search all index values and reach to desired data
- So, Hashing is the technique used to calculate the direct location of data record on the disk without using index structure
- memory location where these records are stored is known as data bucket.
- Hashing function :- used to generate address where data will be stored.
- Hash function
  - apply simple / complex mathematical function on data to generate address
  - Any column value can be choose to generate address.
  - most of the time primary key

→ Types of Hashing :

① Static Hashing

② Dynamic Hashing

① Static Hashing

- Resultant data bucket address is always same.
- Ex.

Hash function is mod 5

suppose, stud_ID = 103

∴ 103 % 5 = (3) —— always same.

Here, no. of data bucket in memory are constant i.e 1-5 —— (for mod 5)

▶ operation of static hashing

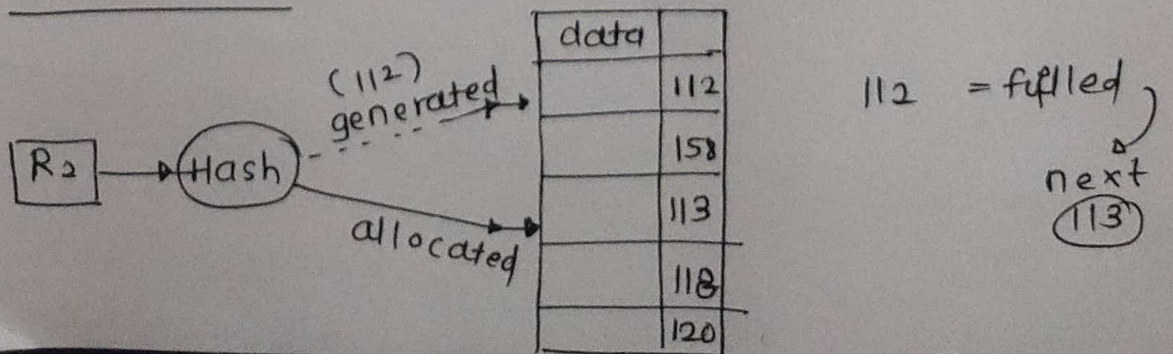
- searching
- Insert
- Delete
- Update

If we want to insert new record, but address generated by hash function is not empty or data already exists there, this situation is bucket overflow.

To overcome,

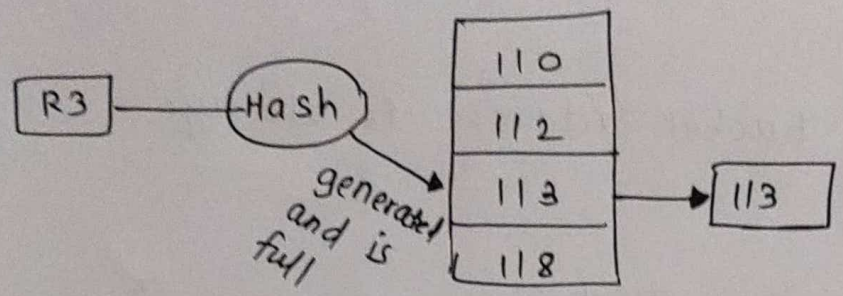
① Open hashing (linear probing) :

- when hash function generates an address at which data is alreddy stored, then next bucket will be allocated.



② close hashing (overflow chaining)

- When buckets are full, ~~same~~ then new data bucket is allocated for the same hash result and is linked after previous one.



② Dynamic Hashing (Extendable hashing):

- Overcome problem of bucket overflow of static hashing
- Here data buckets grows & shrinks as per records increases / decreases.
- Insertion and deletion without resulting in poor performance.

Adv
- performance doesn't decrease with data growth
- memory is well utilized
- good for dynamic DB.

Disadv
- maintaining data bucket is complex