UNIT- V

Data on External Storage

File Organizations and Indexing

Clustered Indexes

Primary and Secondary Indexes

Index Data Structures

Hash based Indexing

Tree based Indexing

Comparision of file organizations

Indexes and Performance tuning

Intuition for tree Indexes

Indexed sequential Access Method (ISAM)

B^+ trees : A dynamic Index structure.

Data on External Storage:-

A DBMS stores vast quantities of data. Therefore data is stored on external storage devices like disks and tapes, and fetched into main memory as needed for processing. The unit of information read from or written to disk is a page. The size of a page is 4KB or 8KB.

* Disks are the most important external storage devices. They allow us to retrieve any page at a fixed cost per page. If we read several pages then they are stored physically, the cost can be much less than the cost of reading same pages in random order.
* Tapes are sequential access devices and force us to read data one page after other. They are mostly used to archive data that is not used on a regular basis.
* Each record in a file has a unique identifier called record id, or rid for short. We can find the disk address of the page containing the record by using rid.

Data is read into memory for processing and written to disk for persistent storage by a layer of software called buffer manager. Space on disk is managed by disk space manager.

* Disks:- It is a permanent storage device. Data is non-volatile i.e when there is a power loss data will be there in the disk. It is secondary storage device.
* Tapes:- These are sequential access devices and force us to read data one page after other.

File Organizations and Indexing:-

A file of records is an important abstraction in DBMS. A file can be created, destroyed and have records inserted to and delete from it. A file organization can be defined as a process of arranging records in a file when the file is stored in disk. A file layer stores the records in a file in a collection of disk pages.

Index:-

An index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations An index allows us to efficiently retrieve all records that satisfies search conditions on the search key fields of index. We use the term data entry to refer to the records stored in index file. A data entry with search key value k denoted as k* contains enough information to locate data records with search key value k.

There are 3 main alternatives for what to store as a data entry in a index.

1. A data entry k* is an actual data record (with search key value k)
2. A data entry is a <k, rid> pair, where rid is the record id of a data record with search key value k.
3. A data entry is a <k, rid-list> pair where rid-list is a list of record id's of the data records with search key value k.

Alternative (1), each entry k* is a data record with search key value k.

Alternatives (2) and (3), which contain data entries that points to data records, are independent of file organizations. that is used for indexed file.

Alternative (3) offers better space utilization than alternative (2)

Clustered Index:-

When a file is organized so that the ordering of data records is same as ordering of data entries in some index we say that index is clustered. An index that uses alternative (1) is clustered index.

In practice, files are kept stored since this is too expensive to main when the data is updated so, the clustered index uses alternative (1). We sometimes refer to an index using alternative (1) as a clustered file.

Primary and Secondary Indexes:-

An index on a set of fields that includes the primary key is called as primary index, other indexes are called secondary indexes.

An index that uses Alternative (1) is called primary index and one that uses alternative (2) & (3) is called Secondary index.

Primary index cannot contain duplicates where as secondary index contains duplicates.

Index Data Structures:-

Indexing is used to optimize the performance of database by minimizing the number of disk accesses required when a query is processed. The index is a type of data structure. Two techniques are used for organizing the data.

① Hash-based indexing
② Tree-based indexing.

An index
- Takes search key as i/p.
- efficiently returns a collection of matching records.

Hash-based Indexing:-

We can organize records using a technique called hashing to quickly find records that have a given search key value.

Eg:- If the file of employee records is hashed on the name field we can retrieve all records with the name Joe.

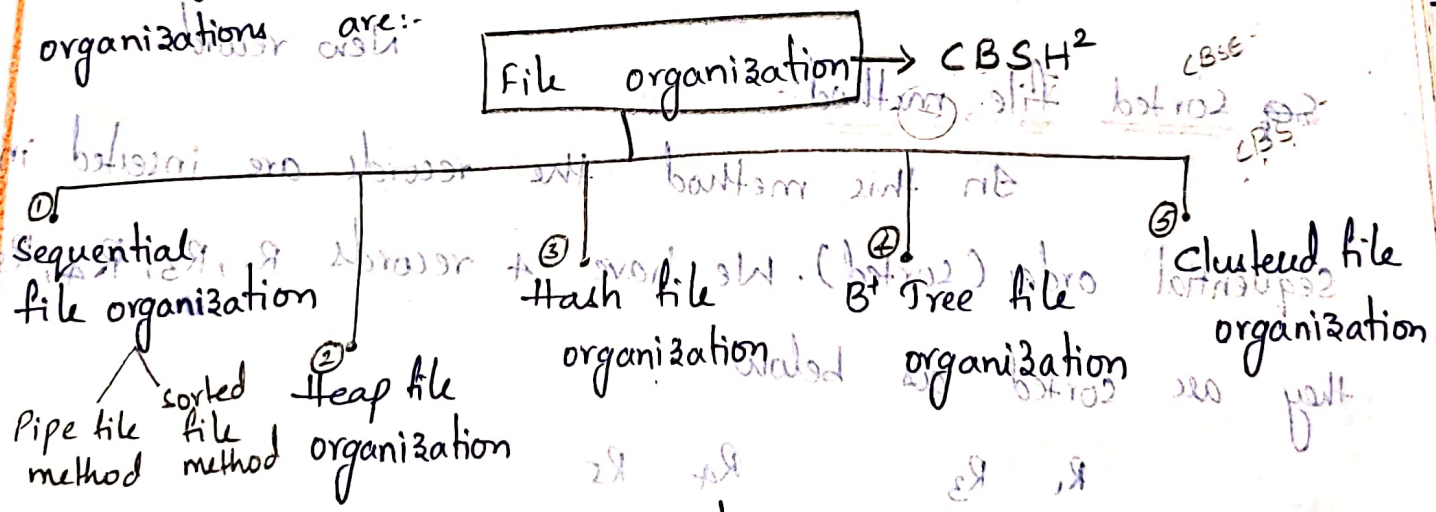
In this approach the records in a file are grouped in buckets, where a bucket consists of primary page and possibly additional pages linked in a chain. The bucket to which a record belongs to can be determined by applying special function called hash function.

* File Organization and Indexing :-

File organization is a logical relationship among various records. This method defines how file records are mapped into disk blocks. File organization is used to describe the records that are stored in terms of blocks in storage medium.

File is collection of related information that is recorded on secondary storage i.e magnetic disks, magnetic tables, optical disks.

various methods have been introduced to organize files. It depends on the programmer to decide which file organization method is required. Some types of file organizations are:-



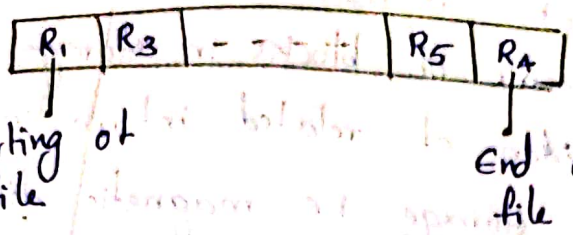
* Sequential file Method Organization :-

The easiest method for file organization is Sequential method. In this method the files are stored one after the another in sequential manner. There are 2 ways to implement this

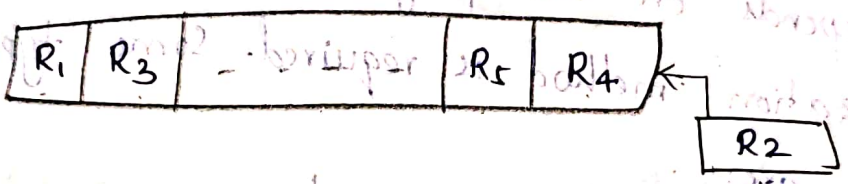
1. Pipe file method
2. Sorted " "

Pipe file method:-

In this method the records are stored in a sequence i.e one after other in the order in which they are inserted into tables.

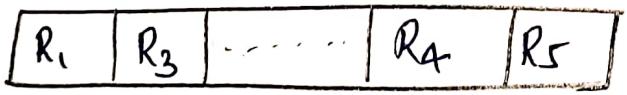


Insert :- We have 4 records in the sequence. Suppose i want to insert a new record R2 then it is simply placed at the end of file.



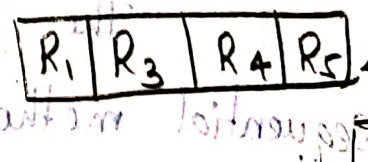
Sorted file method:-

In this method the records are inserted in sequential order (sorted). We have 4 records R_1, R_3, R_4, R_5 they are sorted as below.



Insert:- If I want to insert new record R_2 then

① It is inserted at end of the file



② Later it will be sorted.

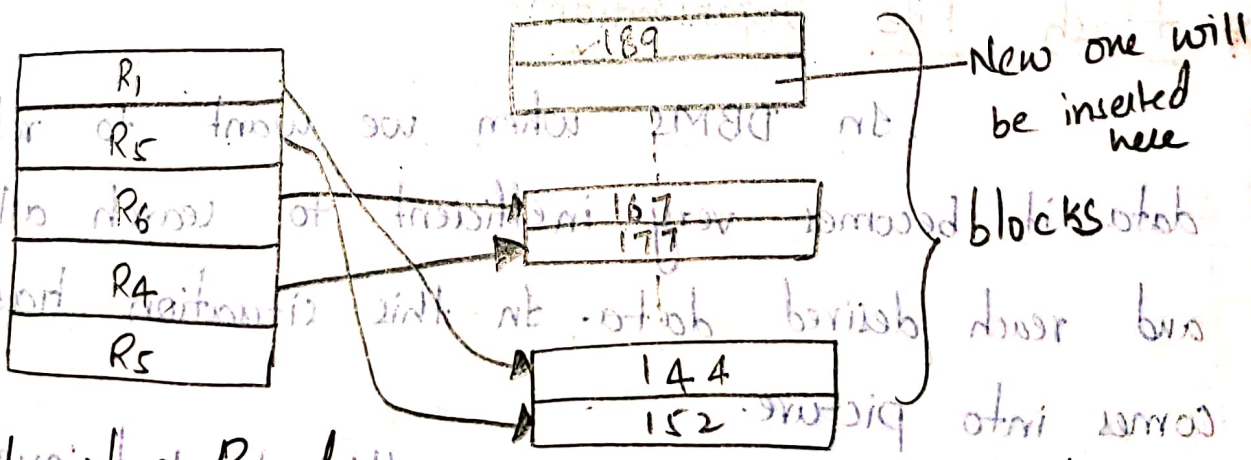
Pros and cons of sequential file organization:-

Pros:-
→ Fast and efficient method for huge amounts of data
→ simple design.
→ Files can be easily sorted
→ It is cheaper cost.

Cons:-
→ Time wastage
→ Takes more space and time

* Heap File Organization:-

Heap file organization works with data blocks. In this method records are inserted into data blocks. No sorting or ordering is required in this method. If a data block is full the new record is sorted in some other block. It is responsibility of DBMS to store and manage new records.



Insertion of new Record:-
We have 4 records R_1, R_5, R_6, R_4 and a new record R_2 has to be inserted in heap then it will be inserted where the block will be empty.

If we want to search, delete or update data in heap file organization then we will transverse the data from beginning of file till we get requested record. If the data base is huge searching, deleting or updating the record will take more time.

Pros and cons of Heap file organization:-

Pros:-

→ Fetching and retrieving is faster than sequential record but only in small databases.

→ When there is a huge number of data it need to be loaded into database at a time then this method of organization is best.

Cons:-

→ Problem of unused memory blocks

→ Inefficient for larger databases.

Hash File Organization:-

In DBMS when we want to retrieve a partic data it becomes very inefficient to search all indexes valu and reach desired data. In this situation hashing techniqu comes into picture.

Hashing is an Efficient technique to direct search the location of desired data on the disk witho using index structure. Data is stored at data blocks who

address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket.

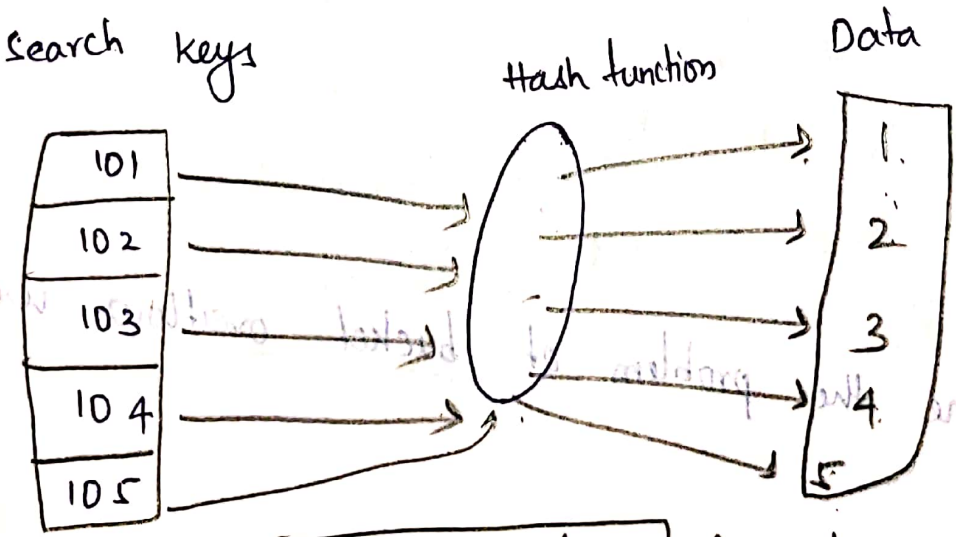
* Important terminologies in Hash file organization:-

Data Bucket:-

Data buckets are the memory locations where the records are sorted. These buckets are also concerned as Unit of Storage.

Hash function:- It is a mapping function that maps all the set of search keys to actual record address. Generally, hash function uses primary key to generate hash index. -address of data block. Hash function can be simple mathem-atical function to complex mathematical function.

Hash Index:- Pretix of an entire hash value. The pretix of entire has value is taken as hash index. Every hash index signifies some bits. These bits can address 2n buckets



* Search key mod 100 → formula

101 mod 100 = 1

Hashing

Static hashing

→ Hash function always computes same address

eg:- (98 mod 5) = 3

→ No. of data buckets in the memory remains constant.

Problem:- bucket overflow.

↓

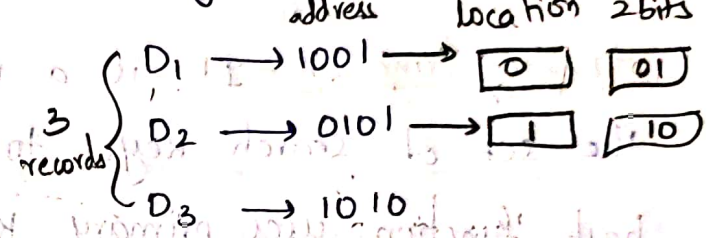
98 mod 5 = 3 → already occupied

93 mod 5 = 3 → again when it would be stored

?

Dynamic hashing

→ Data buckets grows/shrinks dynamically as the database size grows or shrinks.



Operations:-

Insert

update

Delete

Search

* To overcome the problem of bucket overflow we have

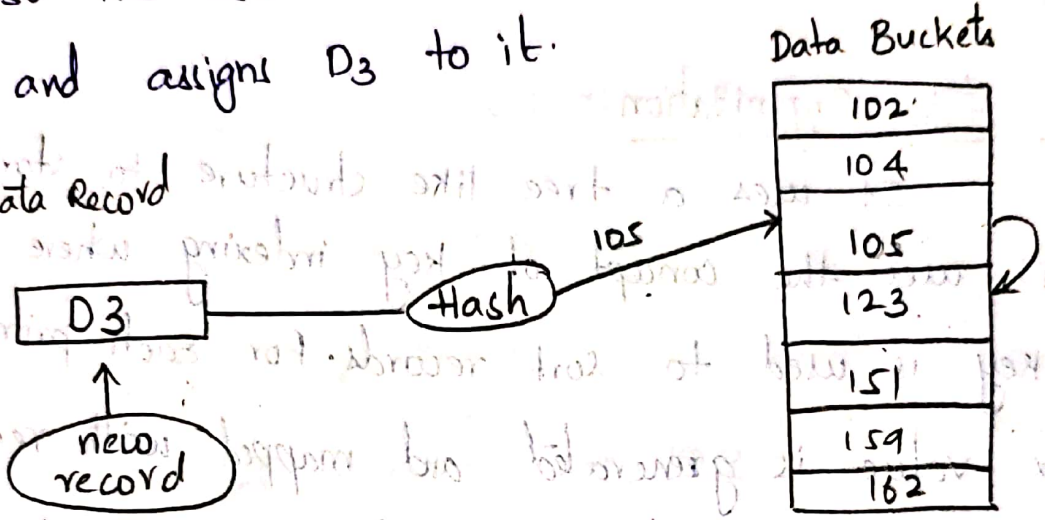
Open hashing

Closed hashing.

Open Hashing:-

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

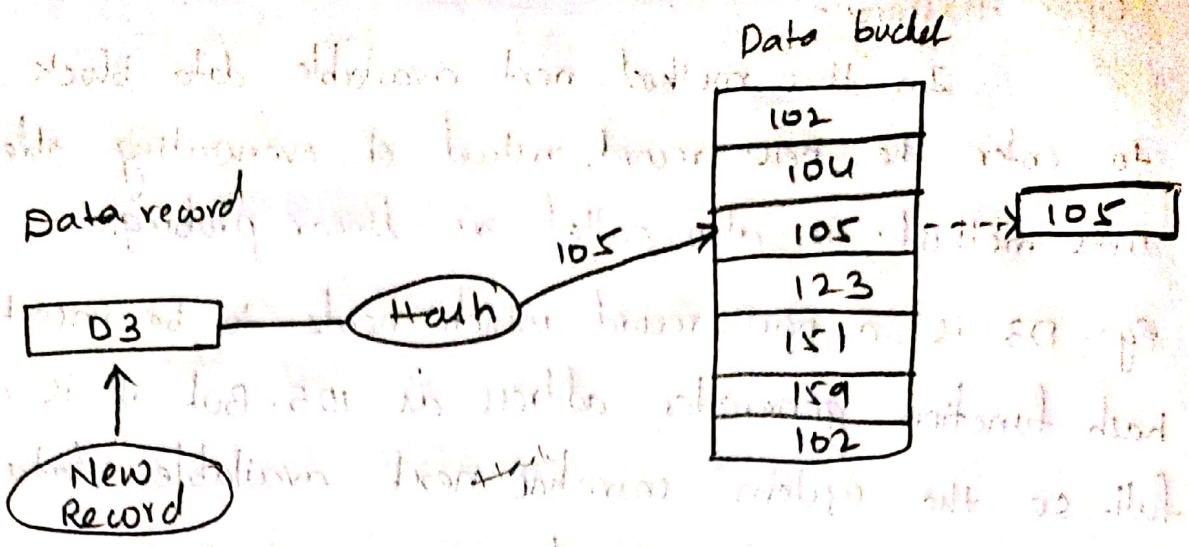
Eg:- D3 is a new record which needs to be inserted. The hash function generates address as 105. But it is already full. so the system searches next available data bucket 123 and assigns D3 to it.



Closed Hashing:-

In this method a new data bucket is allocated with same address and is linked with after the full data bucket. This method is also known as overflow chain.

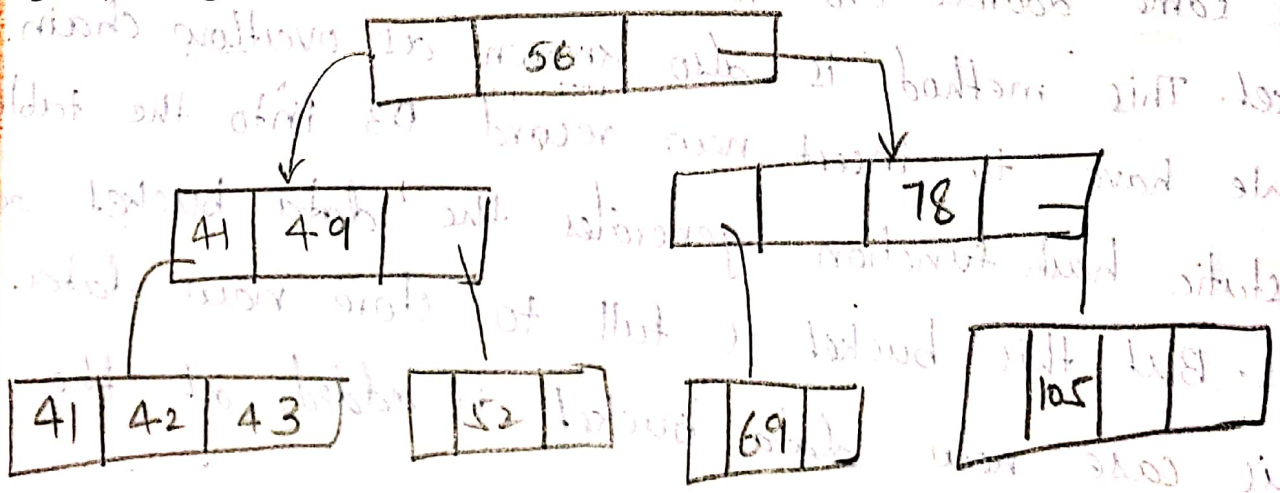
Eg:- We have to insert new record D3 into the tables. The static hash function generates the data bucket addr as 105. But this bucket is full to store new data. In this case new data bucket is added at the end of 105.



B+ Tree File Organization:-

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 records. For each primary key and index value is generated and mapped with record.

B+ tree is similar to binary search tree, the only difference is binary tree will have just 2 children B+ tree can have more than 2.



In this diagram 56 is the root node which is also ca as main node of tree.

→ The intermediate nodes contain address of the leaf node do not contain actual record. Leaf nodes contain actual

Pros and cons of B+ tree

Pros:-

→ Tree traversal is easier and faster

→ Searching becomes easy as all the records are stored only in leaf node & sorted

→ There is no restriction on B+ tree size. It may grow/shrink as the size of data increases/decreases.

Cons:

→ Inefficient for static tables

Note:- Searching is easier because all leaf nodes are balanced.

Cluster File Organization:

In this two or more related tables/records are stored within same file known as clusters. It retrieves various records in different files.

→ Lowers the cost of searching and

Emp

eid	ename	deptid
1	A	d_1
2	B	d_2
3	C	d_3

Dept

deptid	dname
D1	Finance
D2	HR
D3	Admin

Join → After joining store this in single file.

Deptid	Dname	eid	ename
D_1	Finance	1	A
D_2	HR	2	B
D_3	AD	3	C

Cluster key

* Cluster key :-

It is the key with which the joining is performed.

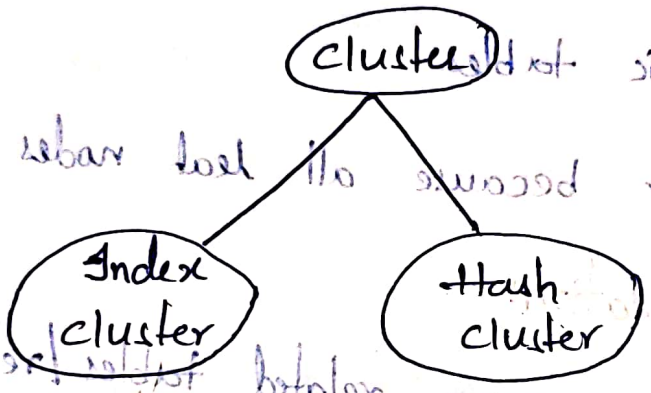
Advantages (Pros):

→ Used when frequent joining of table is required.

Cons:-

→ Low performance when tables are big.

Types of cluster:



Index cluster:- In this the records are grouped based on the cluster key and stored together.

Hash cluster:

In this method, we generate hash key value & store the records.

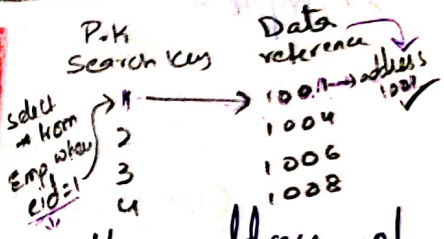
Indexing:-

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An index is a small table having two columns.

1. One column contains search key. Search key may be a primary key or candidate key.
2. second column contains data reference. It is a pointer holding the address of disk block.

TB → Pages
DB → Blocks

Structure of index ⟹



Primary key/ Candidate key. [sorted order]

Pointer holding the address of disk block.

Indexing in database is used to optimize the performance of database by minimizing the no. of disk access required when a query is processed.

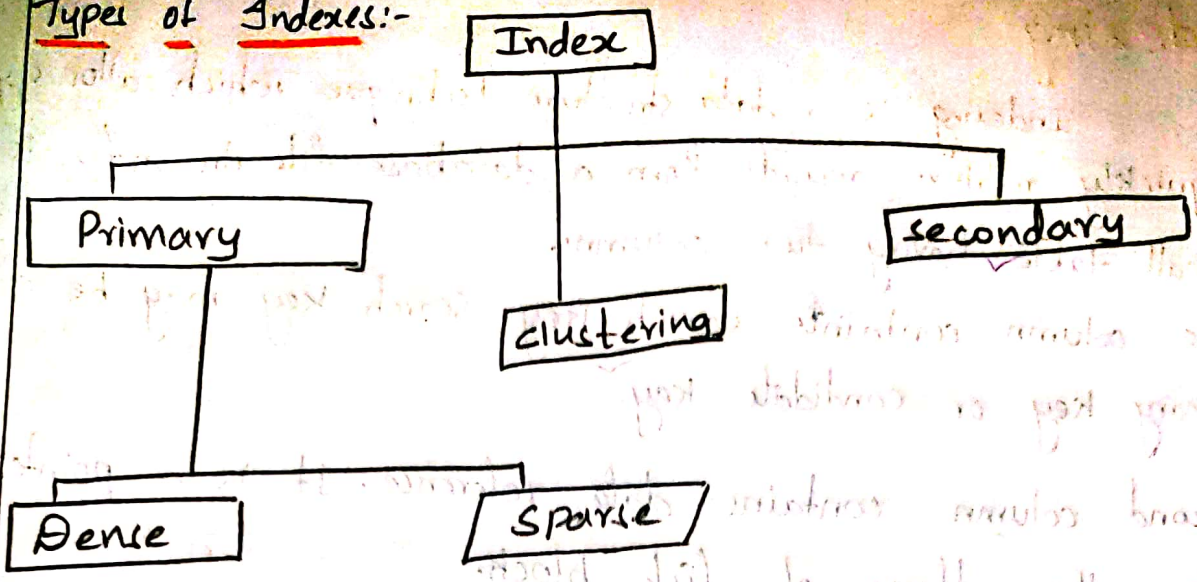
→ An index takes a search key as input.
→ Efficiently returns a collection of matching records.

Database indexing is defined based on its attributes.

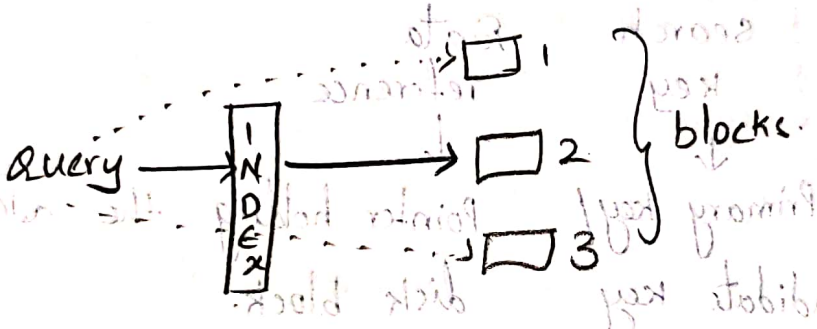
Two main types of indexing methods are:

1. Primary Indexing
2. Secondary Indexing.

Types of Indexes:-



Example:-



Textbook ——— Index

↳ Table of contents with pagenumbers.

→ Index in a data structure which is used to quickly locate and access the data.

Primary Indexing:-

Primary Index is an ordered file which is fixed length size with two fields. The first field is the same as primary key and second field is pointed to that specific data block. In the primary index, there is always one to

one relationship between the entries in the index table. The primary Indexing is also further divided into two types.

Primary Indexing

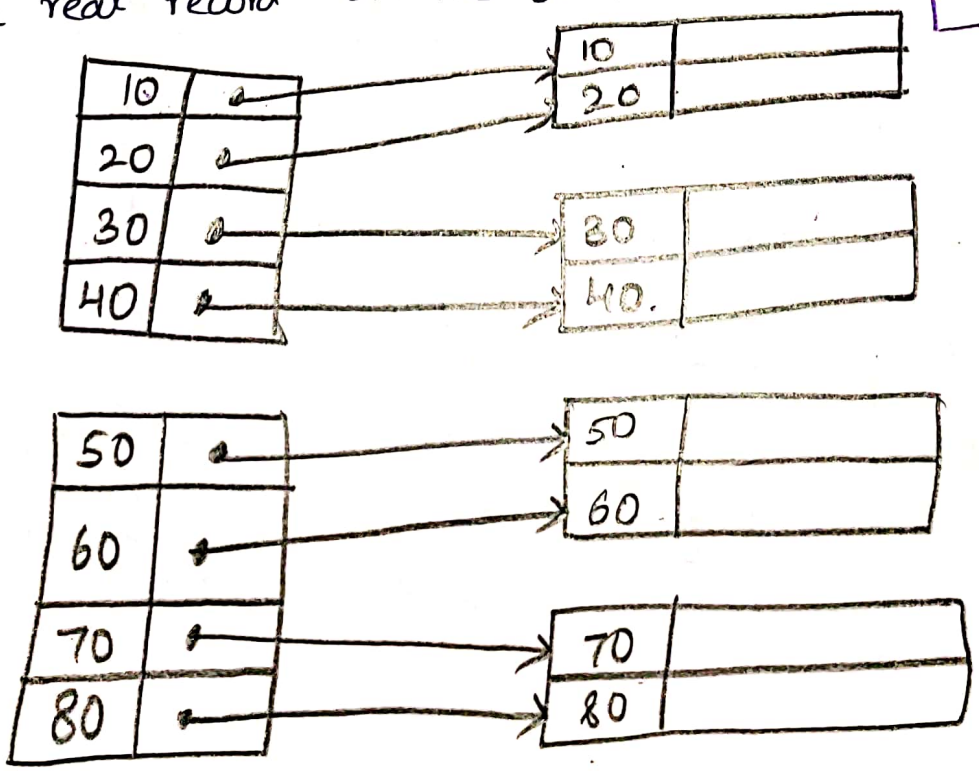
Dense Index Sparse Index.

Primary Index ⇒ key & ordered file.

Dense Index:-

In a dense index, a record is created for every search key value 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.

Search key = records

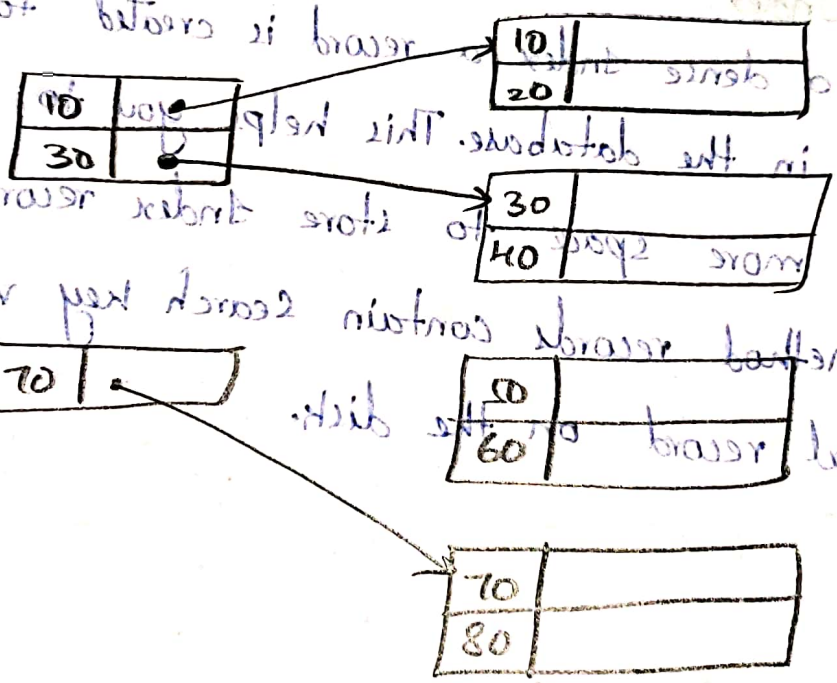


Sparse Index:-

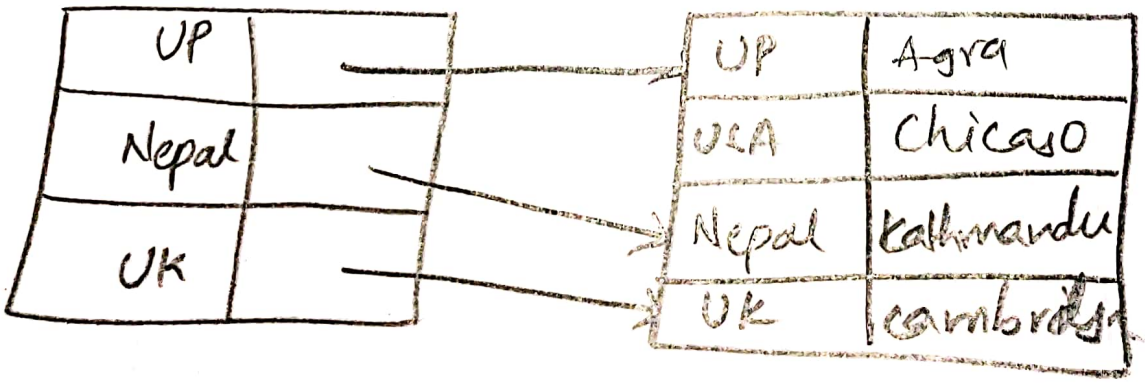
It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing. In this method of Indexing technique, a range of index columns stores the same data block address and when data needs to be retrieved, the block address will be affected. fetched.

It stores index records only for some search key values. It needs less space, less maintainance.

Eg-①



Eg-②

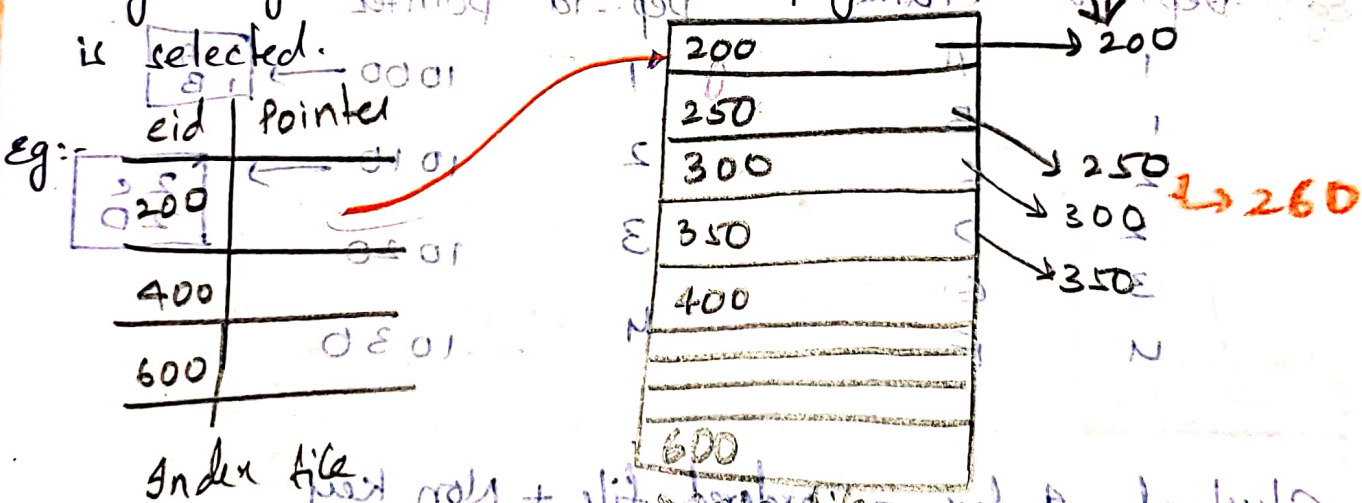


Secondary Index:-

The secondary Index can be generated by a field which has a unique value for each records it should be a candidate key. It is also known as non-clustering Index. This two-level database indexing technique is used to reduce mapping size of the first level. system admin will decide Levels used

first level → Large range of no. is selected.

second level → Actual physical location of data.



If I want to search eid = 260 then it will go to first level with eid 200 & at second level it goes to the block 200 → 250 → 260

Secondary Index = key + unordered file

(candidate key)

Clustering Index:-

In a clustered index, records themselves are stored in the index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such situation you can group two or more columns to get values and create an index which is called a cluster ~~file~~ index.

In this two or more columns are grouped together to uniquely identify the records.

→ Records with similar characteristics are grouped together and indexes are created for these groups.

Eg:-

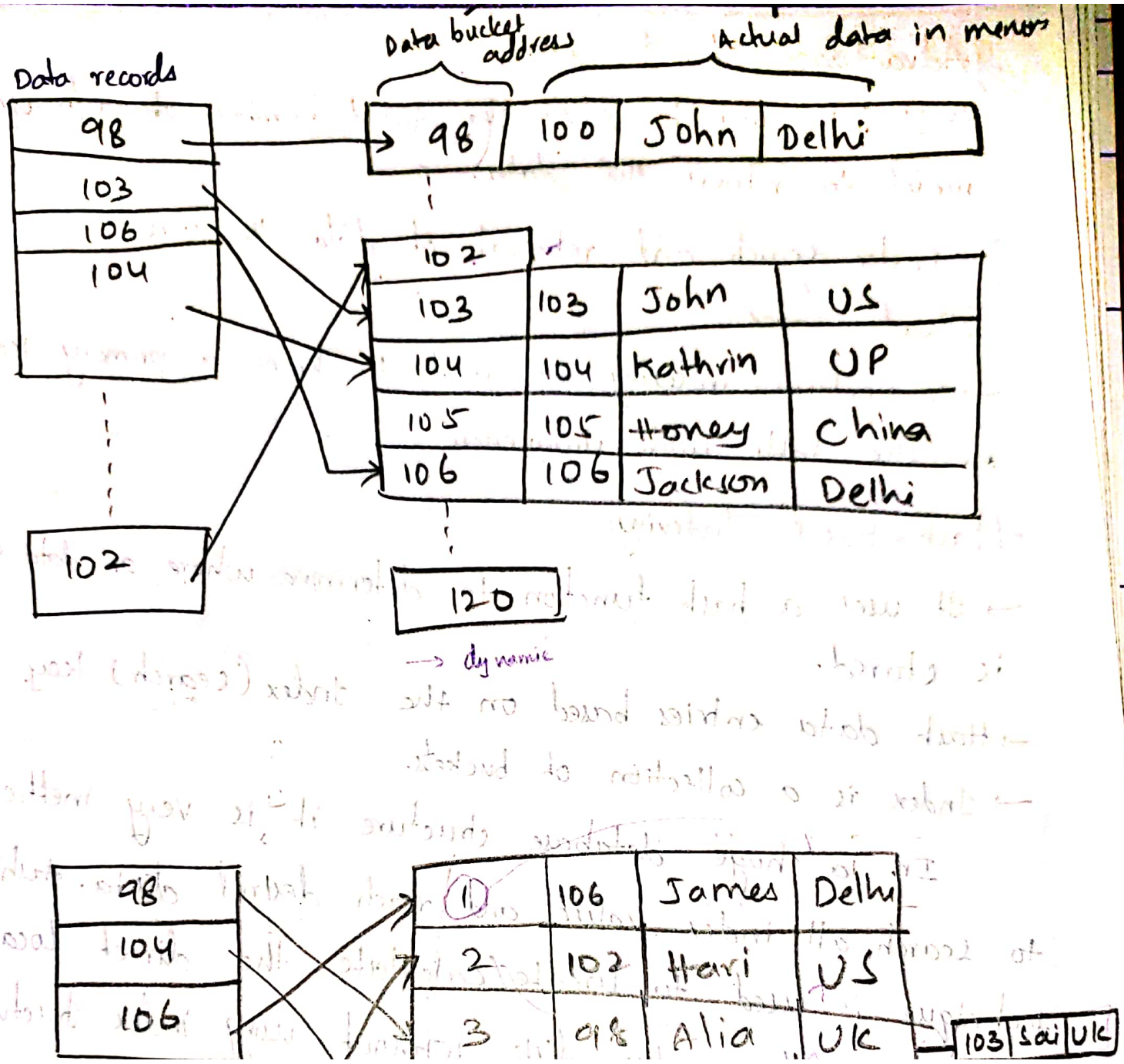
Dept_id	ename
1	A
1	B
2	C
2	D
3	E
4	F

Dep_id	pointer
1	1000 → [1 A, 1 B]
2	1010 → [2 C, 2 D]
3	1020
4	1030

Clustered Index = ordered file + Non key

↳ No uniqueness.

Eg:- 1
1
2
2
3
3



Advantages of Indexing:-

→ It helps you to reduce the total number of I/O operations needed to retrieve the data.

→ Faster search and retrieval of data to users.

Disadvantages:-

→ To perform indexing we need to have a primary key on the table with uniqueness.

Hash-based Indexing:-

→ It uses a hash function to determine where a data entry is stored.

→ Hash data entries based on the Index (search) key.

→ Index is a collection of buckets.

In a huge database structure it is very inefficient to search all index value and reach desired data. Hashing technique is used to locate/calculate the direct location of 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 hash function. The memory location where these records are stored is known as bucket blocks or data bucket.

Hash function uses primary key to generate the address of data block.

Eg:-

Transaction table TT

TID	last LSN
1	567
2	42
7	67
3	12

Dirty Page Table DPT

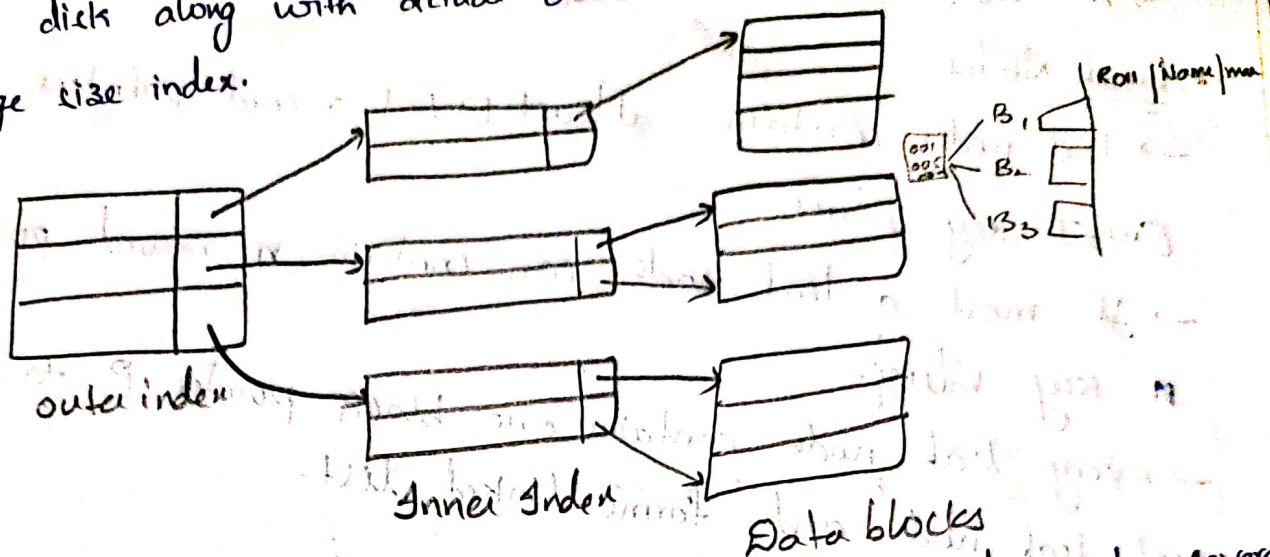
PageID	recovery LSN
42	567
46	568
77	34
3	42

TID :- key of transaction

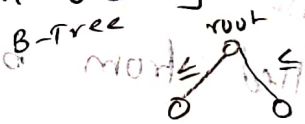
→ Page may contain changes from losers & winners i.e two concurrent transaction may operate on same page.

Tree based Indexing:-

It is a multi level index format which is balanced binary search trees. Multi level index is stored on the disk along with actual database file. It is used for large size index.



Multi level index helps in breaking down the index into several smaller indices.

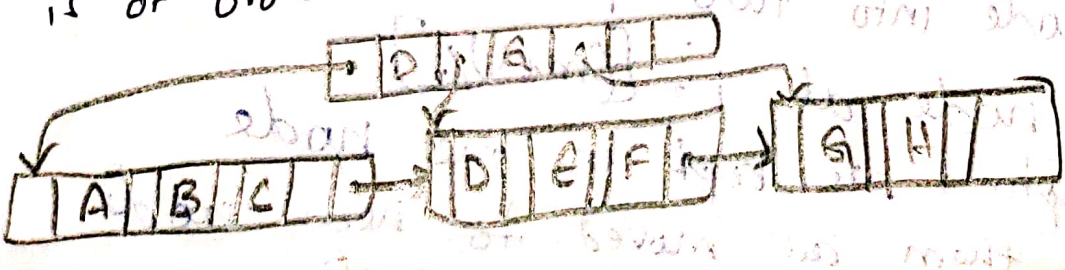


B+ Tree:-

A B+ tree is a balanced binary search tree that follows a multi level index format. The leaf nodes of B+ tree denote actual data pointers. B+ Tree ensures that all leaf nodes remain at the same height thus balanced. It can support random access as well as sequential access.

Structure of B+ tree:-

Every leaf node is at equal distance from root node. A B+ tree is of order n where n is fixed for every B+ tree.



- **Internal nodes:**
  → Internal (non-leaf) nodes contains at least $\lceil n/2 \rceil$ pointers except the root node.
  → At most internal node can contain n pointers.
- **Leaf Nodes:**
  → leaf nodes contain atleast $\lceil n/2 \rceil$ record pointers and $\lceil n/2 \rceil$ key values.
  → At most a leaf node can contain n record pointers and n key values.
  → Every leaf node contains one block pointer P to point to the next leaf node and forms linked list.

B+ tree Insertion:-
→ B+ trees are filled from bottom and each entry is done at leaf node.
→ If a leaf node overflows:
  - Split node into two parts
  - Partition at $i = \lceil (m+1)/2 \rceil$ (m → no. of pointers)
  - First i entries are stored in one node
  - Rest of entries (i+1) are moved to new node.
  - i^{th} key is duplicated at parent of leaf

→ No overflow chain
→ Root to leaf path length will same

→ If a non-leaf node overflows
  - split node into two parts.
  - Partition node at $i = \lceil (m+1)/2 \rceil$
  - Entries upto i are kept in one node
  - Rest of them are moved to new node

B* Tree Deletion:-

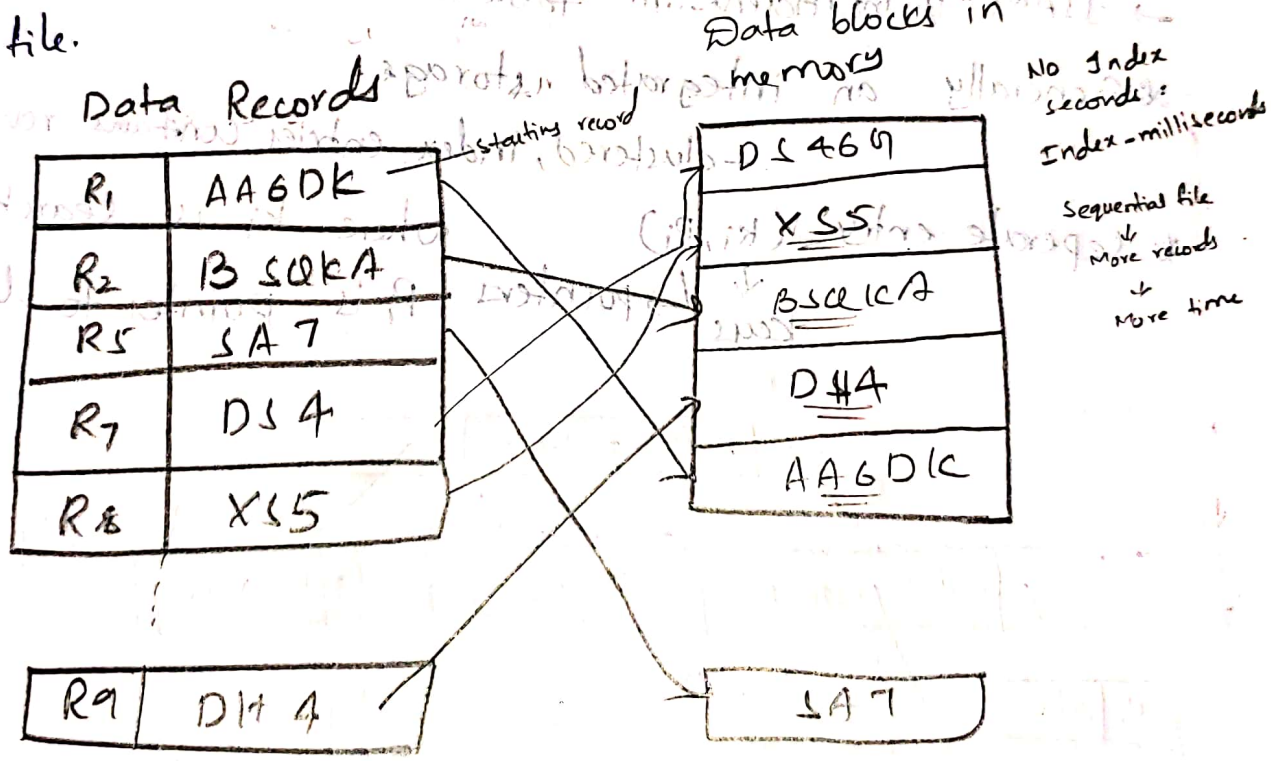
→ B+ tree entries are deleted at leaf nodes.

→ Target entry is searched and deleted.

→ After deletion underflow is tested.

Indexed Sequential Access Method:-(ISAM)

This method is an advanced sequential file organization. In this method records are stored using a primary key. An index value is generated for each primary key and mapped with record. This index contains the address of the record in file.

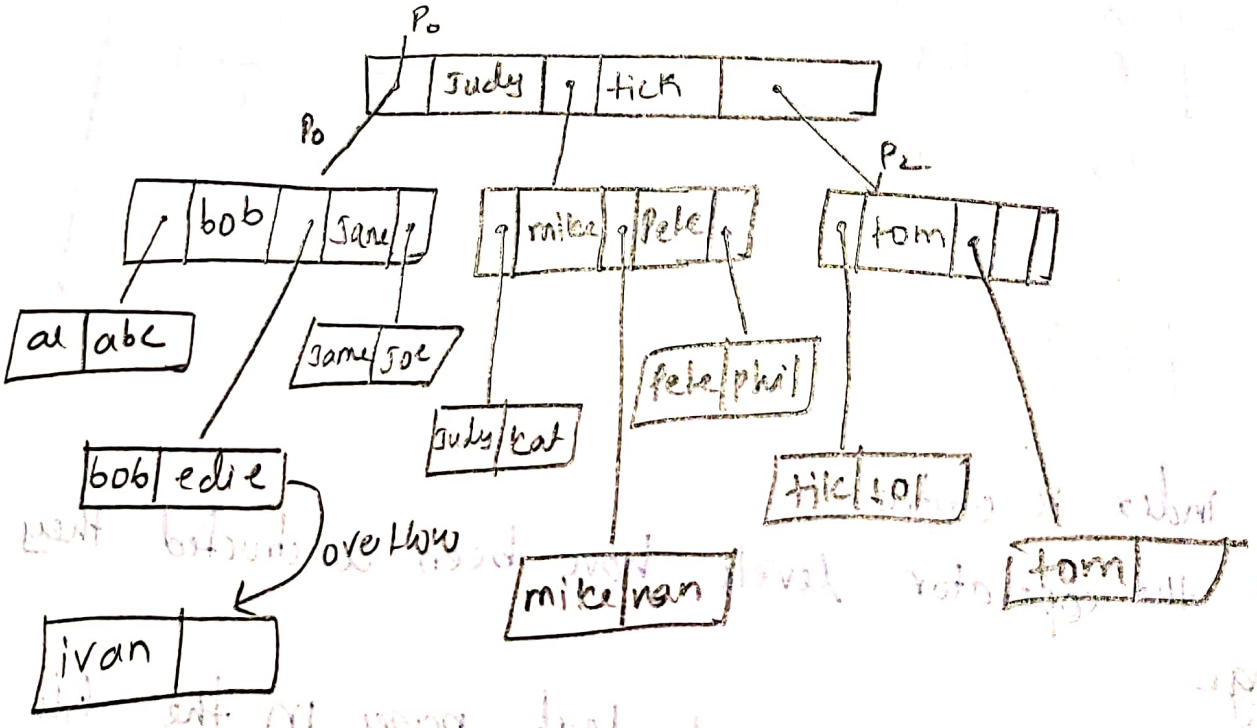


* The index is static

→ once the seperator levels have been constructed they never change

→ Number and position of leaf pages in the file stays fixed.

* Good for equality and range searches.
  → leaf pages stored sequentially in file when storage structure is created to support range searches.
* Supports multiple attribute search keys and partial key searches
→ contents of leaf page can change
→ Row deletion yields empty spot in leaf page.
→ Row insertion can result in overflow leaf page
→ chain might be long
→ ISAM is inefficient if table is dynamic
* Generally an integrated storage.
  – clustered, index entries contains rows
* seperate entry = (ki, Pi) where ki is search key value
  ↓ keys ↳ pointers Pi is pointer to low level page



→ In ISAM the no. of leaf pages are fixed at the time of creation.

→ The memory in ISAM is allocated to Data pages → Index pages

Pointers → 3

Keys → 2

Indexes And Performance Tuning:- Improvement of system Performance

Here, we will discuss when we use indexes in our database how to improve the performance of the database system. The choice of indexes has tremendous impact on system performance and expected workload.

Impact of Work load:- The amount of work that machine produces

Index supports efficient retrieval of data entries that satisfy a given selection condition. We have two kinds of selection. They are → enhancing both performance of database → minimizing the response time at optimum cost.

① Equality selection → Query optimization is one in dbms.

② Range selection: fast execution cost minimization

Hash based indexing technique are optimized only for equality selection.

Tree based indexing supports both kinds of selection conditions efficiently.

Both tree & hash based indexes supports inserts, deletes and updates efficiently. Tree based indexes offer a superior alternative for maintaining fully sorted files of records.

① We can handle inserts and deletes of data entries efficiently

② Finding the correct leaf page when searching for a record by search key value.

Clustered Index Organization:

A clustered index is a file organization for data records. Data records can be large and we should avoid replicating them. So, there can be one clustered index on a given collection of records. On the other hand, we can build several unclustered indexes on a data file.

Eg: employee records are sorted by age and are stored in a clustered file with search key age. And latter must be an unclustered index.

clustered index are less expensive to maintain.

When a new record has to be inserted into a full leaf page, a new leaf page must be allocated and some existing records have to be moved to new page. We use index only evaluation concept.

In an index only evaluation of a query is done by not accessing the data records in the files but relations in the query can be evaluated completely through index on the file. An benefit with index only evaluation is that it works equally efficiently with unclustered indexes.

Eg: SELECT E.dno FROM Employees E WHERE E.age

We have B+ tree index on age, we can use it to retrieve only tuples that satisfy age≥40.

Index depends on condition. If everyone is older than 40 sequential scan is done

Composite search keys:

The search key for an index can contain composite search keys or concatenated keys.

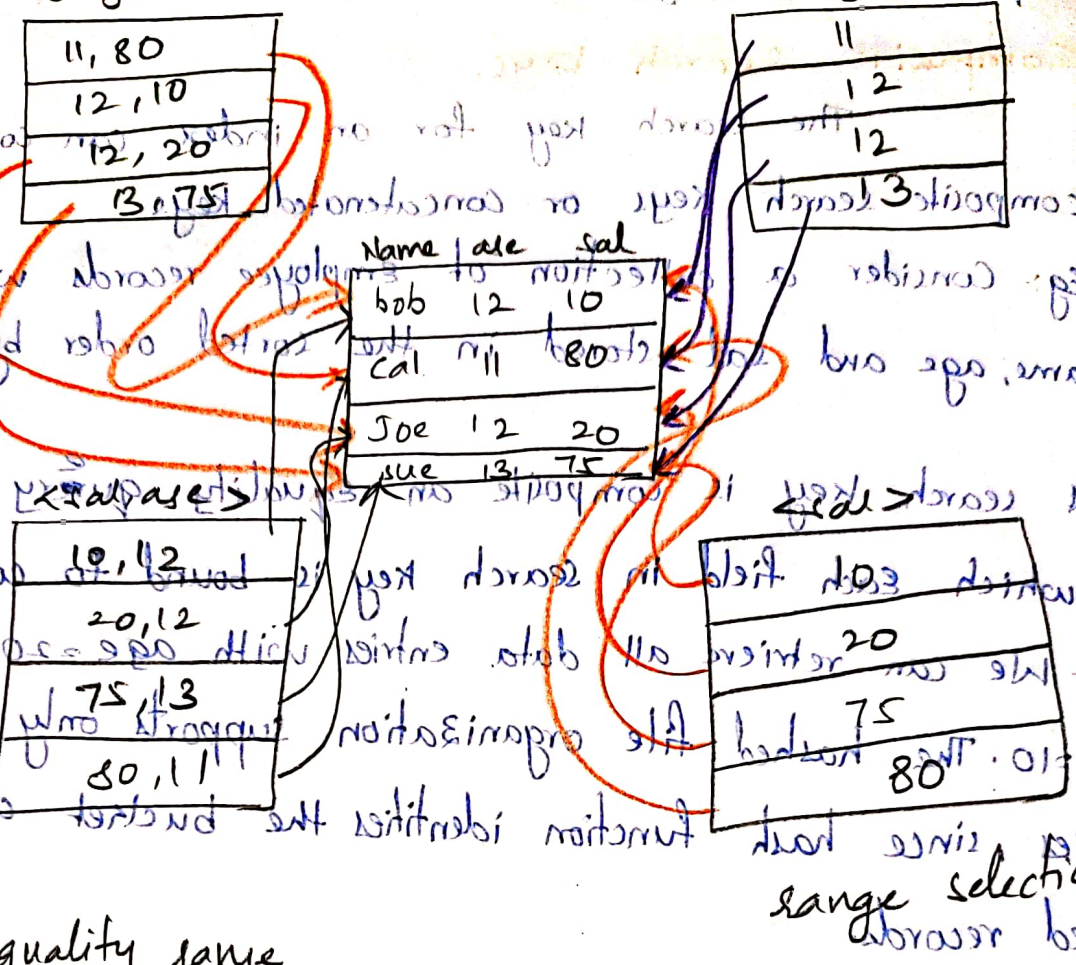
eg:- Consider a collection of employee records with fields name, age and sal stored in the sorted order by name.

If search key is composite an equality query is one in which each field in search key is bound to constant.

eg:- We can retriere all data entries with age=20 and sal=10. The hashed file organization supports only equality queries, since hash function identifies the bucket containing desired records.

With respect to range query not all the fields in the search key are bound to constant.

eg:- We can retrieve all data entries with age<30 and sal>40.



Equality range

range selection

Index Specification in SQL:1999

The SQL:1999 standard does not include any stmt for creating or dropping index structure. DBMS supports one or more kinds of indexes.

Eg: CREATE INDEX Ind Age Rating ON students with STRUCTURE = BTREE KEY = (age, gpa)

This specifies that B+ tree is to be created with columns age and gpa as key.

Comparisions Of File Organizations:

We now compare the cost of some simple operations for basic file organization on a collection of records. The operations we consider are:

1. Scan
2. search with equality selection
3. search with range selection
4. Insert a record
5. Delete a record.

- **Scan:-** Fetch all records in file. The pages in the file must be fetched from disk into the buffer pool. There is also CPU overhead per record for locating the record on the page.
- **Search with Equality selection:-** Fetch all records that satisfy an equality selection.

→ eg:- Find the employee record with age = 30 and sal = 50 ?

Pages that contain qualifying records must be fetched from disk and qualifying records must be located within retrieved page.

- **Search with Range Selection:-**

Fetch all records that satisfy a range selection

eg:- Find all employee records with age greater than 35

- **Insert a Record:-**
  Insert a given record into file. We must identify the page in the file into which the new record must be inserted, fetch that page from disk, modify it to include the new record, and then write back the modified page. Depending on the file organization, we may have to fetch, modify and write back other pages as well.

- **Delete a Record:-**
  Delete a record that is specified using its rid. We must identify the page that contains the record, fetch it from disk, modify it, and write it back.

Cost Model:-

In comparision of file organizations we use cost model to estimate the cost of different database operations. Estimate the cost ↓ execution time.

We use B to denote no. of data pages.

R to " no. of records per page.

D to average time to read or write a disk page.

C → average time to process a record.

values for today are

D = 15 milliseconds

C & H = 100 nano seconds

↓

Hash function

We concentrate on I/O component of cost model.

→ Real systems must consider the costs of CPU

We will see the implications of cost model whenever our assumptions effect different files.

Heap files:-

Scan:- The cost is B(D+RC) because we must retrieve each of B pages taking time D per page and for each page process R records taking time C per record

Search with Equality Selection:-

Suppose, we know that exactly one record matches the equality selection, that is (we) have a candidate key. For each page we must check all records to see which record can be fetched. The cost is $D * B(D+RC)$. If no record satisfies the selection we must scan entire file.

If the selection is not on candidate key for eg:- "Find employees aged 18" we have to scan entire file whose age=18, because we don't know how many records can be fetched on this condition

Search with Range selections:- The entire file must be scanned because qualifying records could appear anywhere in the file we don't know how many qualifying records are there.

Insert:-

Records are always inserted at the end of file. We must fetch the last page in the file, add the record and write the page back. The cost is 2D+C

Delete:-

We must find the record, remove the record from the page and write the modified page back. The cost of searching is BD+D

Sorted Files:-

Scan:- The cost is B(D+RC) because all pages must be examined. There will be no better or worse than the case of unorderded file. It will be retrieved corresponding to sorted order.

Search with equality Selection:- We assume equality selection should match sorted order (age, sal). Selection condition is specified on atleast the afield in Composite key. If not the sort order does not help us and the cost is identical. We can locate the first page containing the desired record with a binary search in $\log_2 B$ steps. These sorted files are stored sequentially.

Search with range (Selection):-
The range selection matches the Composite key.
to on based. The first record satisfies the condition and the record is located.
for search with equality. Data pages are sequentially retrieved until a record is found that does not satisfy the range selection.
The cost is cost of search + cost of retrieving the set of records that satisfy the search.

Insert:-
To insert a record in sort order we must find the correct position in the file, add the record, then fetch and rewrite all subsequent pages.
cost is search + B (D+RC)

Delete:-
To delete a record, remove the record from page and write the modified page back. We must read and rewrite all subsequent pages because all records must be moved up and fill the free space.

Clustered files:-
In a clustered file pages are occupy 67% of space and physical data page occupies 1.5B

Scan:- The cost of scan is 1.5B(D+RC) because all data pages must be examined. Adjustment is done for increased no. of pages.

Search with Equality Selection:- We assume that an employee table equality selection matches with search key <age, sal>. Fetching all pages from root to appropriate leaf. Each step requires comparisons.

Search with range selection:- Range selection matches composite key. First record that satisfies the selection is located subsequently data pages are sequentially retrieved until a record is found that does not satisfy range selection.

Insert:- To insert a record, we must find the correct leaf page in the index, reading every page from root to leaf. Then, we can add new record. Most of the time leaf page will have sufficient space occasionally the leaf is full when the leaf is full we need to retrieve another pages.

The cost is cost of search $1.5B + C\log_2 1.5B$

Delete:-

We must search for record, remove the record and the modified page back.

Scanned with CamScanner

Comparision of I/O costs

File type	Scan	Equality Search	Range Search	Insert	Delete
Heap file	BD	0.5 BD	BD	2D	Search+D
Sorted file	BD	$D \log_2 B$	$D \log_2 B$ + # matching pages	Search + BD	Search + BD
Clustered	1.5BD	$D \log_F 1.5B$	$D \log_F 1.5B$ + # matching Pages.	Search + D	Search + D
Unclustered tree index	BD(R + 0.15)	$D(1+\log_F 0.15B)$	$D(\log_F 0.15B$ + matching records	$D(3 + \log_F 0.15B)$	Search + 2D
unclustered hash index	BD(R + 0.125)	2D	BD	4D	Search + 2D

Heap file has good storage efficiency and supports fast scanning and insertion of records.

sorted file offers good storage efficiency but ~~linear~~ insertion and deletion of records is slow. Searching is faster than heap file.

clustered file offers all advantage. Supports inserts and deletes efficiently. Searches are faster when a large no. of records are retrieved sequentially.

Unclustered tree and hash indexes offer fast searche insertion & deletion. but scans and range searches are slow.

B+ trees:

select * from employee where empid = 123 → equality search

→ It supports equality search and range search.

select * from employee where empid > 100 and empid < 400

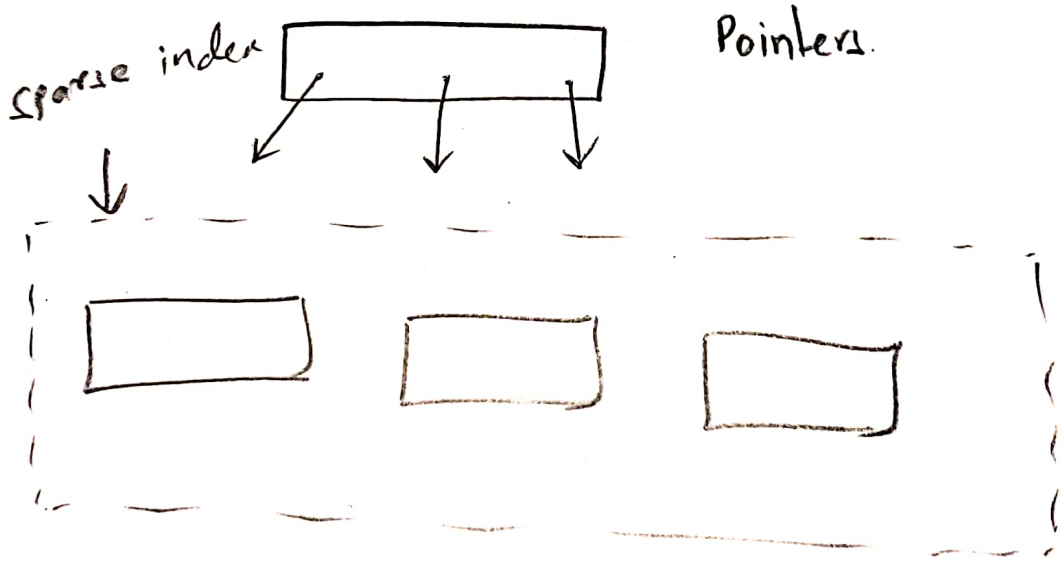
↓
range search.

→ secondary index (in seperate file)

- ↳ index entries at one file
- ↳ data file at one record.

→ Index & data file in same record.

→ Responds to dynamic changes in table.



ISAM: overflow pages, Locking Considerations:

Once the ISAM file is created inserts and delet affects only the contents of leaf pages. A consequence of this is that overflow chain increases as insertions are done on same leaf. This can affect the time to retriev

a record because the overflow chain has to be searched.

The fact is that only leaf pages are modified with respect to concurrent access. When a page is accessed it is typically "locked" by the user. To modify a page it must be locked in "exclusive" mode. Locking can lead to queues of users waiting to get access to a page.