UNIT - III DATA STORAGE INDEXING AND QUERY PROCESSING

Disk storage, Basic file structure and Hashing - Indexing structures for files - Algorithm for query processing and optimization - Introduction to PL/SQL: Procedures and Functions - Cursor.

DISK STORAGE:

→ The records in databases are stored in file formats.

→ Physically, the data is stored in electromagnetic format on a device.

→ The Database Management System stores information on hard disks.

→ This has major implications for DBMS design. like,

READ: transfer data from disk to main memory (RAM)

WRITE: transfer data from RAM to disk

→ Both are high-cost operations, relative to in-memory operations

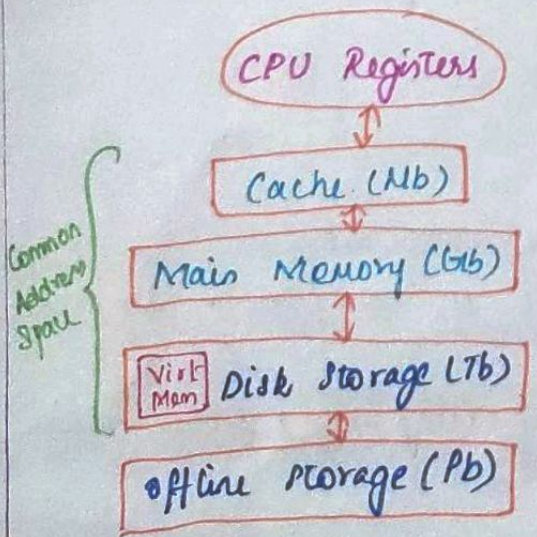
Typical storage hierarchy:

→ CPU Registers - temporary variables

→ Cache - frequently accessed Memory locations.

→ Main Memory (RAM) for currently used "addressable" data.

→ Disk for the main "big data" (secondary storage)



Disks:

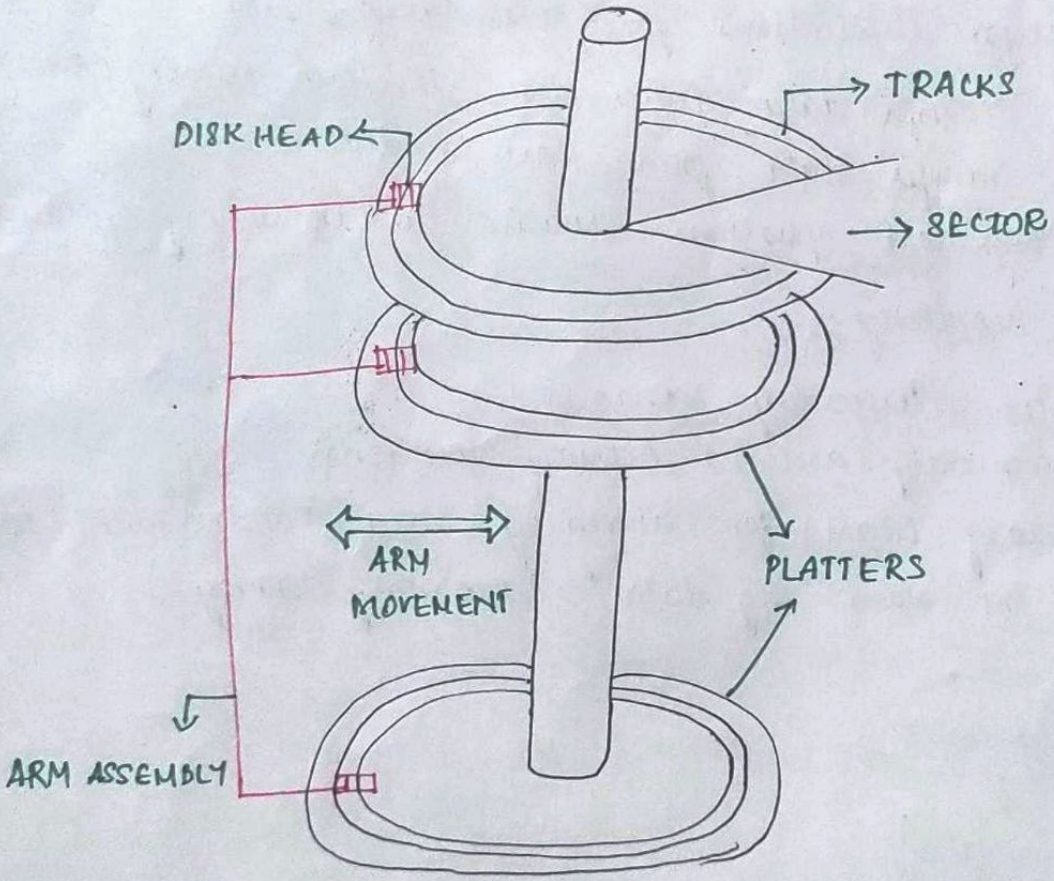
→ The Disks are secondary storage device of choice.

→ The main advantage of disks over other storage system is random access versus sequential.

→ Data is stored and retrieved in units called disks blocks or pages.

→ Time to retrieve a disk page varies depending upon the location on disk. Hence, the relative placement of pages on disk has major impact on the Database performance.

Components of a Disk:



→ The platters spin according to storage movement.

→ The arm assembly is moved in or out to position a head on a desired track.

→ The tracks under heads make a cyclinder (imaginary) fields while rotating.

→ Only one head read / write at any one time.

→ Block size is a multiple of sector size which is always fixed.
→ the time to access the disk block to read and write on the disks consists of the following factors.

Seek time: Moving the arms to position disk head on track

Rotational delay: Waiting for the block to rotate under head.

Transfer time: Actually moving data to / from disk surface.

(All the above factors happens while accessing the pages on the disk).

→ To arrange the pages on the disk we use the Next block concept. (i) blocks on same track, followed by
(ii) blocks on same cylinder, followed by
(iii) blocks on adjacent cylinder.

→ the blocks in a file should be arranged sequentially on disk (by 'next'), to minimize seek and rotational delay.

→ Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.

→ Two main techniques used!

Data striping: Data is partitioned; size of a partition is called the striping unit. The partitions are distributed over several disks.

Distributed Redundancy: More disks → failures. The Redundant information allows reconstruction of data if a disk fails.

DISK SPACE MANAGEMENT:

→ the Lowest layer of DBMS manages the space on disk.

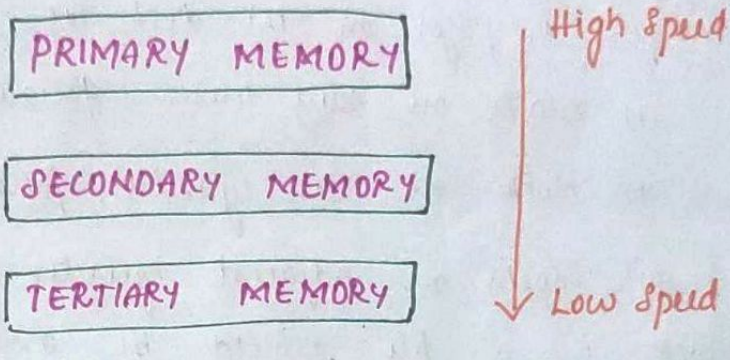
→ Higher levels call upon the layer to allocate and deallocate a page.

→ the pages on the disk should be arraaged in a sequential manner, so the spaa will be managed by satisfying this manner.

STORAGE SYSTEM

→ The capacity secured by the database management system is the memory of the server allocated for the database and the related operations.

Types of Data Storage:



Primary Memory: (Eg: Main Memory, Cache)

→ It is the area that offers quick accen to the stored data.
→ The Primary Memory or the Primary storage can also be called as volatile storage or volatile Memory.
→ This type of memory does not permanently store the data.
→ According to performance and speed, the primary memory devices are the fastest devices
→ Then devices are usually more expensive due to their increand speed and performance.

Secondary Memory:

→ It is the storage area that allows the user to save and store data permanently.
→ They can be called as backup units.
→ They are considered slowes ~~and~~ according to the performance

but it usually as a higher capacity than primary storage systems.

Eg:- Flash Memory - USB, Magnetic Disk storage.

Tertiary Storage:

→ They are used for holding massive amounts and are not required to be connected to the server.

→ They can provide more space than other types of device Memory.

→ They are commonly used one for making copies from the server and database.

→ Similar to Secondary Memory, they also utilize the Memory space wherever required and to delete the contents when not required.

Eg:- Optical Storage - Compact Disk (CD), Tape Storage.

FILE: (BASIC FILE STRUCTURE)

A collection of pages, each said to contain a collection of records which should allow the following to support.

* Insert / delete / modify record
* Read a particular record
* Scan all the records.

File Organization:

→ It determines how the records of files are mapped to block of disks.

→ It defines the logical relationship among various records

→ They used to describe the way in which the records are stored in terms of blocks.

→ Files of fixed length are easier to implement than with the files of variable length.

Objectives:

* The records can be selected as fast as possible.
* The operations like insert, delete or update should be quick and easy.
* Due to the above listed operations the duplicate records cannot be induced.
* The records should be store efficient in order to maintain the minimal cost of storage.

Types of file Organization:

Sequential File Organization:

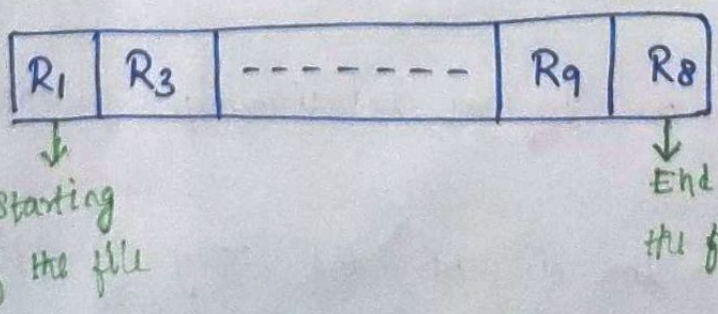
→ The Records are put in the file in a sequential order in the sequential file organization.

→ Every Record contains a data field (attribute) to define or identify the record uniquely.

→ Two ways to implement this:

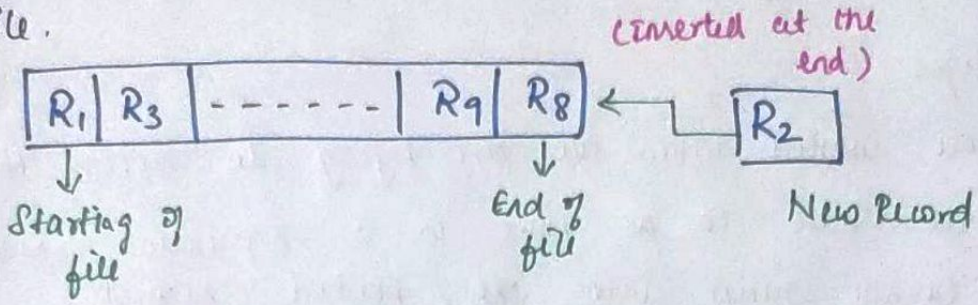
(i) File Fill Method;

The records are stored in a sequence one after the another in which they are inserted into the tables.



Insertion of New Record:

The Records are considered as the rows in a table, hence if we want to insert a new record R_2 in the above sequence, then the corresponding new record will be placed at the end of the file.



(ii) Sorted File Method:

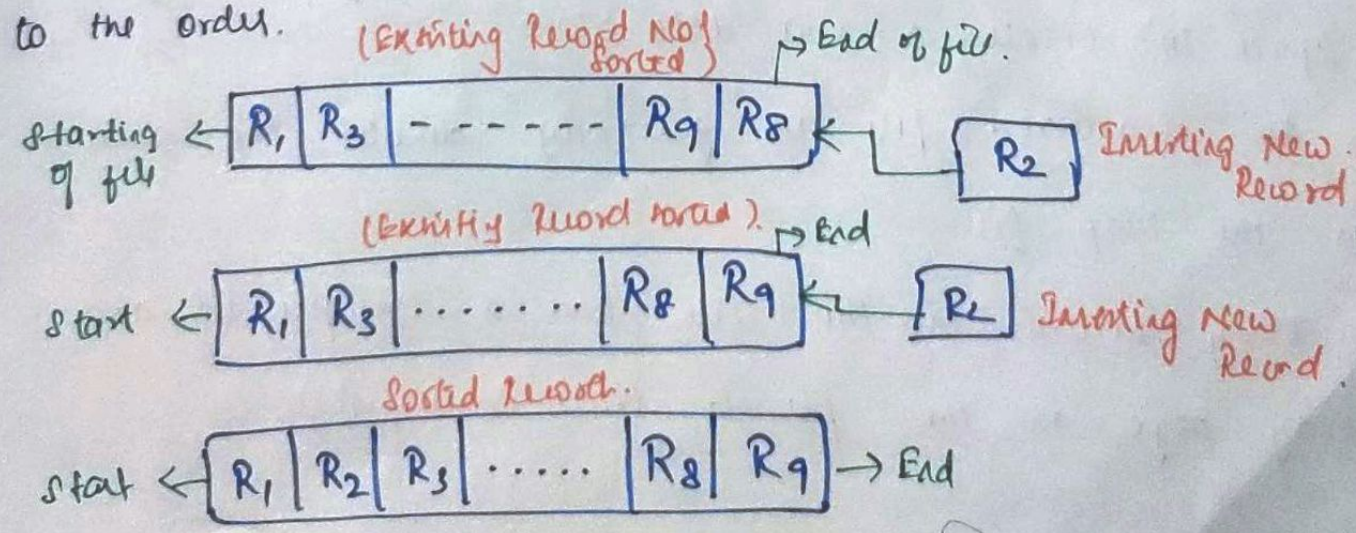
→ The Records are inserted at the file's end and then it is sorted in ascending or descending order.

→ Based on the primary key or any other key the sorting of the records are performed.

→ Incase of updation of records, the update is done and then only the file is sorted.

Insertion of New Record:

The Records which are already inserted are sorted in sequence, so once a new record is inserted, they are inserted at the end of file and then sorted according to the order.



Advantages:

* It is fast and efficient when we use huge amount of data
* Records can be easily stored and considered to be cheap.
* Requires less effort to store the data.

Disadvantages:

* Time gets wasted when we are trying to access the file because we have to go more in a sequential manner. (ie) we cannot jump into the needed record.
* Requires more time and space.

Heap File Organization:

→ This file structure works with data blocks.

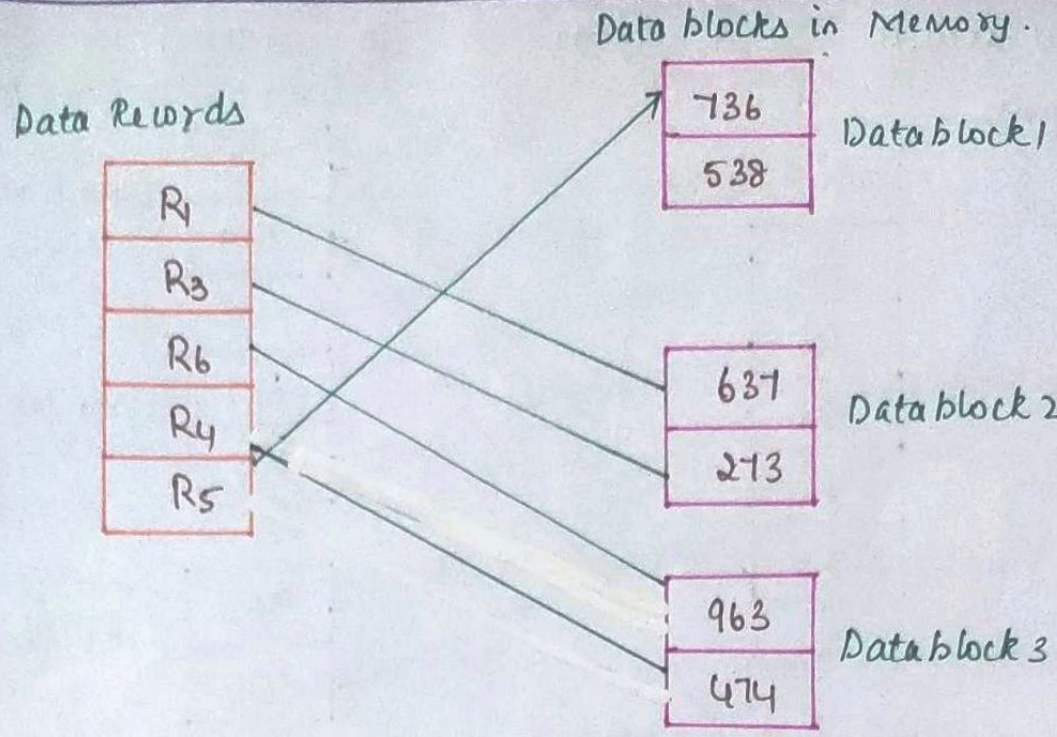
→ The New Records are added at the end of the file but they are not sorted or ordered in any way.

→ The New record is put in a different data block if the previous data block is full.

→ The New data block doesn't start right next to the previous data block but it can take any memory space to create a new data block.

→ An unordered file is generally denoted the same as the heap file.

→ Every record in the file has a unique id and every page in the file is the same size.



Insertion of a New Record:

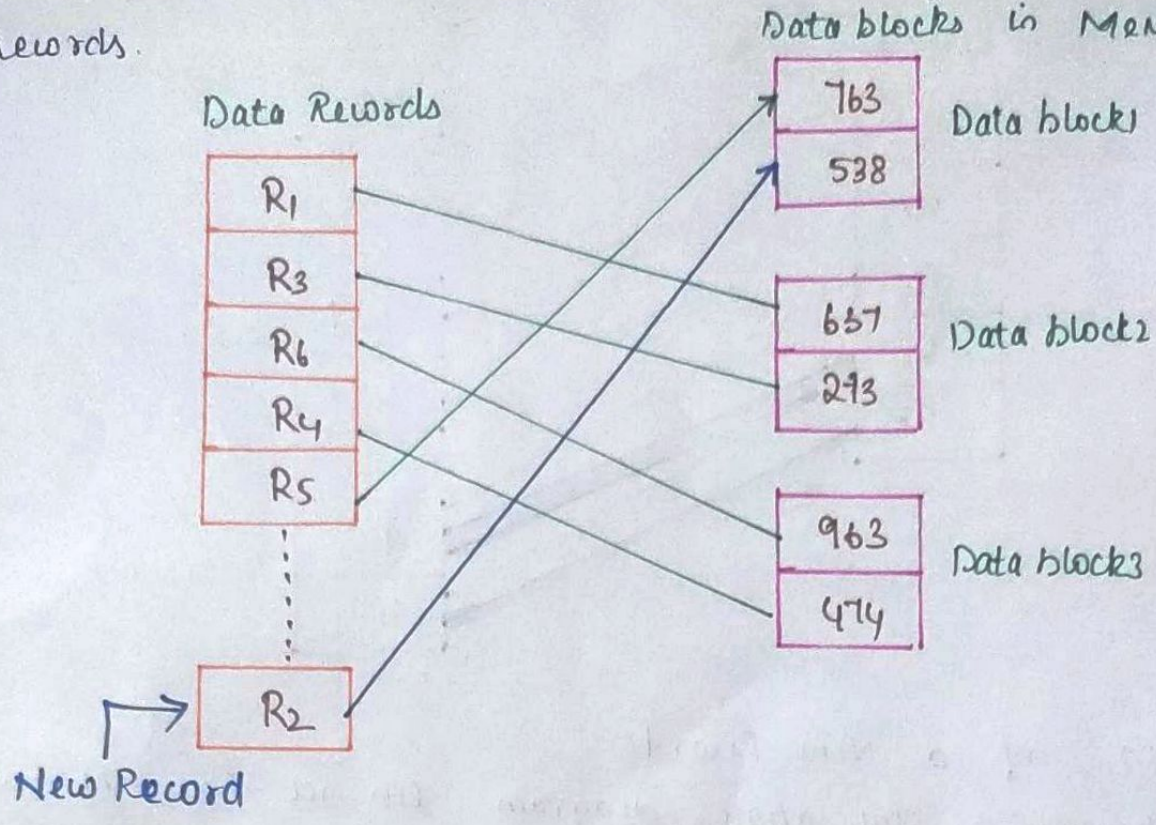
→ Considering the above diagram lets see how the insertion of New Record is done.

→ When a New Record is need to Inserted as we know it is Inserted at the end of the file or the space in which they can fit in a particular data block which have empty space.

→ For example if we need to add a New Record R2, it will first check which data block is having space to fit in, Data block 1 has a free space to fit in so the record gets Inserted there.

→ If we need to do other operations like search, update or remove a data, we must traverse the data from begining of the file until we find the

desired record since there is no ordering or sorting of records.



Advantages:

* Faster incase of a small databases.
* Appropriate for large data when it is to be loaded in databases.

Disadvantages:

* Inefficient due to time constraint
* For complicated databases not flexible.

Hash file Organization:

→ Direct file organization is known as hash file organization.

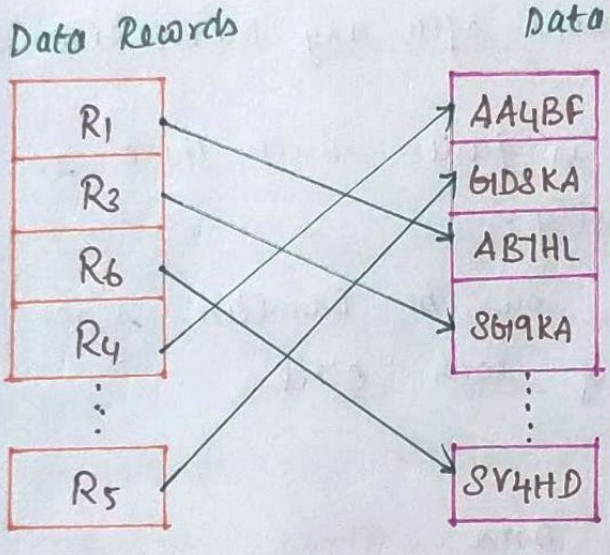
→ A hash function is calculated in this approach for storing the records which provides the address of the block that stores the record.

→ The calculated hash function is applied on the columns / attributes either key or non-key columns to get the block address.

→ Due to this reason the record is stored randomly irrespective of the order they come.

→ Hash key: If the hash function is generated on key column, then that column is called hash key.

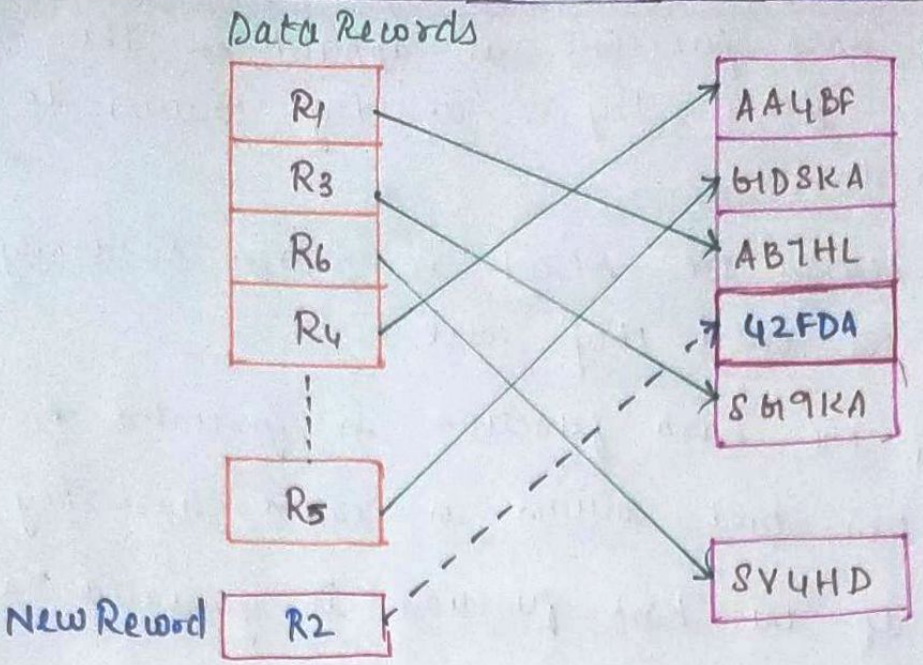
→ Hash column: If the hash function is generated on key column, then that column is hash column.



→ When a record has to be retrieved, based on the hash key column, the address is generated and directly from that address whole record is retrieved.

Inserting a Record:

The address is generated by hash key and record is directly inserted. Each record will be stored randomly in the memory.



Advantages:

* There is no sorting done after any transaction so the effort is less
* Accessing the record is faster since there is hash function to identify easily.
* Multiple transactions can be handled since each record is independent of each other.

Disadvantages:

* There can be loss of Data
* the memory is not efficiently used.
* Range of data cannot be accessed because each record is scattered in various places.

Cluster file Organization:

→ In all the above methods retrieving the data is tough / difficult because the data / records are scattered in Memory.

→ In order to retrieve the data we need to

The two students who have opted for 'Database' and 'Perl' course each.

Though it is stored in separate tables in logical view, when it is stored in physical view, we have combined them. (This is called as the cluster file).

→ The data are sorted based on primary key or the key with which we are searching the data.

→ The clusters are formed based on the join condition.

→ Cluster key: The key with which we are joining the tables is known as cluster key.

Types:

(i) Indexed Clusters: The records are grouped based on the cluster key and stored together.

This method is followed when there is retrieval of data for range of cluster key values or when there is a huge data growth in the cluster.

(ii) Hash cluster: Instead of storing the records based on the cluster key, we generate the hash key value for the cluster key and store the records with same hash key value together in the memory.

Advantages:

* This method is suited when there is frequency in joining the tables with same joining condition.
* 1:M mapping happens between the tables, it retrieves efficiently.

Traverse all the data to retrieve a particular data.

→ To overcome this we can use cluster file organization which helps us to retrieve the data.

→ When the datas are stored in various tables, those tables which are frequently used and joined and stored in the same file called clusters.

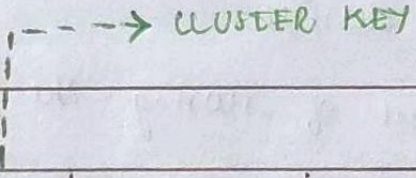
→ Example:

STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS	COURSE_ID
100	Kathy	Troy	230
101	Patricia	Clinton	240
102	James	Fraser Town	230
103	Anatomy	Novi	250
104	Charles	Novi	250

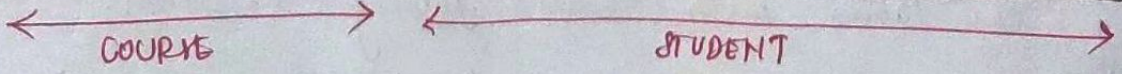
COURSE

COURSE_ID	COURSE_NAME
230	Database
240	Java
250	Pearl.



CLUSTER FILE

COURSE_NAME	COURSE_ID	STUDENT_ID	STUDENT_NAME	ADDRESS
Database	230	100	Kathy	Troy
		102	James	Fraser Town
Java	240	101	Patricia	Clinton Township
DBL	250	103	Antony	Novi
		104	Charles	Novi



Disadvantages:

* Not suitable for very large databases since the performance of the method on them is low.
* Incase of any change in joining condition we cannot use clusters.
* Not suitable for less frequently joined tables or tables with 1:1 condition.

HASHING:

→ It is a technique to directly search the location of desired data on the disk without using index structure.
→ It is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value.
→ The data is stored in the form of data blocks whose address is generated by applying a hash function
→ The records are stored as Data block or data bucket.

Why hashing is needed?

→ For a huge database structure, it is tough to search all the index values, so using hashing we need to reach the destination of the exact data.
→ It is used to index and retrieve items in a database
→ It is an ideal method to calculate the direct location of a data record on the disk.

Terminologies:

Data bucket – They are memory locations where the records are stored. It is also known as Unit of storage.

Key - A DBMS key is an attribute or set of an attribute which helps you to identify a row (tuple).

Hash function - A hash function, it is a mapping function which maps all the set of search key to the address where actual records are placed.

Linear probing - It is a fixed interval between probes. In the next available data block is used to enter the new record, rather than overwriting.

Quadratic probing - It helps to determine the new bucket address. It also helps to add interval between probes by adding consecutive output.

Hash index - It is denoted as the address of the data block.

Double hashing - It is a method used in hash tables to resolve the issues of collision.

Bucket overflow - The condition of bucket overflow is called collision.

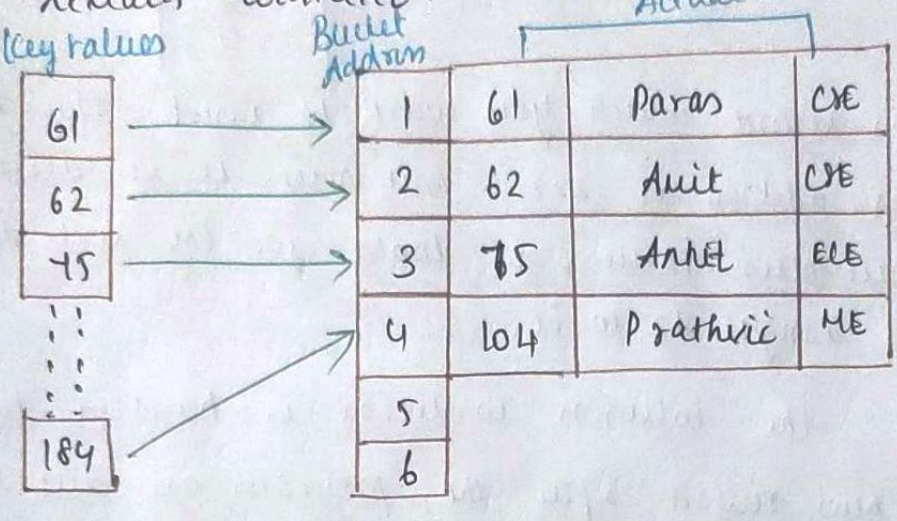
Types:

1) Static hashing: The resultant data bucket address will remain the same.

(ie) for a key k, hash function h(x) always generates the same memory address.

For example, if h(x) = x % 7 then for any 'x' the value will always be the same.

Therefore, the number of data buckets in memory always remains constant.



Opeations in Static hashing:

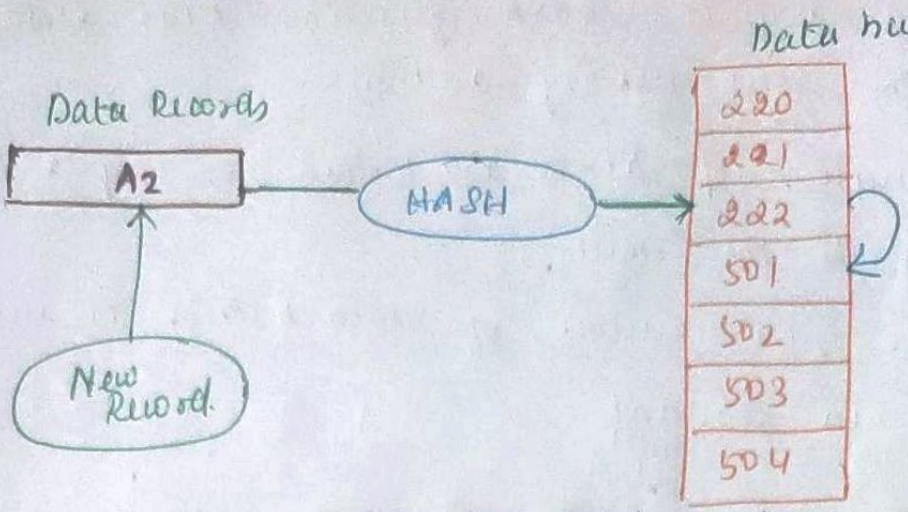
(i) Insertion: To insert a new record, the address is generated using the hash key. Once the address is generated, the record is automatically stored in the location.

(ii) Searching: When a record needs to be retrieved the record, the same hash function should be helpful to retrieve the address of the bucket where data should be stored.

(iii) Delete a Record: Using the hash function first the record is fetched which we need to delete and later the records for that address in memory is removed.

Static hashing types:

(i) Open hashing: Instead of overwriting older one the next available data block is used to enter the new record.



A2 is a new record which you want to insert. the hash function generates address as 222. But since it is already occupied by some other value, it looks for the next data bucket 501 and assigns A2 to it

(ii) 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".

Dynamic hashing: It is shortening a string of characters It creates a smaller, adaptable string of characters, making it faster and easier for users to find objects in a dictionary or group of objects stored in a containing data structure.

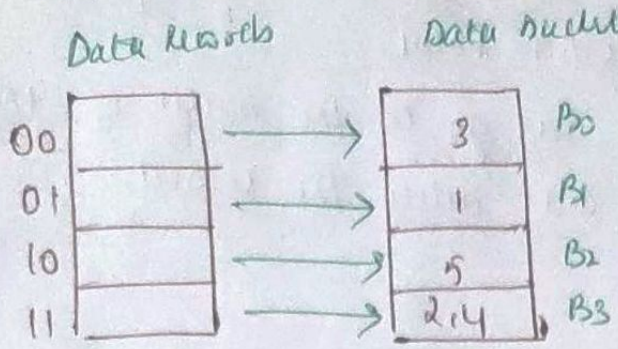
It maximizes the available space for objects, tables, and other data within a system.

Example:

Key	Hash Index
1	11001
2	11011
3	01000
4	10011
5	00010

If the global depth 'k' value is '2' then we can select 2 LSB's of the hash index.

Therefore the obtained values are 01, 11, 00, 11, 10.



The above represents the insertion of the records respectively

Terminologies:

Directories: The containers that store the pointers to the buckets Each bucket has an 'id' associated with them.

Global depth: It can be defined as the number of bits in directory id.

Operations in Dynamic hashing:

Search: (i) Calculate the hash address of the record based on the primary key.

(ii) Convert the hash address into binary form.

(eg) 13 → hash address.

1101 → binary form.

(iii) Select k LSBs from the binary form, where k is the global depth of the directory.

(iv) if k=3, then we select 101.

(v) Now we can navigate to the bucket with directory id 101 and search the record.

Insertion: (i) Same procedure as in search is followed until selecting the k LSB's.

(ii) Now go to the bucket with obtained address and insert the record.

Deletion: (i) Again the same procedure is followed until we reach the k LSB's value.

(ii) Now we can go to the bucket of that particular addrss value and delete the record.

Advantages:

* Performance doesnot come down, it just simply increases the memory.
* The memory is well utilized.
* Good for dynamic databases.

Disadvantages:

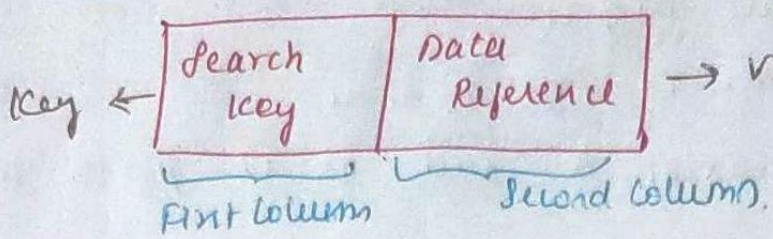
* Maintaing the bucket address table becomes tedious
* Bucket overflow can occur.

Indexing :→ It is a technique that uses data structure to optimize the searching time of a database query.

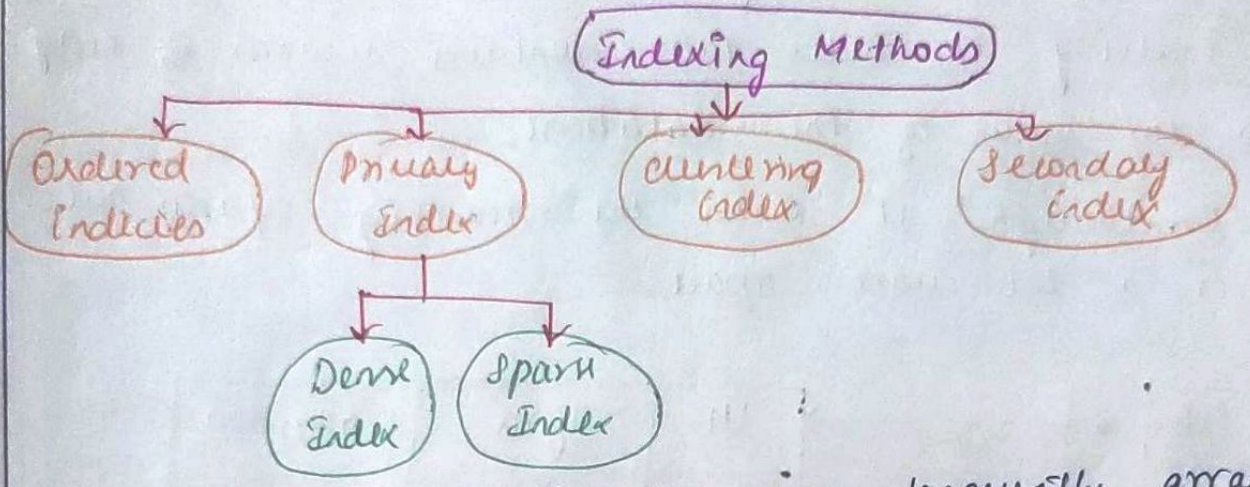
→ It is used for improving database performance by reducing the number of disk accesses. necessary when a query is run.

→ It is considered to having a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. The second column contains set of pointers for holding the address of the disk block where a specific value is stored.

Structure of Index:



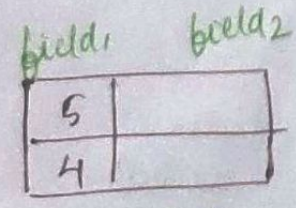
Types of Indexing:



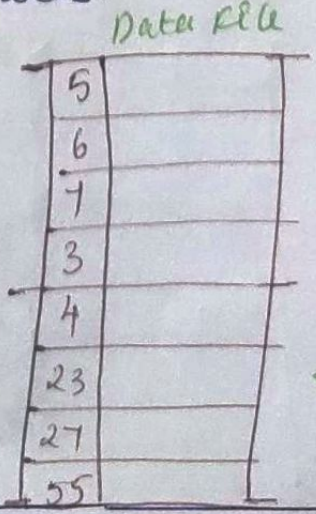
Ordered Indices: The indices are frequently arranged / sorted. Ordered indices are indices that are said to be sorted.

Primary Indexing: It refers to the process of creating an index based on table's primary key.

As the index is comprised of primary key, they are unique, not null and possess one to one relationship with data blocks.



Primary Index



Characteristics of Primary Indexing:

→ Search keys are unique.

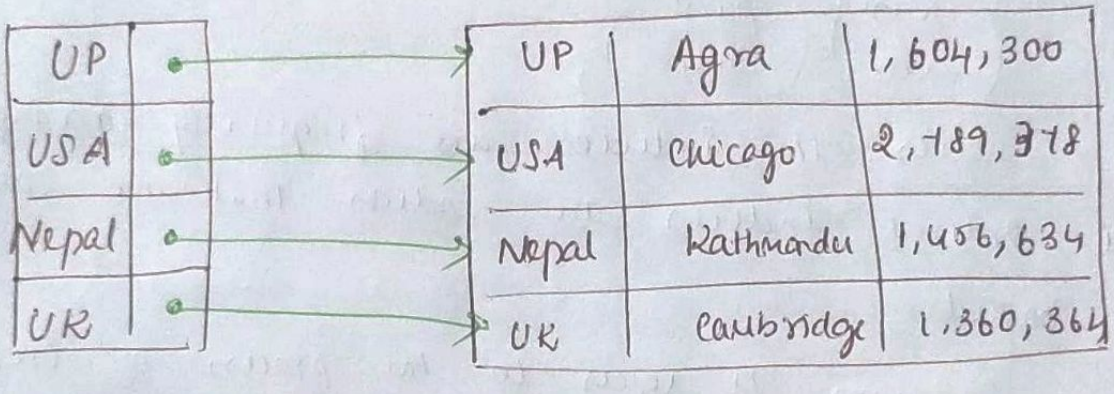
→ Search keys are in sorted order.

→ Search keys cannot be null as it points to a block of data

→ Fast and efficient searching.

Dense Indexing: the index table contains records of every search key value of the database.

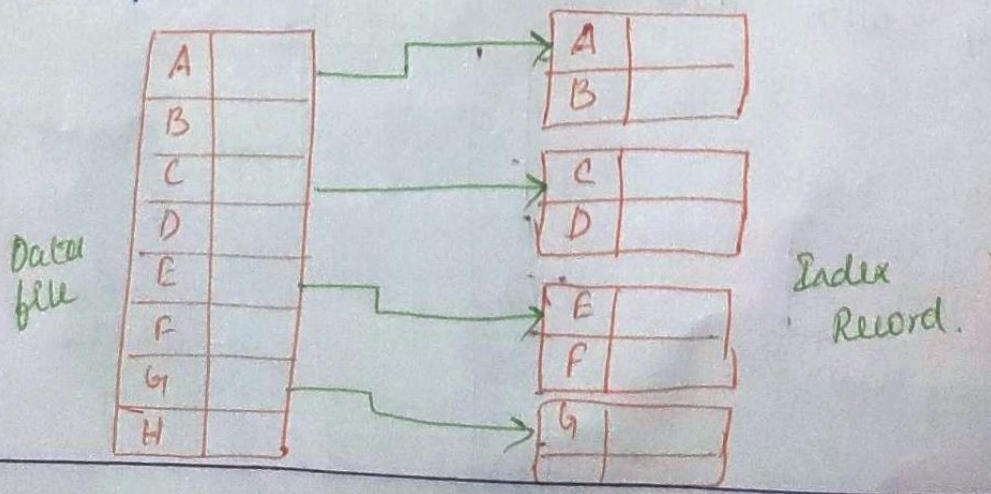
Eventhough it makes the searching faster and requires a lot more space.



Sparse Indexing: consumes lesser space than dense indexing.

We don't include a search key for record despite that we store a search key that points to a block.

The pointed block further contains a group of data.



Secondary Indexing

When using sparse indexing, the size of the mapping grows along with the size of the table.

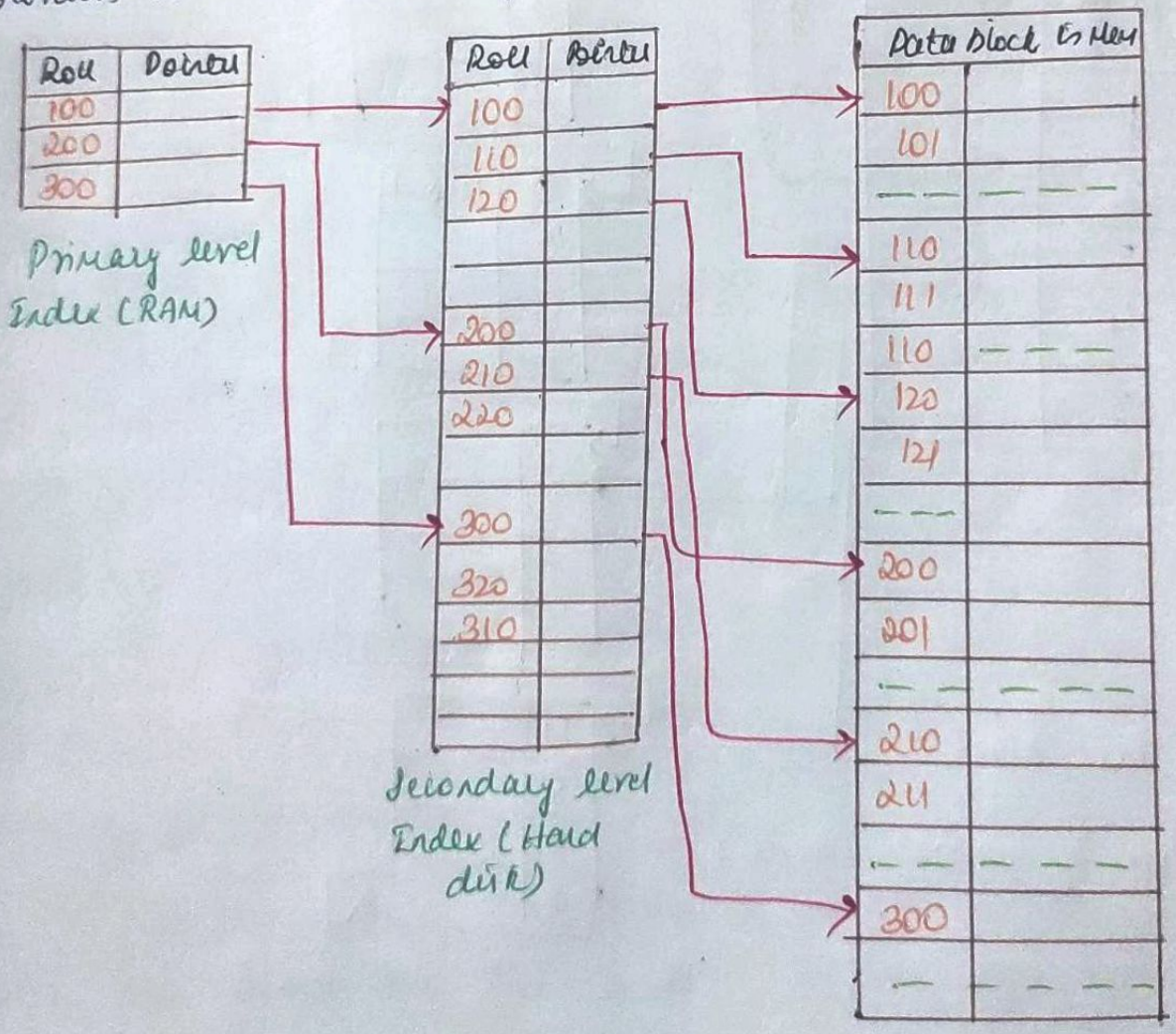
The mappings are frequently stored in the primary memory to speed up address fetching.

The secondary memory then searches the actual data using the address obtained through mapping.

Due to the above reasons we introduce secondary indexing to solve the issue.

It helps in reducing the size of the ~~mapping~~ mapping.

Massive range for the columns is chosen first so the results is small mapping.

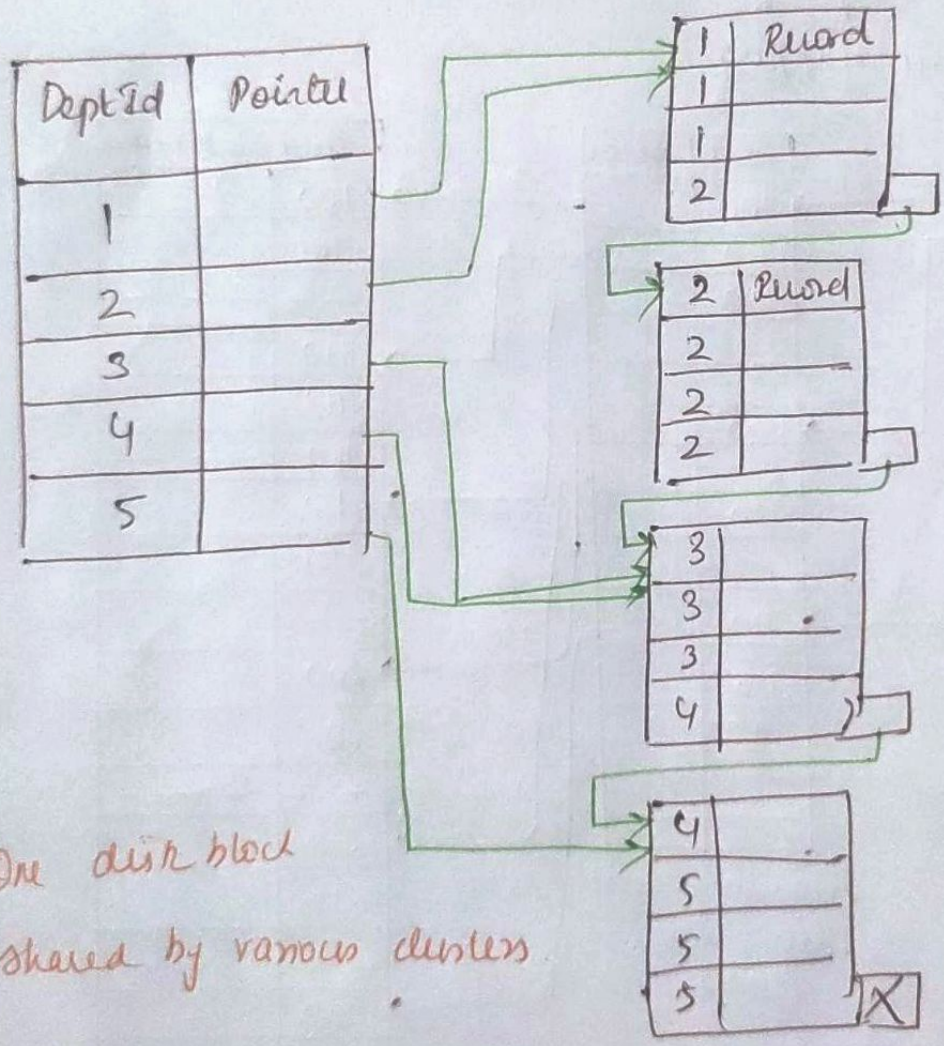


Clustering Index:

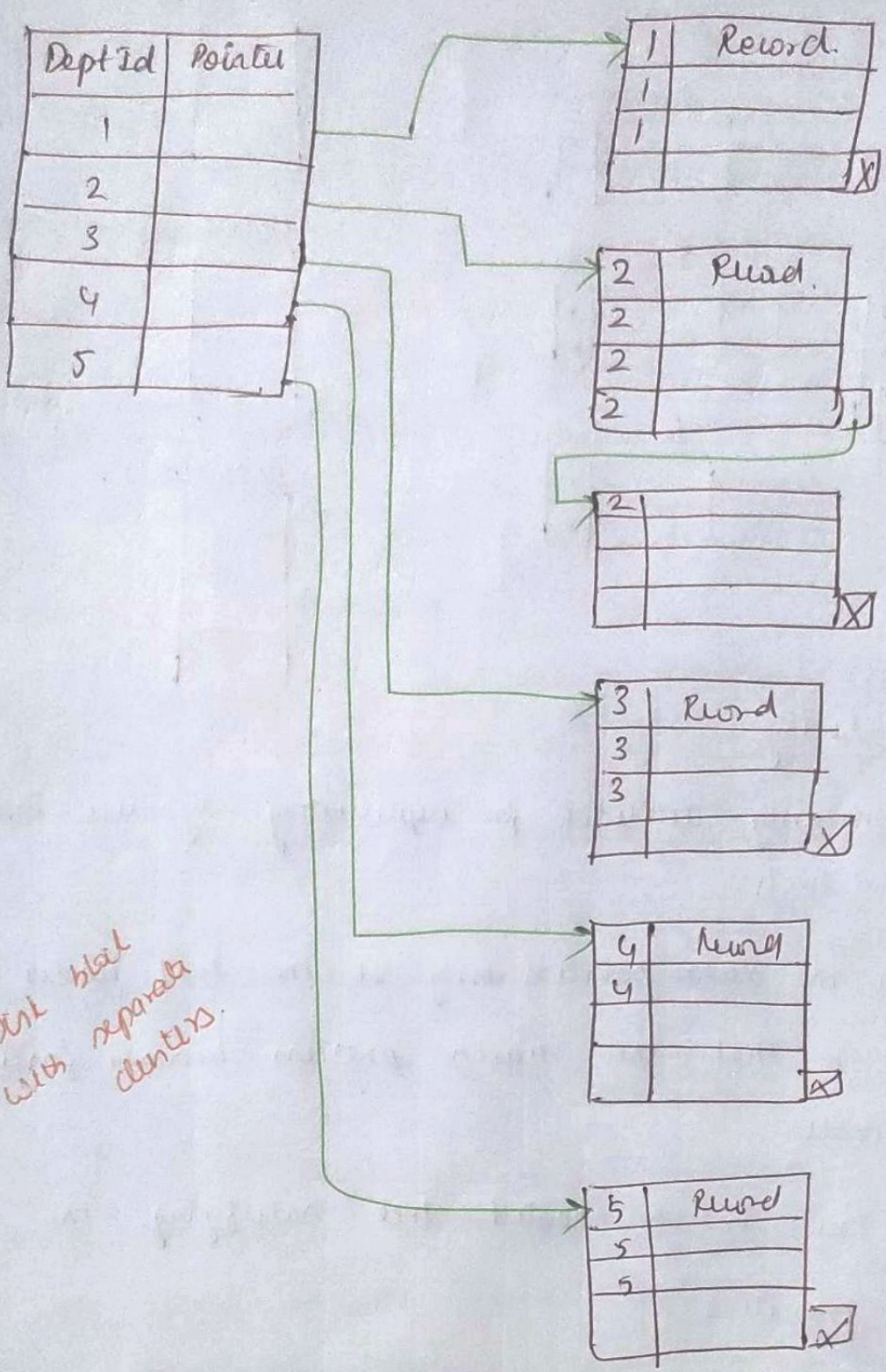
An ordered data file can be defined as a clustered index. Non-primary key columns, which may or may not be unique for each record, are sometimes used to build indices.

In this, 2 or more columns are joined to acquire the unique value and generate an index out of them to take it easier to find the record.

Records with comparable properties are grouped together, and indicies for these groups are connected.

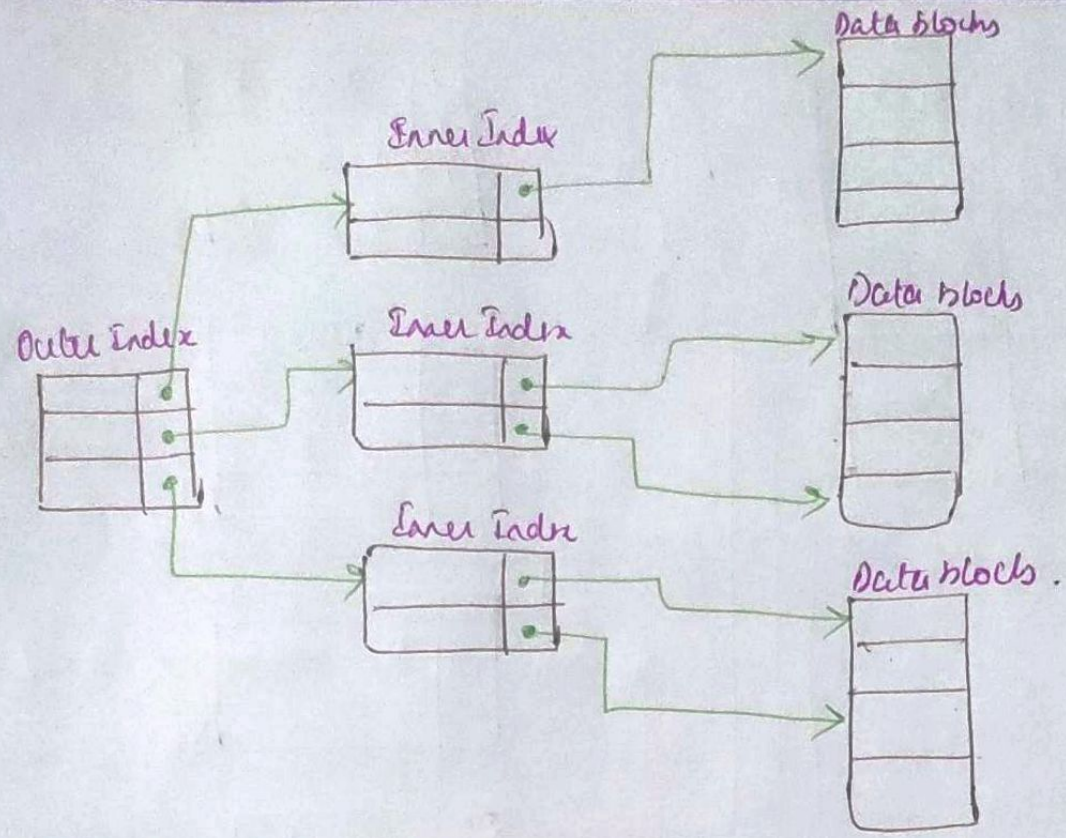


One disk block shared by various clusters.



Multilevel Indexing: It is created when a primary index doesnot fit in memory.

It can reduce the number of disk access to short any record and kept on disk as a sequential file and create a spare base on that file.



B+ Tree Indexing:

* It is primarily utilized for implementing dynamic indexing on multiple levels.

* It stores the data pointers only at the leaf nodes of the tree. so that the search process becomes faster and accurate.

* A B+ Tree is a rooted tree satisfying the following properties.

→ All paths from root to leaf are of the same length (takes the form of balanced tree).

→ Each node is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.

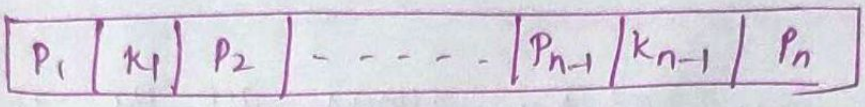
→ A leaf node has between $\lceil (n-1)/2 \rceil$ and n-1 values

→ Special cases

. If the root is not a leaf, it has atleast 2 children

. If the root is leaf (that is, there are no other nodes in the tree), it can have between 0 and n-1 values.

B+ Tree Node structure:



- Ki are the search key values.
- Pi are pointers to children (for non leaf nodes) or pointers to records or buckets of record (for leaf nodes)

. There are upto n-1 search key values and n pointers.

. The search keys in a node are ordered,

$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

Rules for B+ Tree:

→ Leaves are used to store data records

→ It is stored in the internal nodes of the tree.

→ If a target key value is less than the internal node, then the point just to its left side is followed.

→ If a target value is greater than equal to the internal node, then the point just to its right side is followed.

→ The root has a minimum of 2 children.

The Main goal of B+ tree is:

Sorted Intermediary and leaf nodes: Since it is a balanced tree, all nodes should be sorted.

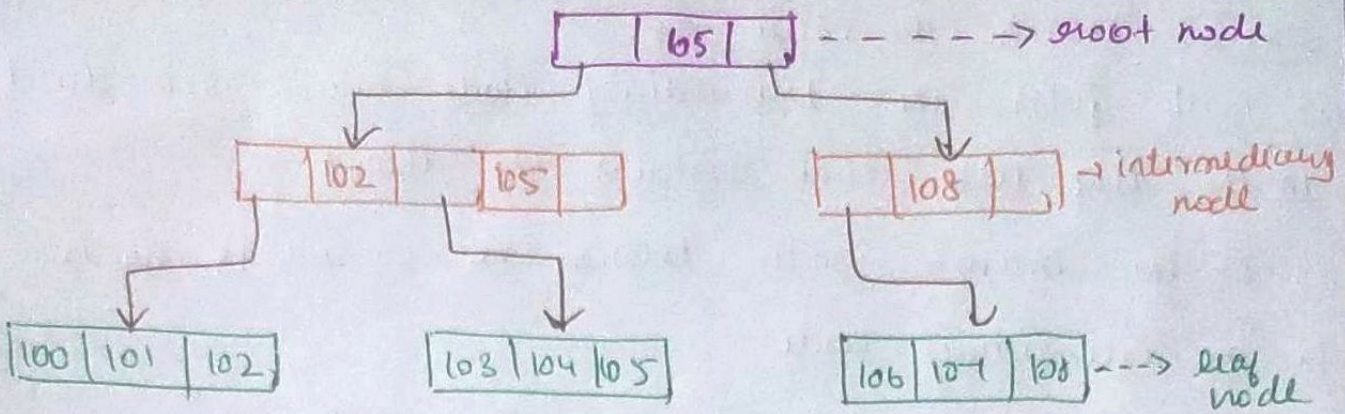
Fast traversal and Quick search: One should be able to traverse through nodes very fast. (ii) we should be able to pass through the intermediary node very easily.

No overflow pages: B+ tree allows all the intermediary and leaf nodes to be partially filled.

Example:

Student		
Student_Id	Student_Name	Address.
100	Joseph	Alaiden Township
101	Allen	Avon Township
102	Chris	Clinton Township
103	Patty	Troy
104	Jack	Avon Township
105	Jenica	Clinton township
106	James	Troy.
107	Antony	Alaiden township
108	Jacob	Troy.

Consider the above table to shown in B+ tree structure.



→ If a Btree has an order of n, where n, is the number of branches. the above has 5 branches so order n=5.

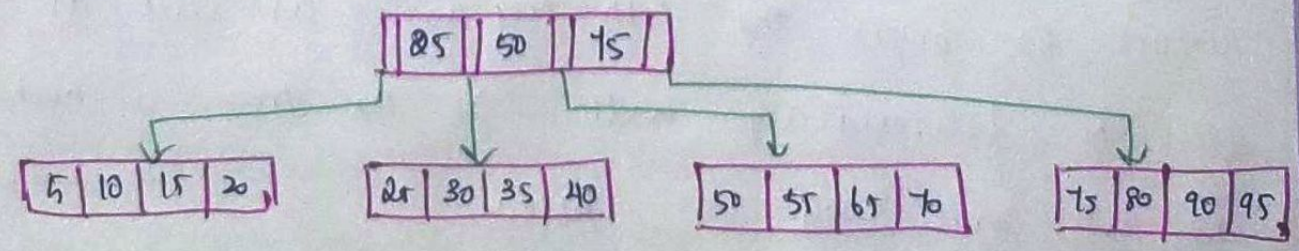
Then it can have n/2 to n intermediary nodes and n/2 to n-1 leaf nodes.

n = 5

Nodes	Min Node	Max Node	Min Node	Max Node
Root	1		1	
Internal Nodes	n/2	n	3	5
Leaf Nodes	n/2	n-1	3	4

~~Ex~~ For n=5 (it has 5 branches from root), then it can have 3 to 5 intermediary nodes and 3 to 4 leaf nodes.

Operations in Btree:



Search: Search the value 65.

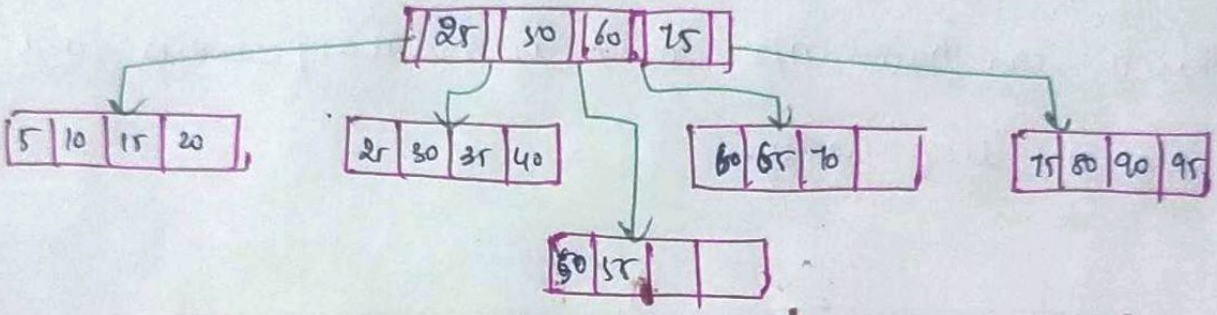
(i) First fetch the intermediary node which will direct to the leaf node that contains the record.

(ii) So the branch nodes between 50 and 75 nodes in the intermediary node.

(iii) Then it is directed to the leaf node.

Note: No insert / update / delete is allowed during search.

Insert: Insert a record value 60.



First it goes to leaf node to insert, so that the current root node is 50.

Inorder to balance we split the leaf node.

So we split it up as (50, 55) and (60, 65, 70)

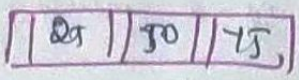
Now the intermediary not cannot branch from 50.

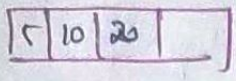
We insert 60 and the we branch out.

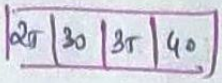
Deletion: Delete 60.

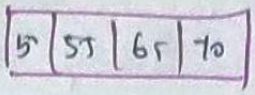
Remove 60 from the 4th leaf node as well as from the intermediary node. So the tree is back to the same way as before.

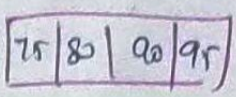
Delete 15. It can just simply traverse to the 1st leaf node and delete it.







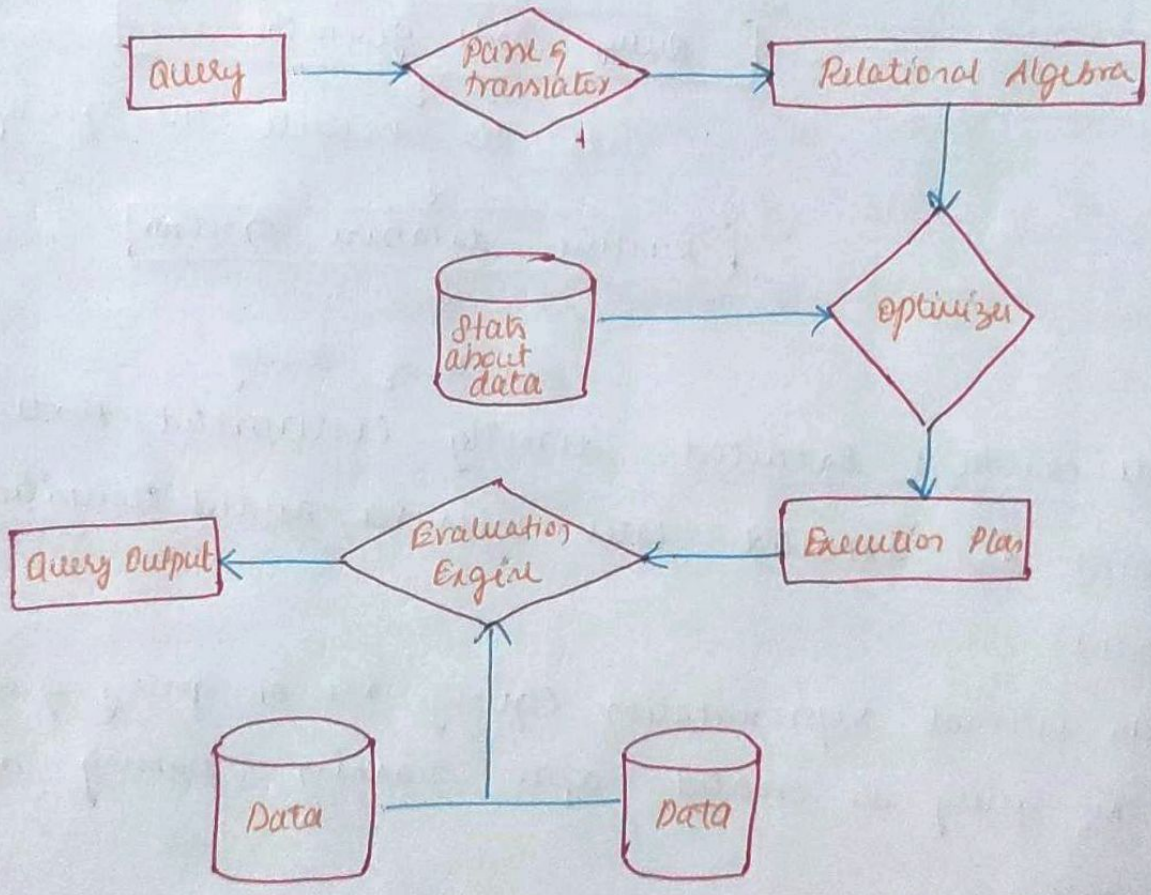




Query Processing:

Query Processing: Query processing refers to activities including translation of high level languages. (HLL) queries into operations at physical file level, query optimization transformations and actual evaluation of queries.

Steps in Query processing:

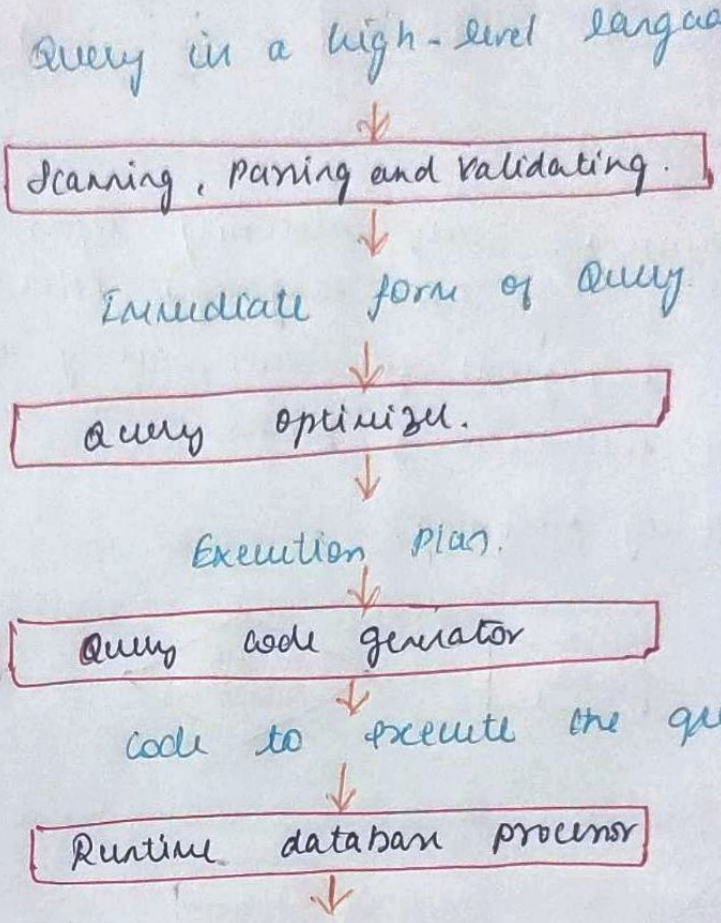


Function of query parser is parsing and translation of HLL query into its immediate form relation algebric expression. A parse tree of the query is constructed and then translated.

The activities involved in parsing, validating, executing and optimizing a query is called Query Processing.

Two Internal representations of a query:

(i) Query Tree (ii) Query Graph.

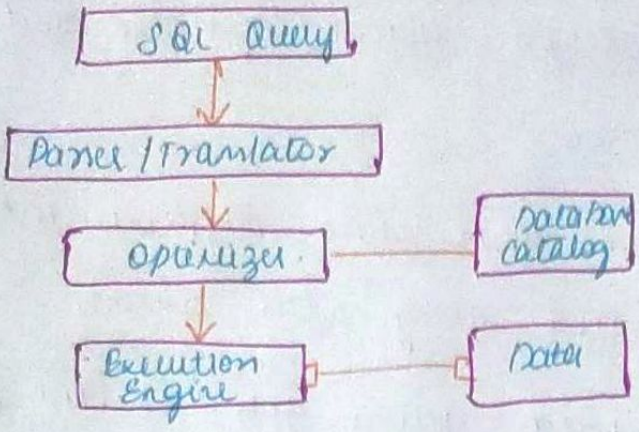


Code can be: Executed directly (interpreted mode) stored and executed later whenever needed (compiled mode)

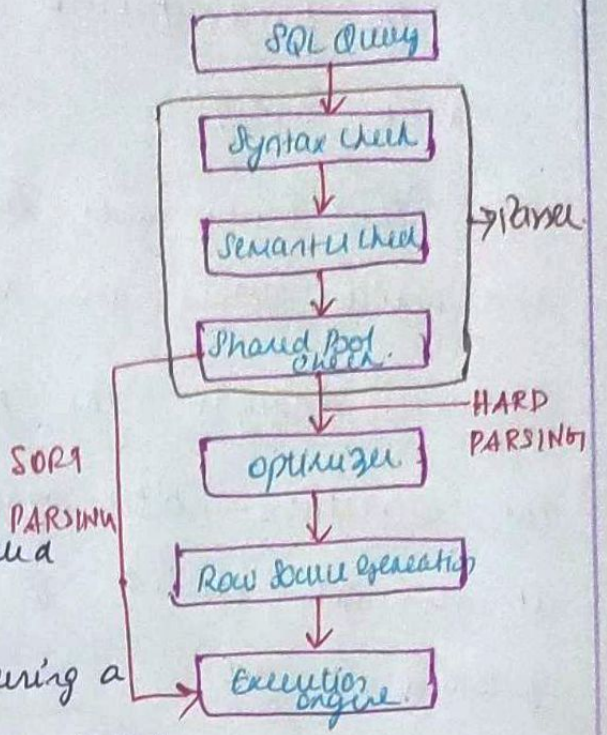
→ An internal representation (query tree or query graph) of the query is created after scanning, parsing and validating.

→ Then DBMS must devise an execution strategy for retrieving the result from the database files.

Block Diagram of Query Procesing:



Detailed Diagram is drawn as:



Query tree: An internal representation of the query is thus created, usually as a tree data structure called a query tree.

Query graph: To represent the query using a graph data structure called a query graph.

Query optimization: A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization.

Query processing can be divided into 2 phases.

* Compile time
* Run time

Compile Time phases:

→ Parsing and Translation (Query Compication)

→ Query optimization

→ Evaluation (code generation).

Parsing and translation:

The given query undergo lexical, syntatic and semantic analysis.

Lexical Analysis: The Query gets broken down into different tokens and white spaces are removed along with comments.

Syntactic Analysis: The processed query gets checked for the correctness, both syntax and semantic wise. and it also gets checked for the SQL rules whether it is followed or not.

Semantic Analysis: The query processor checks if the meaning of the query is right or not, and also the queries mentioned in the tables are present in the DB or not.

Example: SELECT emp_name.

FROM employee.

WHERE

salary >10000;

The above query will be divided into following tokens;

SELECT	WHERE
emp_name	salary
FROM	>
employee	10000

the tokens get validated for,

(i) the name is looked into data dictionary table.

(ii) the name of the columns mentioned (emp_name and salary) in the tokens are validated for existence.

(iii) the type of columns being compared to have the same type.

Translate: Translate the generated set of tokens into a relational algebra query.

Example: Π emp_name (σ salary > 10000)

Query Evaluation:

Once the translation is complete the next process is to apply certain rules and algorithms to generate a few other powerful and efficient data structures.

Example: If the relational graph was constructed, there could be multiple paths from source to destination. Hence the execution plan will be generated for each of the paths.

Query optimization: The aim here is to minimize the query evaluation time.

It evaluates the usage of index present in the table and the columns being used.

Some of the factors weighed is by the optimizer to calculate the cost of a query evaluation plan.

- CPU Time
- Number of ~~tuples~~ tuples to be scanned.
- Disk accen time.
- Number of operations.

Runtime phases:

It includes the execute and evaluate for the query generated in the compile time phase.

Query optimization:

The major purposes of SQL query optimization are:

1) Reduce Response Time: It helps to enhance performance by reducing the response time. The time difference between user requesting data and getting response should be minimized for a better user experience.

2) Reduced CPU execution time: The CPU execution time of a query must be reduced so that faster results can be obtained.

3) Improved Throughput: The number of resources to be accessed to fetch all necessary data should be minimized. The number of rows to be fetched is a

particular query should be in the most efficient manner such that the least number of resources are used.

Metrics involved for calculating the cost of the query:

1) Execution time: Important metric to analyze the cost query performance in the execution time of the query.

It is defined as the time taken by the query to return the rows from the database.

Example: SET STATISTICS TIME ON

SELECT * FROM SalesLT.Customer;

SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 1 ms.

SQL Server Execution time:

CPU time = 0 ms elapsed time = 0 ms.

parse time, compile time, execution time and also completion time of the query is shown.

2) Statistics IO: It is the major time spent accessing the memory buffer for reading operations in case of query.

We get the number of physical and logical reads performed to execute the query.

Logical Reads: Number of reads that well performed from the buffer cache.

Physical Reads: Number of reads that were performed from the storage device or they were not available in the cache.

3. Execution Plan: It is a detailed step by step processing plan by the optimizer to fetch the rows.

It can be enabled in the database

It helps us to analyze the major phases in the execution of a query.

Example: SELECT p.Name, color, ListPrice From SalesCI.Product p

INNER JOIN SalesCI.ProductCategory pc

ON p.ProductCategoryID = pc.ProductCategoryID;

Approaches to Query Optimization:

(i) Cost Based (Physical):

It is based on the cost of the query.

They can use different paths based on indexes, constraints, sorting methods etc.

It mainly uses the record size, number of

records per block, number of blocks, table size, whether whole table fits in a block etc.

Ex. Dynamic programming.

2) Heuristic Based Optimization:

It is known as the rule based optimization. It is based on the equivalence rule on relational expressions; here the number of combination of queries gets reduces here.

It creats relational tree for the given query based on the equivalence rules.

Rules:

→ Perform all selection operation as early as possible.

→ Perform all projection as early as possible

→ Perform most restrictive joins and selection operations.

→ Sometimes we can combine the heuristic steps with cost based optimization technique to get better results.

Algorithms:

Algorithms for External sorting:

Refer to the sorting algorithms that are suitable for large files of record stored on disk that do not fit entirely in main memory.

Sort-Merge strategy: Starts by sorting small subfiles (runs) of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.

Algorithms for SELECT:

S1 Linear Search (brute force:) Retrieve every record in the file, and test whether its attribute value satisfy the selection condition.

S2 Binary Search: If the selection condition involves an equality comparison on a key attribute on which the file is ordered.

S3 Single Record Search: If the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key), use the primary index to ~~ca~~ retrieve the record.

S4 Multiple Search (Primary index): If the comparison condition is >, ≥, <, or ≤ on a key field

with a primary index, use the index to find the record satisfying the corresponding equality condition. then retrieve all subsequent records.

S5 Multiple search (clustering index): If the selection conditions involve an equality comparison on a non key attribute with a clustering index, to retrieve all the records satisfying the selection condition.

S6 Range query (B+ tree): To use and to retrieve records on condition involving >, >=, <, or <=

Algorithms for JOIN

JOIN (EQUIJOIN, NATURAL JOIN)

2 way join → involves joining of 2 files.

Multiway join → involves joining of more than 2 files

J1 Nested-loop join (brute force):

For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the 2 records satisfy the join condition t[A] = s[B].

Single loop join

Using an access structure to retrieve the matching records.

Sort-Merge Join: The records of each file are scanned only once each for matching with the other file. - unless both A and B are non key attributes, in which case the method needs to be modified slightly.

Algorithms - for PROJECT and SET

The Algorithms used for project can be sorting and Hashing to remove duplicates

Extract all the tuples from the table with only the values for the attributes in attribute list.

If the attribute list doesnot include a key of relation, the duplicate tuples must be removed

Algorithms for SET:

UNION: Sort the 2 relations on the same attributes. Scan and merge both sorted files concurrently, whenever the same tuple exists in both, only one is kept in merged results.

INTERJECTION: The merged results are kept only those tuples that appear in both relations.

SET DIFFERENCE: Keep in the merged results only those tuples that appear in relation R but not in S.