Unit-5

File Organization and Index Structure: File & Record Concept, Placing file records on Disk, Types of Records, Types of Single-Level Index, Multilevel Indexes, Dynamic Multilevel Indexes using B tree and B+ tree . Mongo DB, NoSQL types, Features and tools.

File Organization in DBMS

A database consists of a huge amount of data. The data is grouped within a table in RDBMS, and each table has related records. A user can see that the data is stored in the form of tables, but in actuality, this huge amount of data is stored in physical memory in the form of files.

What is a File?

A file is named a collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes, and optical disks.

What is File Organization?

File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record. In simple terms, Storing the files in a certain order is called File Organization. **File Structure** refers to the format of the label and data blocks and of any logical control record.

The Objective of File Organization

- It helps in the faster selection of records i.e. it makes the process faster.
- Different Operations like inserting, deleting, and updating different records are faster and easier.
- It prevents us from inserting duplicate records via various operations.
- It helps in storing the records or the data very efficiently at a minimal cost.

Types of File Organizations

Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection. Thus it is all upon the programmer to decide the best-suited file Organization method according to his requirements.

Some types of File Organizations are:

- Sequential File Organization
- Heap File Organization
- Hash File Organization

- B+ Tree File Organization
- Clustered File Organization
- ISAM (Indexed Sequential Access Method)

We will be discussing each of the file Organizations in further sets of this article along with the differences and advantages/ disadvantages of each file Organization method.

Sequential File Organization

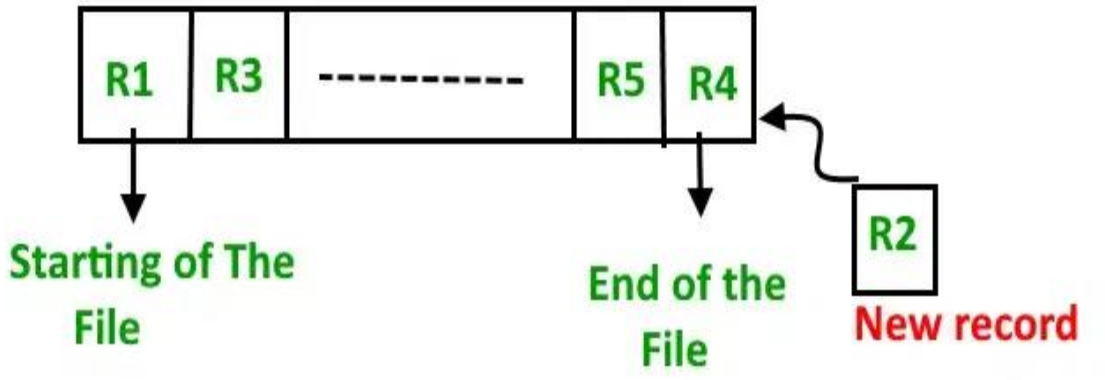
The easiest method for file Organization is the Sequential method. In this method, the file is stored one after another in a sequential manner. There are two ways to implement this method:

1. Pile File Method

This method is quite simple, in which we store the records in a sequence i.e. one after the other in the order in which they are inserted into the tables.

Pile File Method

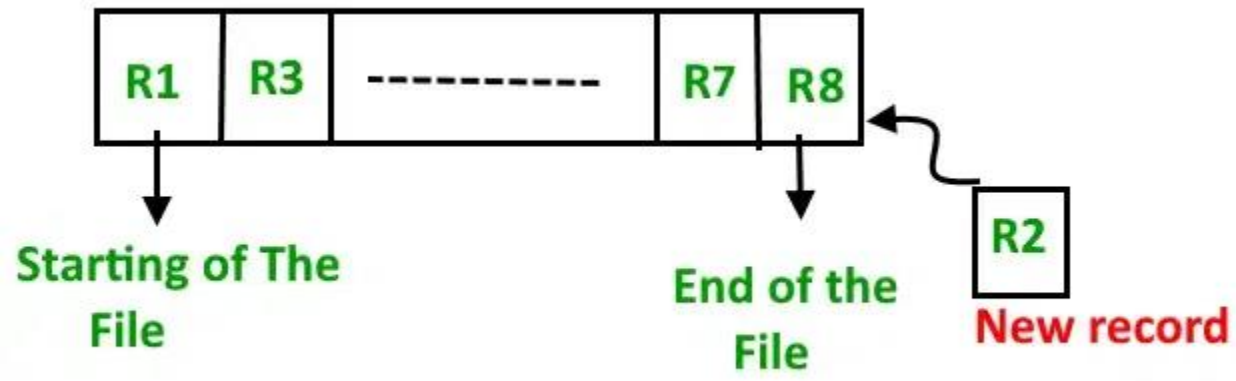
Insertion of the new record: Let the R1, R3, and so on up to R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



New Record Insertion

2. Sorted File Method

In this method, As the name itself suggests whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. The sorting of records may be based on any primary key or any other key.



Sorted File Method

Insertion of the new record: Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on up to R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence.

new Record Insertion

Advantages of Sequential File Organization

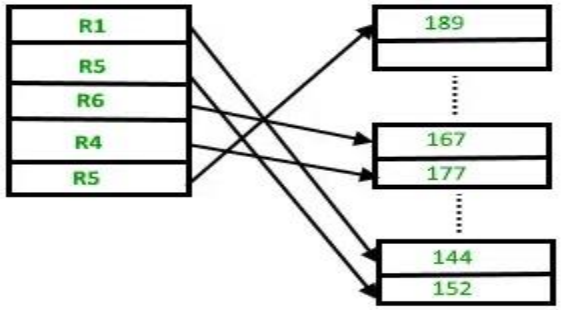
- Fast and efficient method for huge amounts of data.
- Simple design.
- Files can be easily stored inmagnetic tapes i.e. cheaper storage mechanism.

Disadvantages of Sequential File Organization

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- The sorted file method is inefficient as it takes time and space for sorting records.

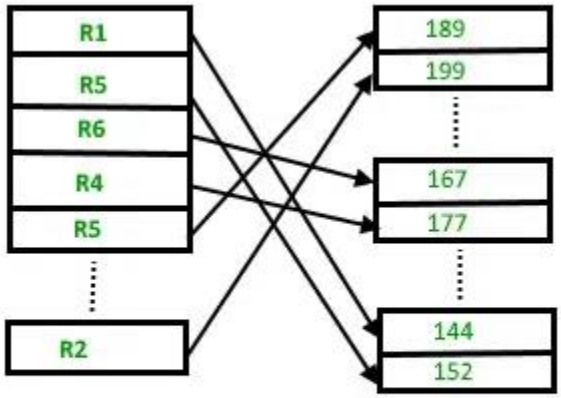
Heap File Organization

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



Heap File Organization

Insertion of the new record: Suppose we have four records in the heap R1, R5, R6, R4, and R3, and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the data blocks selected by the DBMS, let's say data block 1.



New Record Insertion

If we want to search, delete or update data in the heap file Organization we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting, or updating the record will take a lot of time.

Advantages of Heap File Organization

- Fetching and retrieving records is faster than sequential records but only in the case of small databases.
- When there is a huge number of data that needs to be loaded into thedatabase at a time, then this method of file Organization is best suited.

Disadvantages of Heap File Organization

- The problem of unused memory blocks.
- Inefficient for larger databases.

Conclusion

In Conclusion, it is critical to choose the appropriate file organization in a database management system (DBMS). While random structure offers flexibility but may lead to fragmentation, sequential arrangement is better for ordered access. Indexing finds a balance, clustering improves

efficiency for particular queries, and hashed structures maximize speedy access. Sustained efficiency requires routine maintenance. The selection should be based on the requirements of the particular application; for best DBMS performance, a careful combination of strategies may be required.

Certainly! Let's delve into the concept of file records, their placement on disk, and the types of records in a Database Management System (DBMS).

File Record Concept

A **file record** is a collection of related data fields that are treated as a unit. In a DBMS, records are used to store data in a structured format, making it easier to retrieve, update, and manage information.

Placing File Records on Disk

When placing file records on disk, several strategies can be employed to optimize performance and storage efficiency:

1. **Sequential Storage**: Records are stored one after another in a contiguous block of disk space. This method is simple and efficient for sequential access but can lead to fragmentation over time.
2. **Indexed Storage**: An index is created to map record keys to their physical locations on disk. This allows for faster search and retrieval of records but requires additional storage for the index itself.
3. **Hashed Storage**: Records are placed on disk based on a hash function applied to their key values. This method provides quick access but can suffer from collisions, requiring collision resolution techniques.

Types of Records in DBMS

1. **Fixed-Length Records**: Each record has a fixed size, making it easy to calculate the position of any record on disk. This simplifies access but can lead to wasted space if the data doesn't fully utilize the allocated size.
2. **Variable-Length Records**: Records can vary in size, allowing for more efficient use of space. However, this requires additional mechanisms to manage and locate records, such as pointers or delimiters.
3. **Heap Files**: Records are stored in no particular order, and new records are placed in the first available space. This method is simple but can lead to inefficient access patterns.
4. **Sorted Files**: Records are stored in a sorted order based on one or more fields. This facilitates efficient range queries and ordered access but requires maintenance to keep the records sorted.

5. **Clustered Files**: Records that are frequently accessed together are stored close to each other on disk. This improves access times for related records but requires careful planning and maintenance.

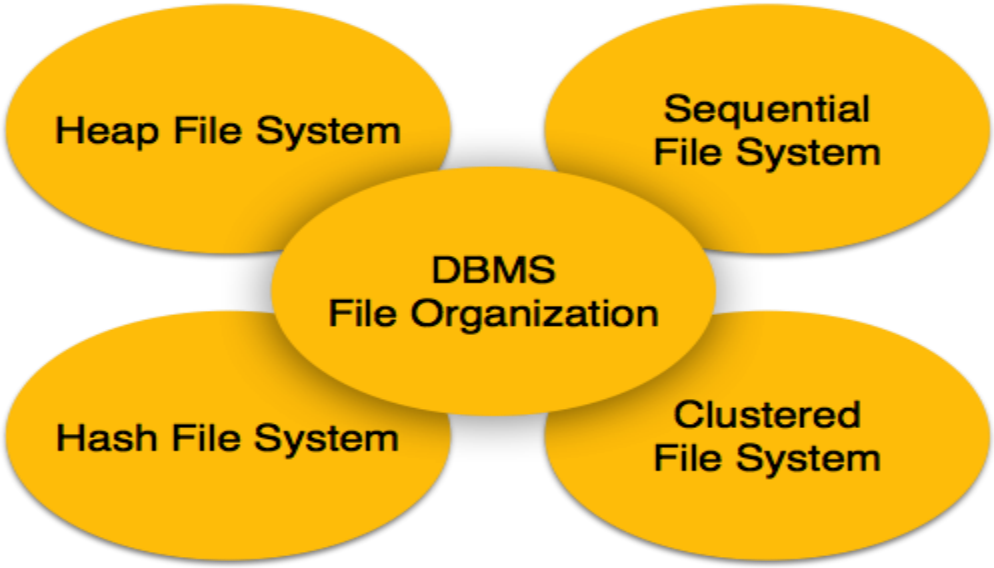
By understanding these concepts, you can better appreciate how data is organized and managed within a DBMS, leading to more efficient database design and operation. If you have any specific questions or need further details, feel free to ask!

DBMS - File Structure

Relative data and information is stored collectively in file formats. A file is a sequence of records stored in binary format. A disk drive is formatted into several blocks that can store records. File records are mapped onto those disk blocks.

File Organization

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records −



Heap File Organization

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

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

Hash File Organization

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

Clustered File Organization

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

File Operations

Operations on database files can be broadly classified into two categories −

- **Update Operations**
- **Retrieval Operations**

Update operations change the data values by insertion, deletion, or update. Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering. In both types of operations, selection plays a significant role. Other than creation and deletion of a file, there could be several operations, which can be done on files.

- **Open** − A file can be opened in one of the two modes, **read mode** or **write mode**. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
- **Locate** − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
- **Read** − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
- **Write** − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.
- **Close** − This is the most important operation from the operating systems point of view. When a request to close a file is generated, the operating system
    - removes all the locks (if in shared mode),
    - saves the data (if altered) to the secondary storage media, and

- releases all the buffers and file handlers associated with the file.

The organization of data inside a file plays a major role here. The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

DBMS - Indexing

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely.

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

Indexing is defined based on its indexing attributes. Indexing can be of the following types −

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

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

Dense Index

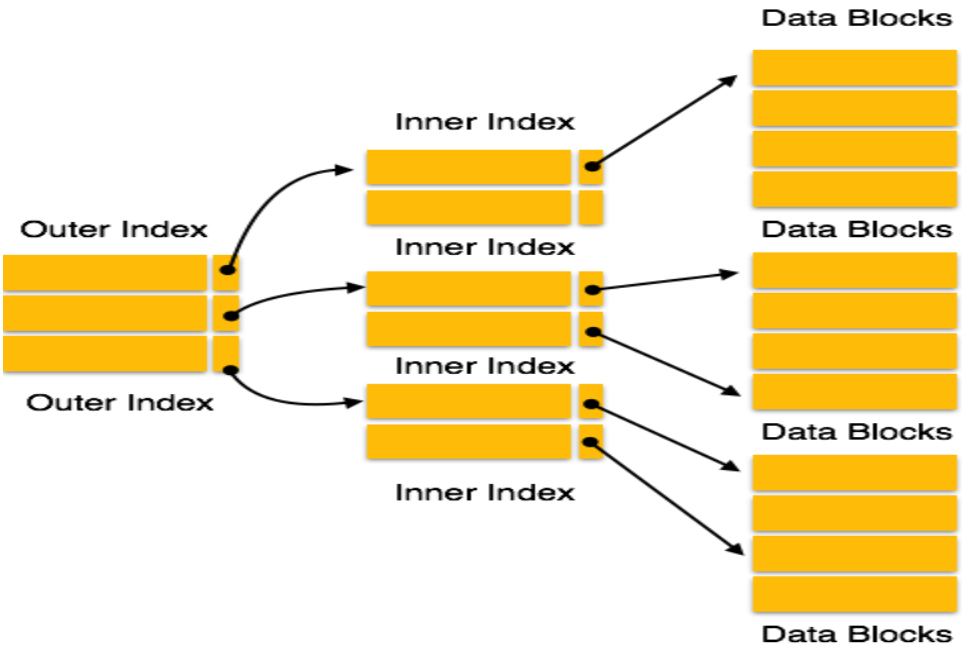
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Multilevel Index

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



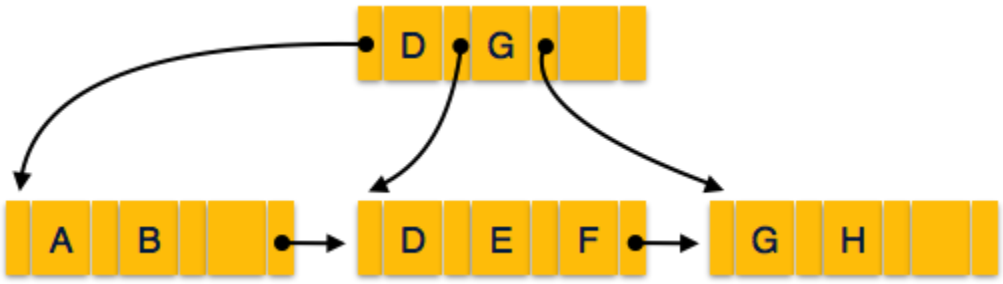
Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B^+ Tree

A B^+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B^+ tree denote actual data pointers. B^+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

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



Internal nodes −

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes −

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ 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 next leaf node and forms a linked list.

B^+ Tree Insertion

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

- If a non-leaf node overflows −
    - Split node into two parts.
    - Partition the node at **i = ⌈(m+1)/2⌉**.
    - Entries up to **i** are kept in one node.
    - Rest of the entries are moved to a new node.

B^+ Tree Deletion

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

MongoDB

MongoDB is a **document-oriented NoSQL database** developed by MongoDB Inc. It stores data in **JSON-like** format called **BSON (Binary JSON)**, making it highly flexible and scalable.

🔹 **Key Features of MongoDB:**

- **Schema-less**: Collections can have documents with different structures.
- **High Performance**: Fast read/write operations.
- **Scalability**: Horizontal scaling via **sharding**.
- **Replication**: Ensures data availability with **replica sets**.
- **Indexing**: Supports various types of indexes (single, compound, geospatial, text, etc.).
- **Aggregation Framework**: Advanced data processing and transformation.

🧩 **Types of NoSQL Databases**

NoSQL databases are non-relational and are mainly categorized into 4 types:

Type	Description	Example DBs
Document Store	Stores data as documents (JSON, BSON)	MongoDB, CouchDB
Key-Value Store	Data stored as key-value pairs	Redis, DynamoDB
Column Store	Data stored in columns rather than rows	Apache Cassandra, HBase
Graph Store	Data stored as nodes and relationships	Neo4j, ArangoDB

✨ Features of NoSQL Databases (General)

- **Flexible schema**
- **Horizontal scaling**
- **High availability and fault tolerance**
- **Optimized for specific data models and access patterns**
- **Efficient for Big Data and real-time web applications**

🛠️ Popular Tools for MongoDB & NoSQL

Tool	Purpose
MongoDB Compass	GUI for MongoDB; explore and visualize data
Robo 3T (formerly Robomongo)	Lightweight GUI for MongoDB
Studio 3T	Advanced IDE with SQL support for MongoDB
Mongoose	ODM for Node.js (MongoDB interaction)
Mongo Shell	Command-line interface for MongoDB
NoSQLBooster	GUI with intelligent shell for MongoDB
Atlas	Cloud database as a service by MongoDB