Unit 6
File organization and indexing

6.1 Disks and storage

6.2 Organization of records into blocks

6.3 File organizations - The sequential and the indexed sequential file organizations

6.4 B+ Tree index

6.5 Hash index

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.

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)**

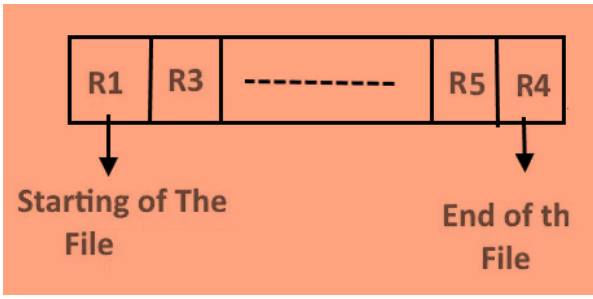
  We will be discussing each of the file Organizations in further sets with the differences and advantages/ disadvantages of each file Organization method.

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:

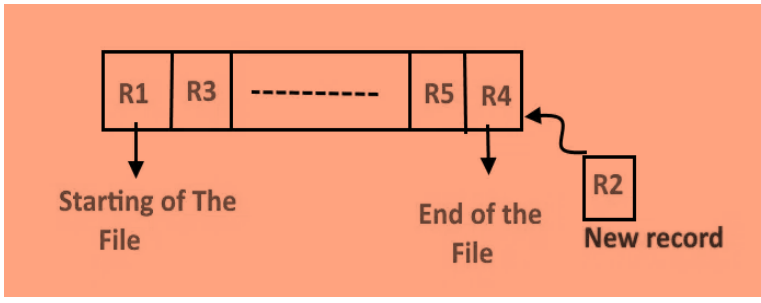
1. Pile File Method

- 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

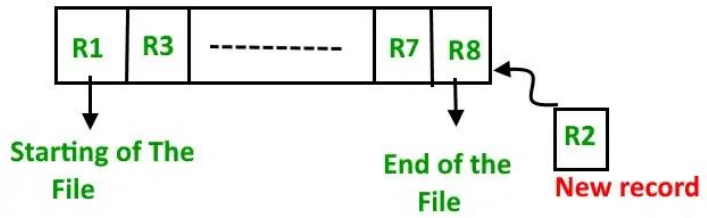
- Insertion of the new record: Let the R1, R3, and so on up to R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



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.



- 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.

Advantages of Sequential File Organization

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

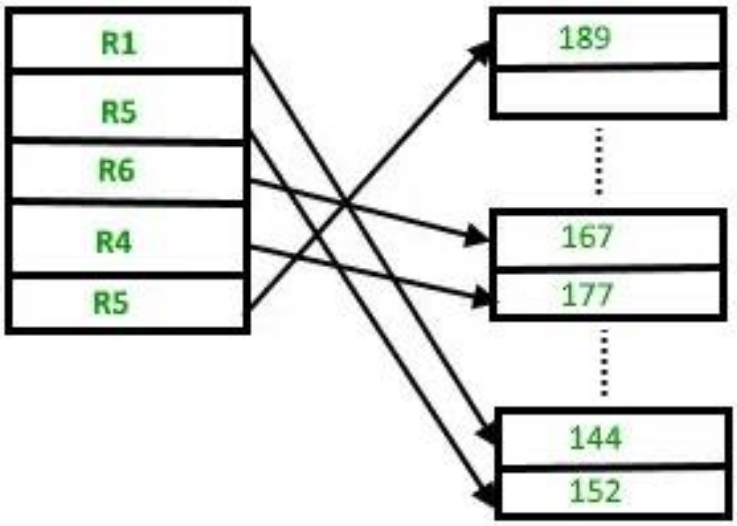
Disadvantages of Sequential File Organization

- 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.

- **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.



- 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.

6.1 Disks and storage

- In computing, **disks** and **storage** refer to the devices and technologies used to store digital data, whether for the operating system, software applications, or user data.
- There are various types of storage devices, each with distinct characteristics related to speed, capacity, and form factor.

Here are the main components and concepts in disks and storage:

1. Types of Storage Devices

- **Hard Disk Drive (HDD)**:
  - A traditional magnetic storage device.
  - Consists of spinning platters coated with magnetic material, where data is stored and retrieved by a read/write head.
  - **Advantages**: Higher capacity and lower cost per gigabyte.
  - **Disadvantages**: Slower speed, sensitive to physical shock, and noisy operation.

- **Solid State Drive (SSD)**:
  - Uses flash memory to store data instead of moving parts.
  - **Advantages**: Faster read/write speeds, lower power consumption, and greater durability.
  - **Disadvantages**: More expensive than HDDs, with lower capacity in some cases.

- **Hybrid Drive (SSHD)**:
  - Combines features of both HDD and SSD.
  - A small SSD cache is used to store frequently accessed data, while the larger HDD provides bulk storage.
  - **Advantages**: Balance between cost and performance.

- **Optical Discs (CD/DVD/Blu-ray)**:
  - Use lasers to read and write data on optical discs.
  - **Advantages**: Portable and often used for media distribution.
  - **Disadvantages**: Limited storage capacity compared to other modern storage types.

- **Flash Drives and External SSDs**:
  - Portable storage devices using NAND flash memory.

- **Advantages**: Highly portable, fast, and durable.
- **Disadvantages**: Expensive relative to traditional HDDs for similar capacities.

2. Storage Technologies

- **NAND Flash**:
  - Used in SSDs, USB flash drives, and memory cards.
  - Offers high-speed data access with no moving parts.

- **RAID (Redundant Array of Independent Disks)**:
  - A data storage virtualization technology combining multiple physical disks into a single logical unit.
  - Types of RAID configurations include:
    - **RAID 0**: Striping (performance boost, no redundancy).
    - **RAID 1**: Mirroring (data redundancy, slower writes).
    - **RAID 5**: Striping with parity (data redundancy with a balance between performance and capacity).
    - **RAID 10**: Combination of RAID 1 and RAID 0 (high performance and redundancy).

- **Cloud Storage**:
  - Online storage managed by a third-party provider.
  - **Examples**: Google Drive, Dropbox, iCloud.
  - **Advantages**: Access from anywhere, scalable, and typically backed by redundancy.
  - **Disadvantages**: Dependent on internet connectivity and security concerns.

3. Storage Performance and Characteristics

- **Speed**: Refers to how quickly data can be read from or written to the storage device. SSDs generally have much higher speed than HDDs.
- **Capacity**: Refers to the amount of data that can be stored on the device, often measured in gigabytes (GB) or terabytes (TB). Larger capacity drives are often used for data-intensive tasks such as video editing, gaming, or large-scale databases.
- **Reliability**: How likely a storage device is to fail or lose data. SSDs tend to be more reliable than HDDs due to the absence of moving parts.

- **Durability**: Related to how well a device withstands physical impact, wear, and tear. SSDs are more durable than HDDs in this aspect.

4. Disk File Systems

- **NTFS (New Technology File System)**:
  - Commonly used in Windows systems.
  - Supports large file sizes, file permissions, and journaling.

- **FAT32 (File Allocation Table 32)**:
  - An older file system used in many portable devices and flash drives.
  - Limited to 4 GB file sizes.

- **exFAT (Extended File Allocation Table)**:
  - Designed for flash drives and external storage devices, providing support for large files (over 4 GB).

- **ext4 (Fourth Extended File System)**:
  - Commonly used in Linux environments.
  - Supports journaling, large file sizes, and extended attributes.

- **APFS (Apple File System)**:
  - Used by Apple devices like macOS and iOS.
  - Designed for high-performance storage, encryption, and snapshot features.

5. Storage Management

- **Partitioning**: The process of dividing a physical disk into multiple logical units, each with its own file system.
- **Formatting**: The process of preparing a partition on a disk to store data by creating a file system on it.
- **Backup and Redundancy**: Techniques to ensure data is protected from loss, often using RAID, cloud services, or external storage devices to create copies of critical data.

- **Data Encryption**: Ensures data on storage devices is protected by converting it into an unreadable format, which can only be decrypted with the proper key.

- Disks and storage technologies form the backbone of modern computing, enabling the efficient storage and retrieval of data.
- The choice of storage device, whether HDD, SSD, or cloud storage, depends on specific needs like performance, capacity, and budget.
- Understanding these devices and their characteristics is crucial for optimizing data management and ensuring system performance.

6.2Organization of records into blocks

- In data storage and database management, organizing records into blocks is crucial for efficient storage, retrieval, and processing of data.
- Records are typically structured pieces of data (such as rows in a table, for example), and blocks (or pages) are the units in which these records are stored on physical storage devices (like hard drives or SSDs).
- The organization of records into blocks can impact the performance, speed, and scalability of data storage systems.

Here's a breakdown of how records are organized into blocks:

1. Concept of a Block

- A **block** (or **page**) is a fixed-size unit of storage that contains one or more records.
- Blocks serve as the basic unit of data exchange between the storage medium and the operating system or database management system (DBMS).
- In most modern file systems or DBMSs, a block typically ranges from **512 bytes** to **4 KB** in size, though larger blocks (up to 8 KB or 16 KB) are also used in some systems.

2. Why Organize Records into Blocks?

Organizing records into blocks has several advantages:

- **Efficiency**: Storing data in blocks reduces the overhead of managing small individual records. It allows for better use of the storage medium and optimizes the read/write operations.
- **Access Optimization**: Data is typically read from storage devices in blocks, so organizing records into blocks minimizes the number of read operations needed to retrieve related records.
- **Disk I/O Optimization**: Since disk I/O operations are typically performed at the block level, organizing records into blocks reduces the number of I/O operations, improving performance.

- **Cache Utilization**: When a block is loaded into memory, it is generally stored as a single unit in the cache. Organizing records into blocks helps take advantage of cache memory, which improves access times for frequently accessed data.

3. Record Organization in Blocks

In a typical database or storage system, **records** are organized in **blocks** in the following ways:

- **Fixed-Length Records**: Each record has a fixed size, and a block contains a number of these fixed-length records. For example, if each record is 100 bytes and the block size is 512 bytes, each block can hold 5 records.

  Advantages:

  - Predictable storage usage and easy management.
  - Simple to calculate how many records will fit in a block.

  Disadvantages:

  - Waste of space if some records are smaller than the fixed size, leading to internal fragmentation.

- **Variable-Length Records**: Records have varying sizes, and blocks contain as many records as they can fit based on their size.

  Advantages:

  - More efficient use of space, as records take up only as much space as they need.

  Disadvantages:

  - More complex management since the system must track variable-length records and handle fragmentation within blocks.
  - Possible wasted space at the end of the block (external fragmentation).

4. Block Structure

Each block typically consists of:

- **Header**: A block header may contain metadata about the block, such as:
  - Block identification (block number or address).
  - The number of records contained in the block.
  - Record offset information (for variable-length records).
  - A checksum or other error-detection information.

- **Data Area**: This is where the actual records are stored. It may contain:
    - For fixed-length records: A contiguous area where each record occupies a predefined space.
    - For variable-length records: A section with pointers and data, where records may be stored in a non-contiguous way, often with pointers linking to overflow or linked blocks if necessary.

5. Types of Block Organization

- **Contiguous Organization**: Records are stored in adjacent locations within the block.
    - This is simple to implement and very fast for reading or writing records sequentially.
    - It's efficient when records have a consistent size and there's minimal need for updates.

- **Linked Organization**: A block contains pointers that link to other blocks (or overflow blocks) that store additional records.
    - This organization is used when records are of variable length or the block is partially full.
    - It can introduce additional overhead when reading records, as the system must follow pointers to get to the next block.

- **Indexed Organization**: A block contains an index or pointers to records, making it easier to access records directly.
    - An index block helps locate records without having to scan all the data sequentially.
    - Typically used for systems that require fast searches or random access to records.

6. Overflow Blocks

When a block reaches its capacity but additional records need to be stored, **overflow blocks** are used. Overflow blocks are additional blocks allocated to store records when the primary block is full. The system must ensure efficient management of overflow blocks to maintain performance and prevent fragmentation.

7. Disk and File System Interaction

When records are organized into blocks, they are stored in disk sectors and accessed by the disk's file system. The file system manages blocks at a lower level, handling allocation, deallocation, and fragmentation. The database or application system interacts with the file system to read and write blocks of records.

8. Performance Considerations

- **Block Size**: The choice of block size can significantly impact system performance. Small blocks reduce internal fragmentation but increase the overhead of managing many blocks. Large blocks may improve throughput but can cause significant internal fragmentation if the records are small.
- **Sequential vs. Random Access**: Sequential access benefits from contiguous block organization, while random access benefits from indexed or linked block organizations.
- **Cache Efficiency**: Larger blocks are more efficient for caching, as more data can be fetched with each read operation.

9. File Systems and Record Organization

- File systems like **NTFS**, **ext4**, and **FAT** organize data into blocks and implement strategies like journaling or indexing to improve data integrity, speed, and reliability.
- Database systems like **MySQL**, **PostgreSQL**, and **Oracle** manage records within blocks as part of their storage engines, often using more sophisticated techniques like indexing, clustering, and transaction logging to optimize performance.

- The organization of records into blocks is a fundamental concept in data storage, file systems, and database management.
- The way records are stored in blocks affects performance, storage efficiency, and retrieval speed.
- Understanding the trade-offs between fixed-length and variable-length records, block size, and organization methods like sequential or indexed access is essential for optimizing storage systems.

6.3 File organizations - The sequential and the indexed sequential file organizations

1. **Sequential File Organization**:
   - **Description**: In sequential file organization, records are stored one after another, usually sorted based on a specific field (key).
   - **Access**: Records are accessed in sequence, either from the beginning to the end (forward) or from the end to the beginning (backward).
     1. **Advantages**:
        1. Simple and inexpensive to implement.
        2. Efficient for bulk processing and report generation.
     2. **Disadvantages**:

1. Slow for random access since the entire file may need to be read to find a specific record.
2. Insertion and deletion of records are slow as the entire file may need to be reorganized.

2. **Indexed Sequential File Organization**:
   - **Description**: Combines sequential organization with an index that allows for faster random access to records.
   - **How it works**:
     - Records are stored sequentially in the file.
     - An index is created to map specific keys to the corresponding records, allowing for quick access.
   - **Access**:
     - A primary index is used to quickly locate a block that contains the record. After finding the block, records are accessed sequentially within the block.

1. **Advantages**:
   1. Supports both sequential and random access.
   2. Insertion and deletion can be done efficiently by updating the index.
2. **Disadvantages**:
   1. Requires additional storage for the index.
   2. May need to reorganize the index periodically.

Use Cases and Considerations

Sequential File Organization:

- **Best suited for**: Applications where data is processed in order (e.g., payroll systems, batch processing).
- **Challenges**: It is not efficient for systems requiring random access, frequent updates, or deletions.

Indexed Sequential File Organization:

- **Best suited for**: Systems that require both **efficient sequential processing** and **fast random access** (e.g., inventory management systems, customer databases).
- **Challenges**: More complex, requiring additional storage space and management of the index.

Comparison of Sequential and Indexed Sequential File Organizations

Feature	Sequential File Organization	Indexed Sequential File Organization
Access Method	Sequential only	Both sequential and random access
Performance for Sequential Access	Very efficient	Efficient but requires index access
Performance for Random Access	Slow	Fast with index
Complexity	Simple	More complex (requires maintaining an index)
Storage Overhead	Minimal	Higher due to the index
Ideal Use Case	Batch processing, log files	Databases, systems requiring both random and sequential access
Insert/Delete Performance	Efficient at the end, but difficult for middle operations	More complex, but can be faster with indexing
Flexibility	Limited to sequential access	More flexible with index for faster searches

6.4 B+ Tree index

A **B+ Tree** is a self-balancing tree data structure that maintains sorted data and allows efficient insertion, deletion, and search operations. It is widely used in database systems and file systems to index large amounts of data. The **B+ Tree** is a variant of the B-Tree and is designed to optimize range queries and sequential access.

- **Structure**:
  - **Internal Nodes**: Contain keys that guide the search path to the leaves.
  - **Leaf Nodes**: Store the actual data or pointers to data records.
- **Properties**:
  - **Balanced**: All leaf nodes are at the same level, ensuring that search, insert, and delete operations are efficient.
  - **Sorted Order**: The keys in the leaves are arranged in sorted order, allowing for efficient range queries.
- **Operations**:
  - **Search**: O(log n) time complexity for searching for a key.
  - **Insert and Delete**: O(log n) time complexity, with the tree being rebalanced as necessary.

- **Advantages**:
    - Efficient for both point queries (specific key) and range queries (range of keys).
    - Supports dynamic inserts and deletes without significant performance degradation.

8. Use Cases of B+ Tree

B+ Trees are particularly useful in scenarios where:

- **Efficient searching** is required for large datasets.
- **Range queries** need to be performed on a sorted set of data.
- **Databases** and **file systems** that need to index large volumes of data (e.g., relational database management systems, key-value stores, and search engines).

Common Applications:

- **Database Indexing**: B+ Trees are widely used to index tables in relational databases, improving the speed of search queries, especially those involving range searches.
- **File Systems**: File systems that manage large datasets often use B+ Trees to organize files and directories for fast lookup and retrieval.
- **Key-Value Stores**: In distributed storage systems like **NoSQL databases**, B+ Trees are used to index key-value pairs.

6.5 Hash index

- **Hash Index**: A type of index where a hash function is used to map a key to a specific location in the index table. Each key is hashed to a unique location where its corresponding record is stored.
    - **Structure**:
        - **Hash Function**: A mathematical function that converts a key into a hash value, which is then mapped to a specific memory location (bucket).
        - **Buckets**: Containers where data records or pointers are stored after hashing.
    - **Operations**:
        - **Search**: O(1) time complexity for direct lookups, assuming a good hash function with minimal collisions.
        - **Insert and Delete**: O(1) average time complexity, but collisions may lead to slower performance.
    - **Collisions**: Occurs when two keys hash to the same location. To handle collisions, techniques like:

        - **Chaining**: Store multiple entries in the same bucket using a linked list.
        - **Open Addressing**: Search for the next available bucket when a collision occurs.
    - **Advantages**:
        - Extremely fast for lookups, particularly when there are few collisions.
        - Efficient for exact-match queries (e.g., finding a specific key).
    - **Disadvantages**:
        - Not suitable for range queries (e.g., finding records within a specific range of values).
        - Collisions can degrade performance if not handled properly.
    - **Use Case**: Hash indexes are ideal for databases where quick lookups of specific keys are required, such as in hash-based databases and key-value stores.

Q/A

Fill in the Blanks (20 Questions)

Multiple Choice Questions (MCQ) (20 Questions)

Short Questions (20 Questions)

Comprehensive Questions (20 Questions)

ANSWER of (MCQ)

ANSWER of Short Questions