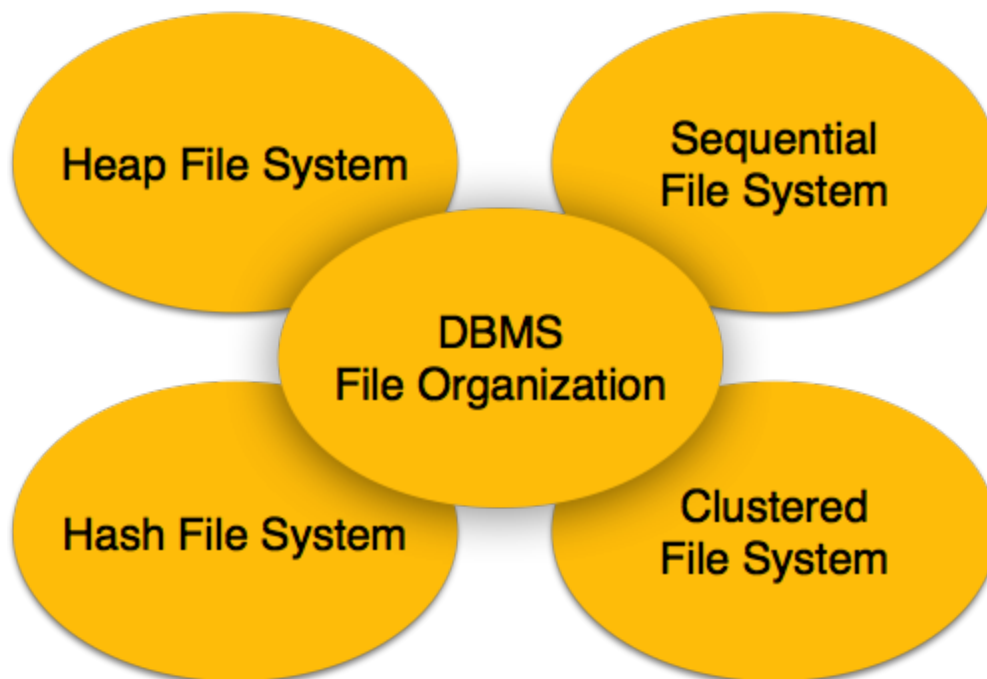


DBMS 4th Unit Notes

| | |
|------------|--------------------------|
| 🕒 Created | @June 1, 2024 5:06 AM |
| 📄 Type | Self Learning |
| ☑ Reviewed | <input type="checkbox"/> |

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

Some types of File Organizations are:

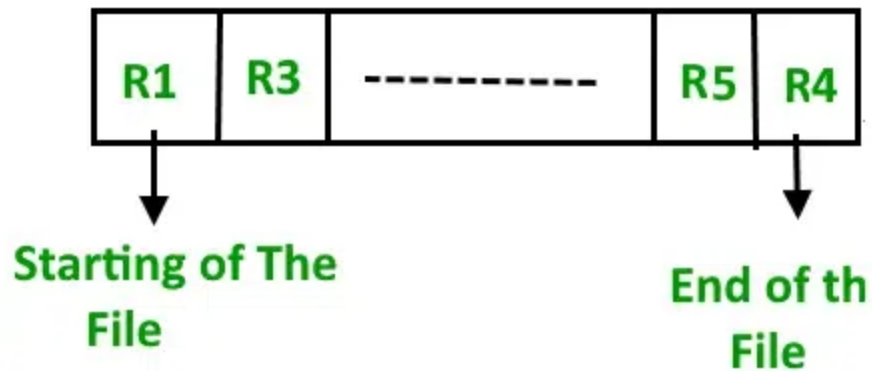
- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Clustered File Organization
- ISAM (Indexed Sequential Access Method)

Sequential File Organization

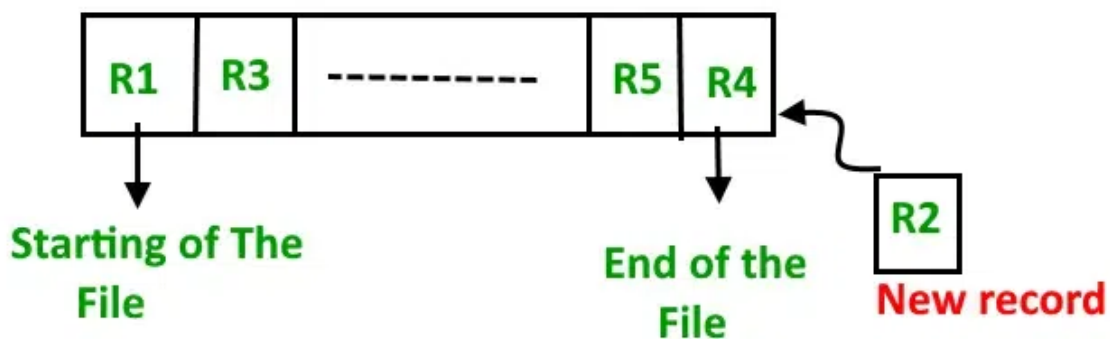
The easiest method for file Organization is the Sequential method. In this method, **the file is stored one after another in a sequential manner.** There are two ways to implement this method:

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.**

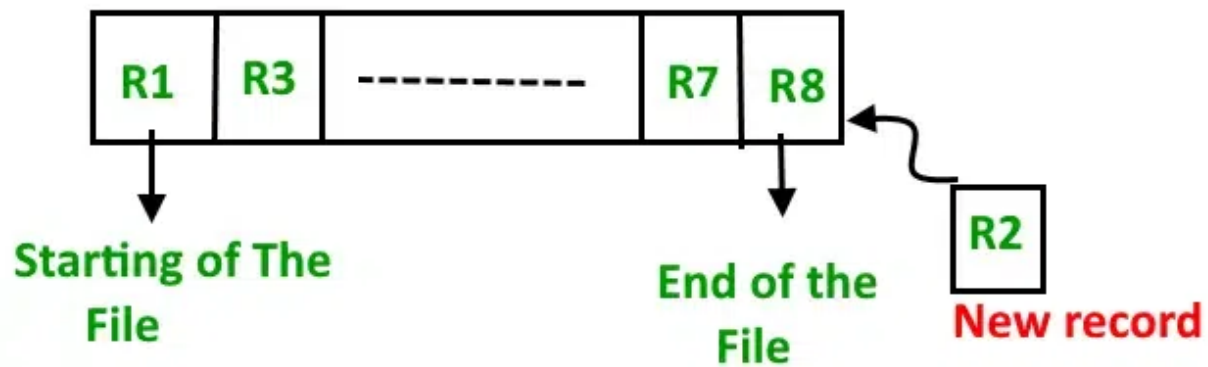


Insertion of the new record: Let the R1, R3, and so on up to R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



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

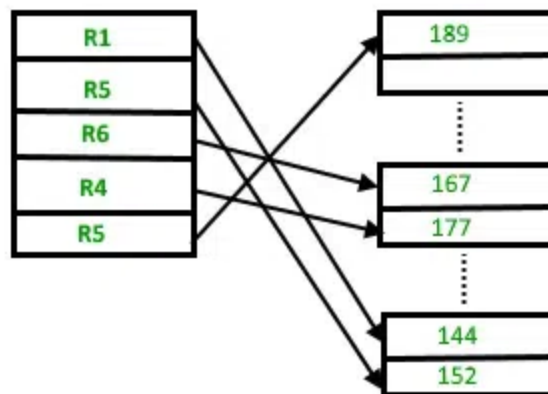
- Fast and efficient method for huge amounts of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e. cheaper storage mechanism.

Disadvantages of Sequential File Organization

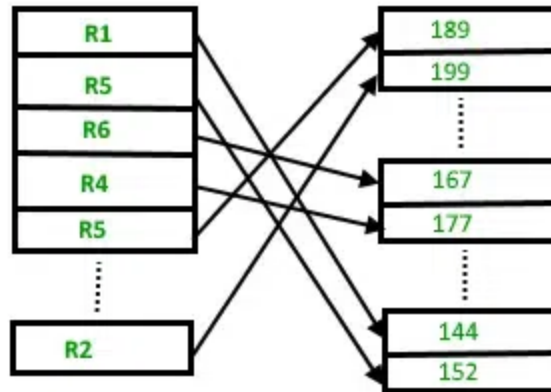
- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- The sorted file method is inefficient as it takes time and space for sorting records.

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 the **database** 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.

Hash File Organization

- **Data bucket** – Data buckets are the memory locations where the records are stored. These buckets are also considered Units of Storage.

- **Hash Function** – The hash function is a mapping function that maps all the sets of search keys to the actual record address. Generally, the hash function uses the primary key to generate the hash index – the address of the data block. The hash function can be a simple mathematical function to any complex mathematical function.
- **Hash Index**–The prefix of an entire hash value is taken as a hash index. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2^n buckets. When all these bits are consumed? then the depth value is increased linearly and twice the buckets are allocated.

Static Hashing

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if we want to generate an address for STUDENT_ID = 104 using a mod (5) **hash function**, it always results in the same bucket address 4. There will not be any changes to the bucket address here. Hence a number of data buckets in the memory for this static hashing remain constant throughout.

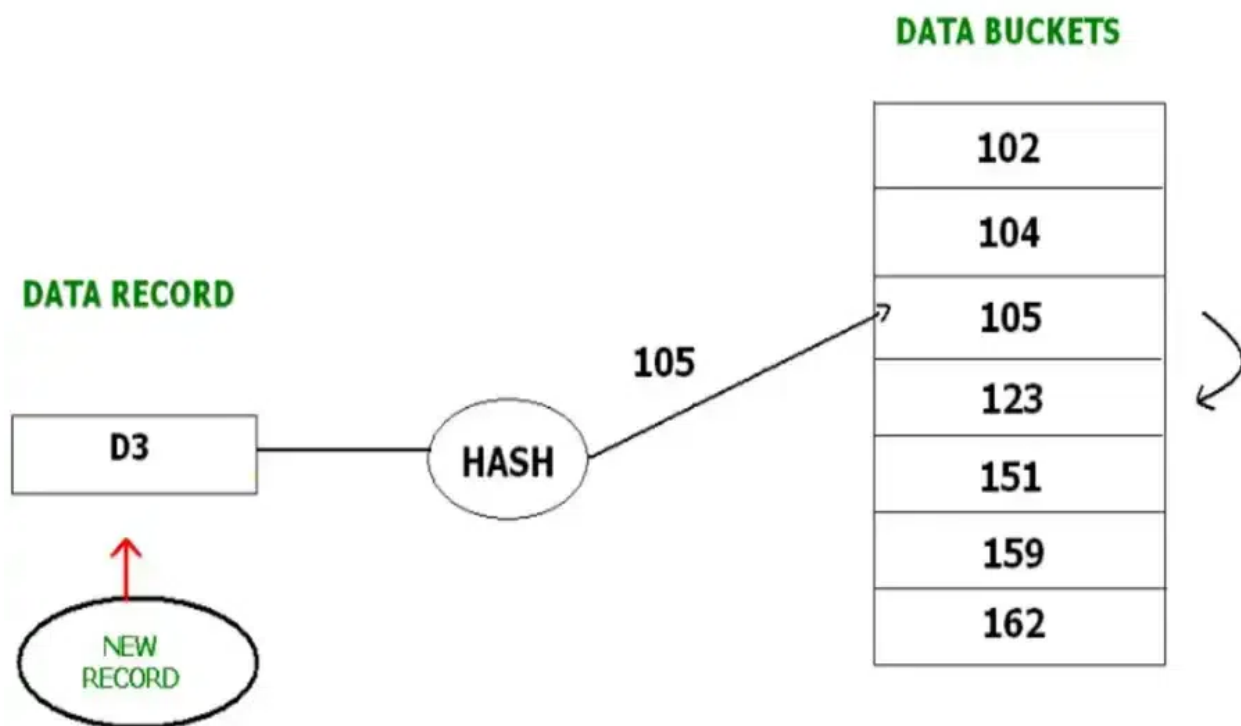
Operations:

- **Insertion** – When a new record is inserted into the table, The hash function h generates a bucket address for the new record based on its hash key K .
Bucket address = $h(K)$
- **Searching** – When a record needs to be searched, The same hash function is used to retrieve the bucket address for the record. For Example, if we want to retrieve the whole record for ID 104, and if the hash function is mod (5) on that ID, the bucket address generated would be 4. Then we will directly got to address 4 and retrieve the whole record for ID 104. Here ID acts as a hash key.
- **Deletion** – If we want to delete a record, Using the hash function we will first fetch the record which is supposed to be deleted. Then we will remove the records for that address in memory.
- **Updation** – The data record that needs to be updated is first searched using the hash function, and then the data record is updated.

Now, If we want to insert some new records into the file But the data bucket address generated by the hash function is not empty or the data already exists in that address. This becomes a critical situation to handle. This situation is static hashing is called **bucket overflow**. How will we insert data in this case? There are several methods provided to overcome this situation.

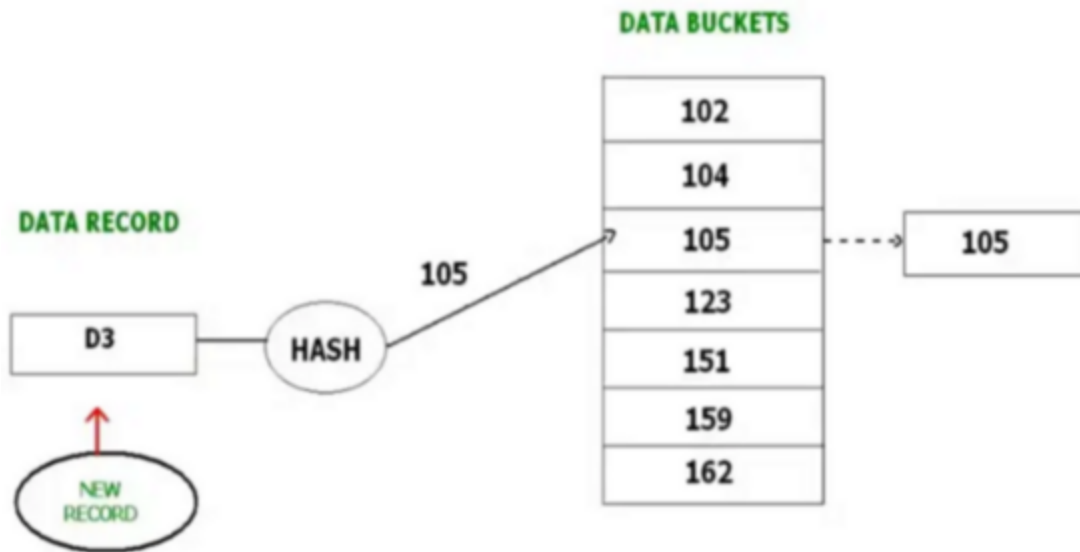
Some commonly used methods are discussed below:

- **Open Hashing** – In the Open hashing method, the next available data block is used to enter the new record, instead of overwriting the older one. This method is also called linear probing. For example, D3 is a new record that needs to be inserted, the hash function generates the address as 105. But it is already full. So the system searches the next available data bucket, 123, and assigns D3 to it.



Closed hashing – In the Closed hashing method, a new data bucket is allocated with the same address and is linked to it after the full data bucket. This method is also known as overflow chaining. For example, we have to insert a new record D3

into the tables. The static hash function generates the data bucket address as 105. But this bucket is full to store the new data. In this case, a new data bucket is added at the end of the 105 data bucket and is linked to it. The new record D3 is inserted into the new bucket.



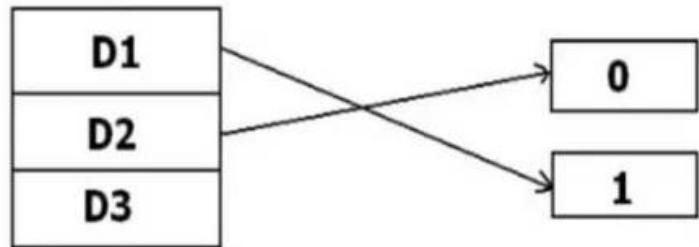
- **Quadratic probing:** Quadratic probing is very much similar to open hashing or linear probing. Here, The only difference between old and new buckets is linear. The quadratic function is used to determine the new bucket address.
- **Double Hashing:** Double Hashing is another method similar to linear probing. Here the difference is fixed as in linear probing, but this fixed difference is calculated by using another hash function. That's why the name is double hashing.

Dynamic Hashing

The drawback of static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. In **Dynamic hashing**, data buckets grow or shrink (added or removed dynamically) as the records increase or decrease. Dynamic hashing is also known as extended hashing. In dynamic hashing, the hash function is made to produce a large number of values. For Example, there are three data records D1, D2, and D3. The hash function

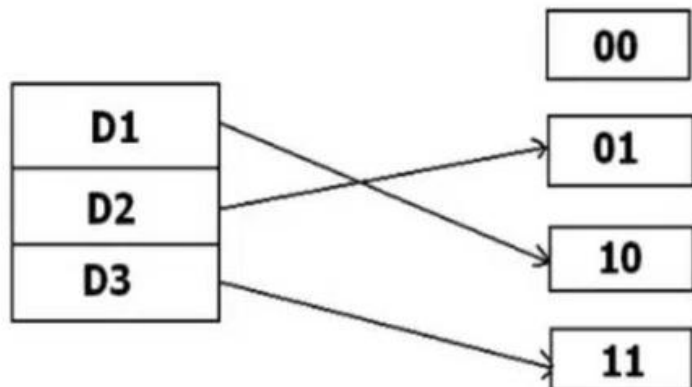
generates three addresses 1001, 0101, and 1010 respectively. This method of storing considers only part of this address – especially only the first bit to store the data. So it tries to load three of them at addresses 0 and 1.

$h(D1) \rightarrow 1001$
 $h(D2) \rightarrow 0101$
 $h(D3) \rightarrow 1010$



But the problem is that No bucket address is remaining for D3. The bucket has to grow dynamically to accommodate D3. So it changes the address to have 2 bits rather than 1 bit, and then it updates the existing data to have a 2-bit address. Then it tries to accommodate D3.

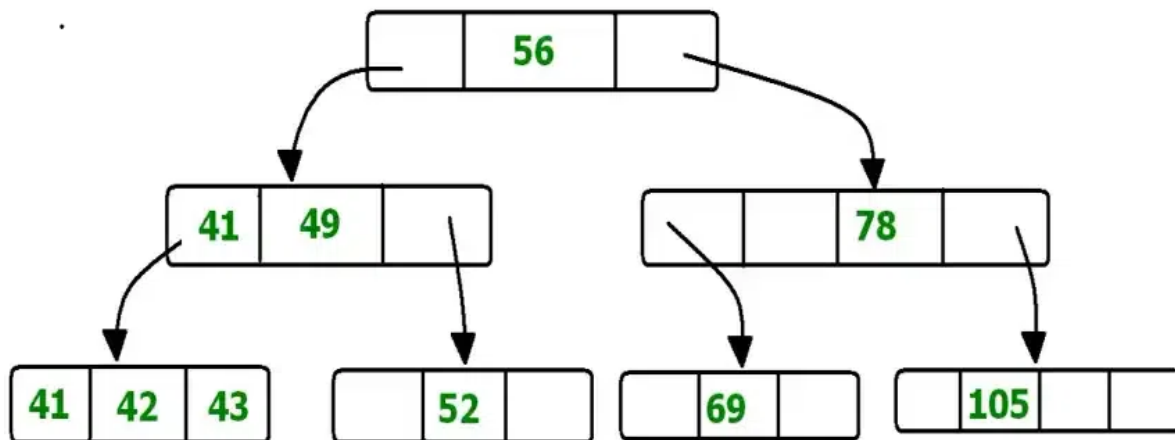
$h(D1) \rightarrow 1001$
 $h(D2) \rightarrow 0101$
 $h(D3) \rightarrow 1010$



B+ Tree File Organization

B+ Tree, as the name suggests, **uses a tree-like structure to store records in a File**. It uses the concept of Key indexing where the primary key is used to sort the records. For each primary key, an index value is generated and mapped with the record. An index of a record is the address of the record in the file.

B+ Tree is very similar to a binary search tree, with the only difference being that instead of just two children, it can have more than two. All the information is stored in a leaf node and the intermediate nodes act as a pointer to the leaf nodes. The information in leaf nodes always remains a sorted sequential linked list.



In the above diagram, 56 is the root node which is also called the main node of the tree.

The intermediate nodes here, just consist of the address of leaf nodes. They do not contain any actual records. Leaf nodes consist of the actual record. All leaf nodes are balanced.

Advantages of B+ Tree File Organization

- **Tree traversal** is easier and faster.
- Searching becomes easy as all records are stored only in leaf nodes and are sorted in sequentially linked lists.

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

Disadvantages of B+ Tree File Organization

- Inefficient for static tables.

Cluster File Organization

In **Cluster file organization**, two or more related tables/records are stored within the same file known as clusters. These files will have two or more tables in the same data block and the key attributes which are used to map these tables together are stored only once.

Thus it lowers the cost of searching and retrieving various records in different files as they are now combined and kept in a single cluster. For example, we have two tables or relation Employee and Department. These tables are related to each other.

EMPLOYEE


| EMP ID | EMP_NAME | EMP_ADD | DEP_ID |
|--------|----------|-----------|--------|
| 01 | JOE | CAPE TOWN | D_101 |
| 02 | ANNIE | FRANSISCO | D_103 |
| 03 | PETER | CROY CITY | D_101 |
| 04 | JOHN | FRANSISCO | D_102 |
| 05 | LUNA | TOKYO | D_106 |
| 06 | SONI | W.LAND | D_105 |
| 07 | SAKACHI | TOKYO | D_104 |
| 08 | MARY | NOVI | D_101 |

DEPARTMENT

| DEP_ID | DEP_NAME |
|--------|----------|
| D_101 | ECO |
| D_102 | CS |
| D_103 | JAVA |
| D_104 | MATHS |
| D_105 | BIO |
| D_106 | CIVIL |

Therefore this table is allowed to combine using a **join operation** and can be seen in a cluster file.

CLUSTER KEY



| DEP_ID | DEP_NAME | EMP ID | EMP_NAME | EMP_ADD |
|--------|----------|--------|----------|-----------|
| D_101 | ECO | 01 | JOE | CAPE TOWN |
| | | 02 | PETER | CROY CITY |
| | | 03 | MARY | NOVI |
| D_102 | CS | 04 | JOHN | FRANSISCO |
| D_103 | JAVA | 05 | ANNIE | FRANSISCO |
| D_104 | MATHS | 06 | SAKACHI | TOKYO |
| D_105 | BIO | 07 | SONI | W.LAND |
| D_106 | CIVIL | 08 | LUNA | TOKYO |

DEPARTMENT + EMPLOYEE

If we have to insert, update or delete any record we can directly do so. Data is sorted based on the primary key or the key with which searching is done.

The **cluster key** is the key with which the joining of the table is performed.

Types of Cluster File Organization

There are two ways to implement this method.

- **Indexed Clusters:** In Indexed clustering, the records are grouped based on the cluster key and stored together. The above-mentioned example of the Employee and Department relationship is an example of an Indexed Cluster where the records are based on the Department ID.
- **Hash Clusters:** This is very much similar to an indexed cluster with the only difference that instead of storing the records based on cluster key, we

generate a hash key value and store the records with the same hash key value.

Advantages of Cluster File Organization

- It is basically used when multiple tables have to be joined with the same joining condition.
- It gives the best output when the cardinality is 1:m.

Disadvantages of Cluster File Organization

- It gives a low performance in the case of a large database.
- In the case of a 1:1 cardinality, it becomes ineffective.

ISAM (Indexed Sequential Access Method):

A combination of sequential and indexed methods. Data is stored sequentially, but an index is maintained for faster access. Think of it like having a bookmark in a book that guides you to specific pages.

Advantages of ISAM :

- Faster retrieval compared to pure sequential methods.
- Suitable for applications with a mix of sequential and random access.

Disadvantages of ISAM :

- Index maintenance can add overhead in terms of storage and update operations.
- Not as efficient as fully indexed methods for random access.

What is Secondary Storage?

Secondary storage, also known as auxiliary storage, is used for storing data that is not as frequently accessible as data in primary storage. This non-volatile memory preserves data until it has either been overwritten or deleted. This type of memory can be hosted on external devices or within the cloud.

Secondary storage devices allow organizations to store data from megabytes to petabytes. It complements primary storage. This means the data does not require primary storage and gets migrated to secondary storage devices. It helps free up space and improves the performance of primary storage devices. These devices are used as secondary storage for backup, data archival and disaster recovery data.

Why is Secondary storage important?

Secondary storage is associated with external devices not directly connected to the production server. By default, the data gets saved to production storage tied to an active application or workload. Storing all the **data** in one place is risky. **Hardware and software** are affected by misconfiguration, malware, and other threats or errors. These events can hamper the production data and impact the organization adversely.

Data storage on a secondary storage platform isolated from the network and production environment prevents data loss to ensure recovery. It is a low-cost storage tier where the data is not immediately accessible. Organizations use secondary storage to ensure that one copy of business data remains inaccessible on the internal network or the **internet**.

Types of Secondary Storage Devices

The different types of secondary storage devices include the following:

1. Magnetic Storage Devices

These secondary storage devices use magnetization to write, rewrite and access data. Such storage devices store data in binary form as tiny and magnetized dots. Such dots are created, read and erased through magnetic fields that are created using tiny electromagnetics. The following are the different types of magnetic storage devices:

1.1 Hard disk drives (HDDs)

A hard disk drive (HDD) is an electro-mechanical data storage device for storing and retrieving digital data using magnetic storage. This magnetic storage comes with one or multiple rigid rapid-rotating platters coated with magnetic material.

Such platters are paired with magnetic heads arranged on a rotating actuator arm that reads and writes data to platter surfaces.

1.2 Magnetic tape drives

A tape drive is a data storage device for reading and writing data on magnetic tape. It is used for archival and offline data storage. It provides sequential access storage that provides direct access storage. They can move to any position on the disk within a few milliseconds, but the tape drive must wind the tape between reels to read any data.

1.3 Floppy Disk Drive

Also known as a floppy diskette, it is a disk storage type composed of thick and flexible storage made of magnetic storage medium. It stores digital data that can be read and written when the disk is inserted into a floppy disk drive that is connected to a computer. These secondary storage devices have become obsolete and have been replaced with memory cards, flash drives and cloud storage.

2. Solid-state drives (SSDs)

These devices are also known as solid-state drives (**SSDs**) that use NAND-based flash memory for storing and retrieving data. Unlike traditional hard disk drives that use spinning disks to access data, SSDs do not have any moving parts.

2.1 Optical storage devices

These devices use optical storage technology for reading and writing data. This data gets digitally stored, and for reading and writing data, lasers are used.

2.2.1 Compact Disk (CD)

It is a digital optical disc data storage format adapted for general-purpose data storage. The capacity of compact discs is extended to 80 minutes and is 700 MiB by arranging the data more closely on the same-sized discs.

2.2.2 DVDs

DVD (Digital Video Disc) can store any digital data widely used for video programs or storing software. These can store 15 times CDs' data and rich multimedia files

that require high storage capacity. DVDs are used in DVD-Video consumer digital video and DVD-Audio consumer digital audio formats.

2.2.3. Blu-Ray Disc

Also known as Blu-ray, it is a digital optical disc data storage format. High-definition videos are stored on Blu-ray discs with up to 1920×1080 pixel resolution at 24& 50/60 progressive or 50/60 interlaced frames per second. Besides hardware specifications, it is associated with a set of multimedia formats.

Characteristics of Secondary Storage Devices

- **Non-Volatility:** Secondary storage devices are non-volatile, meaning they retain data even when not powered. This contrasts with primary storage devices like RAM, which lose data when power is lost. Non-volatility is crucial for long-term data storage.
- **Capacity:** Secondary storage devices typically have a high capacity, allowing them to store large amounts of data. The capacity of a secondary storage device can vary from a few gigabytes (GB) to several terabytes (TB), depending on the type of device.
- **Speed:** The speed of a secondary storage device refers to how quickly data can be read from or written to the device. While secondary storage devices are generally slower than primary storage devices, technological advancements have led to developing secondary storage devices with high data transfer rates.
- **Cost:** Generally, secondary storage devices are cost-effective, providing a large amount of storage space for a relatively low cost. However, the cost can vary depending on the type of device, with solid-state drives (SSDs) being more expensive than hard disk drives (HDDs), for example.

How Secondary Storage Devices Work?

- **Data Writing Process:** When written to a secondary storage device, it is transferred from the primary storage (such as RAM) to the secondary storage device. The data is then stored on the device, allowing it to be retrieved later. The exact process of writing data can vary depending on the type of

secondary storage device. For example, data is written in a hard disk drive by magnetizing a thin film of ferromagnetic material on a disk.

- **Data Reading Process:** When data is read from a secondary storage device, it is retrieved from the device and transferred to the primary storage (such as **RAM**), where the CPU can access it. The process of reading data can also vary depending on the type of secondary storage device. For example, in a hard disk drive, data is read by detecting the magnetized areas of the disk.

OS File Operations

File operations within an operating system (OS) encompass a set of essential tasks and actions directed at files and directories residing within a computer's file system. These operations are fundamental for the effective management and manipulation of data stored on various storage devices. In this article, we will learn different file operations and what are the **system calls** and APIs used to perform them in a Linux / Windows-based OS.

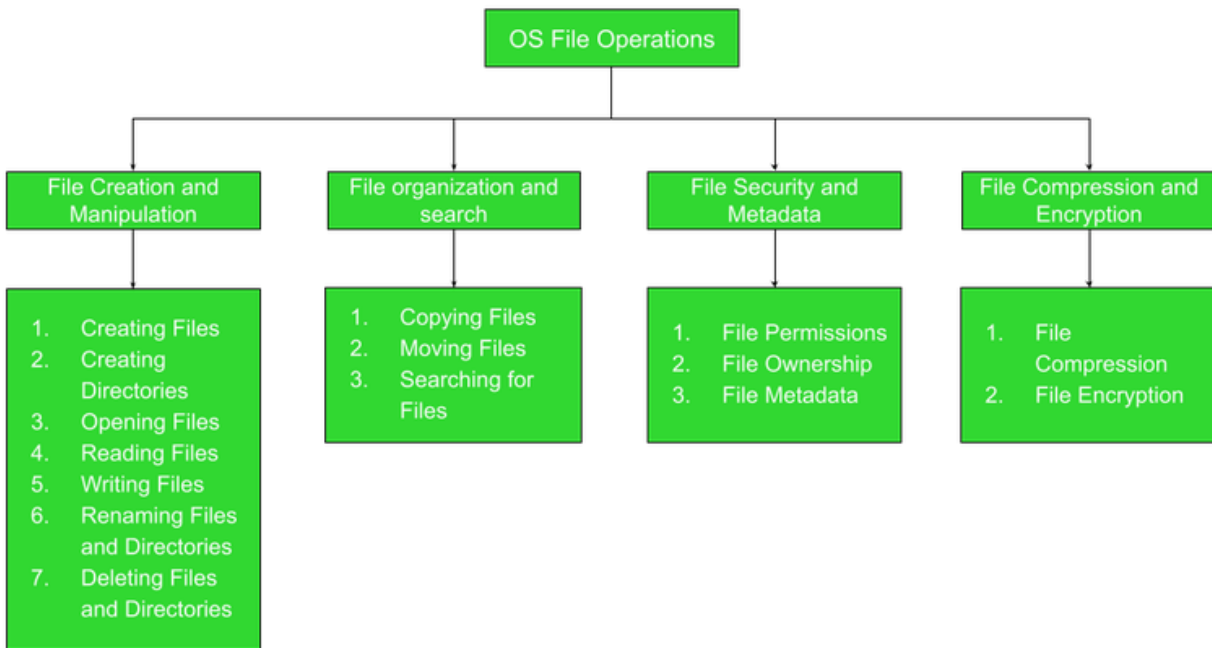


Fig: 1.1 OS File Operations

File Creation and Manipulation

File Creation and Manipulation encompasses essential operations within an operating system that involve creating, modifying, and organizing files and directories. These actions are vital for managing data efficiently and are integral to the functioning of computer systems.

| File Operation | Description | System Calls / APIs |
|-----------------------------|---|-----------------------------|
| Creating Files | Create a new file for data storage. | CreateFile() (Windows) |
| Creating Directories | Create a new directory for organizing files. | CreateDirectory() (Windows) |
| Opening Files | Open a file that you already have open to read or write from. | OpenFile() (Windows) |
| Reading Files | Retrieve data from an open file. | ReadFile() (Windows) |

| | | |
|---------------------------------------|---|---|
| Writing Files | Store data in an open file. | WriteFile() (Windows) |
| Renaming Files and Directories | If you want to rename a file or directory,. | MoveFile() (Windows) |
| Deleting Files and Directories | Remove files or directories. | DeleteFile() (Windows) RemoveDirectory() (Windows) |

File Organization and Search

File organization and search are key OS operations for arranging files systematically and swiftly locating specific data, optimizing file management and user efficiency.

| File Operation | Description | System Calls / APIs |
|----------------------------|---|--|
| Copying Files | Create duplicates of files in another location. | CopyFile() (Windows) |
| Moving Files | Relocate files from one location to another. | MoveFile() (Windows) |
| Searching for Files | Locate files based on specific criteria. | FindFirstFile() and FindNextFile() (Windows) |

File Security and Metadata

File Security and Metadata are vital components of file management, encompassing access control and crucial file information preservation within an operating system. They are essential for data security and efficient organization.

| File Operation | Description | System Calls / APIs |
|-------------------------|---|-------------------------------|
| File Permissions | Control access rights to files and directories. | SetFileSecurity (Windows) |
| File Ownership | Assign specific users or groups as file owners. | SetFileSecurity (Windows) |
| File Metadata | Retrieve and manipulate file information. | GetFileAttributesEx (Windows) |

File Compression and Encryption

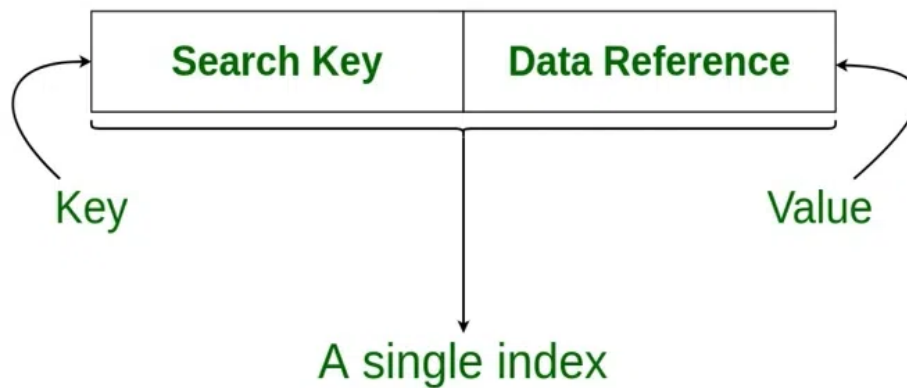
File Compression and Encryption are essential for optimizing storage and enhancing data security. Compression reduces file sizes, while encryption safeguards data privacy by making it unreadable without the correct decryption key.

| File Operation | Description | System Calls / APIs |
|-------------------------|--|---|
| File Compression | Reduce file sizes to save storage space. | Compress-Archive (Windows) |
| File Encryption | Protect data by converting it into an unreadable format. | Windows provides encryption libraries and APIs for encryption operations. |

Indexing in Databases

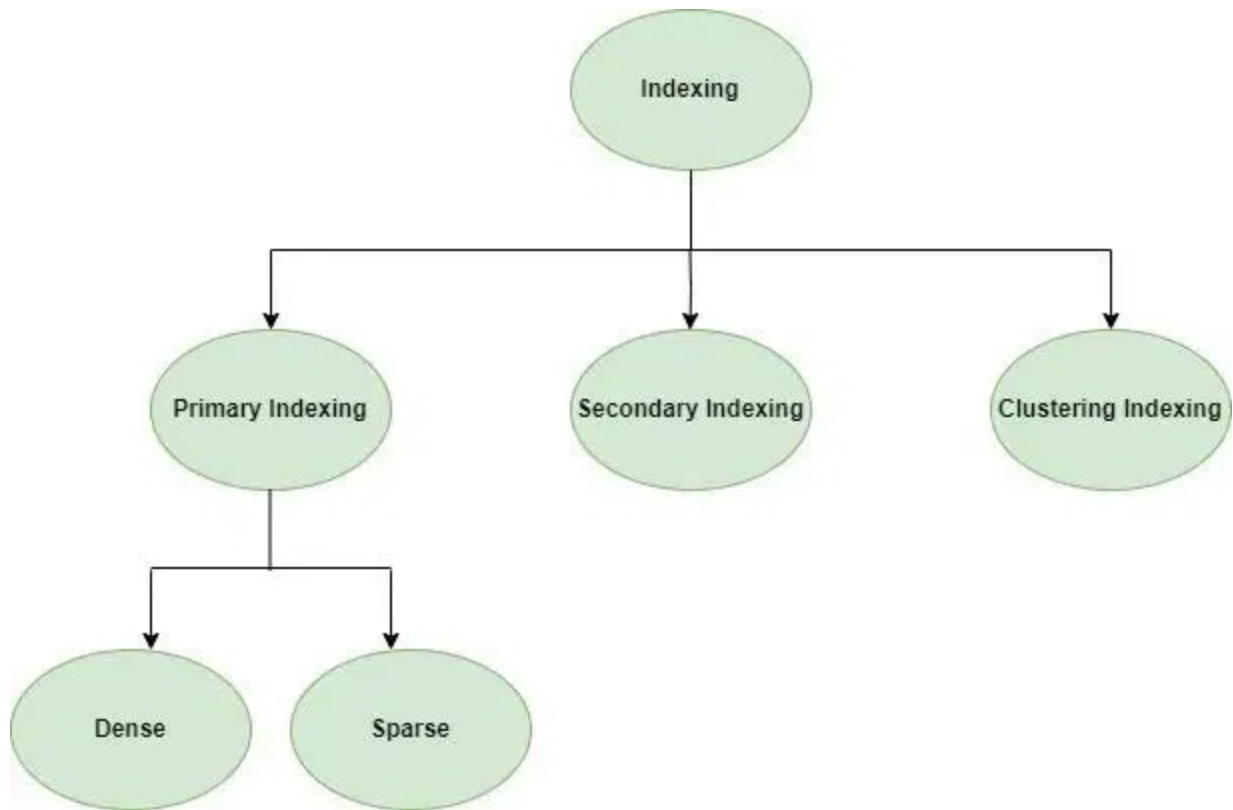
Indexing improves database performance by minimizing the number of disc visits required to fulfill a query. It is a data structure technique used to locate and quickly access data in databases. Several database fields are used to generate indexes. The main key or candidate key of the table is duplicated in the first column, which is the Search key. To speed up data retrieval, the values are also kept in sorted order. It should be highlighted that sorting the data is not required. The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.

Structure of an Index in Database



Attributes of Indexing

- **Access Types:** This refers to the type of access such as value-based search, range access, etc.
- **Access Time:** It refers to the time needed to find a particular data element or set of elements.
- **Insertion Time:** It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time:** Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead:** It refers to the additional space required by the index.



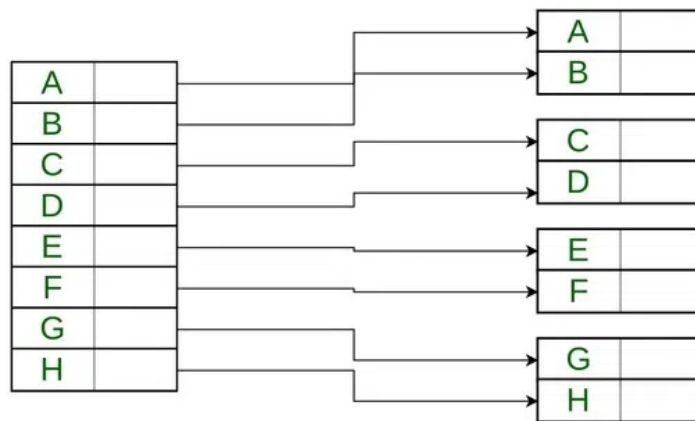
In general, there are two types of file organization mechanisms that are followed by the indexing methods to store the data:

Sequential File Organization or Ordered Index File

In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organizations might store the data in a dense or sparse format.

- **Dense Index**
 - For every search key value in the data file, there is an index record.
 - This record contains the search key and also a reference to the first data record with that search key value.

Dense Index



For every search value in a Data File,

There is an Index Record.

Hence the name **Dense Index**.

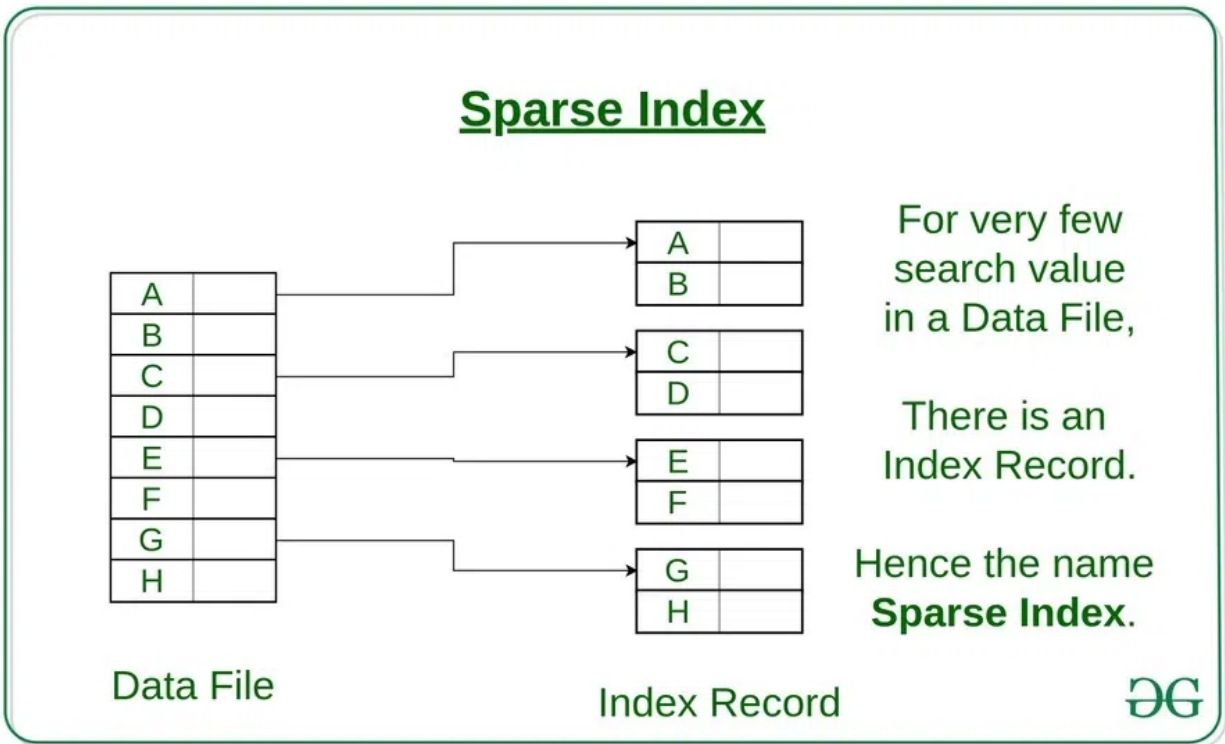
Data File

Index Record



- **Sparse Index**

- The index record appears only for a few items in the data file. Each item points to a block as shown.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.
- Number of Accesses required= $\log_2(n)+1$, (here n =number of blocks acquired by index file)



Hash File Organization

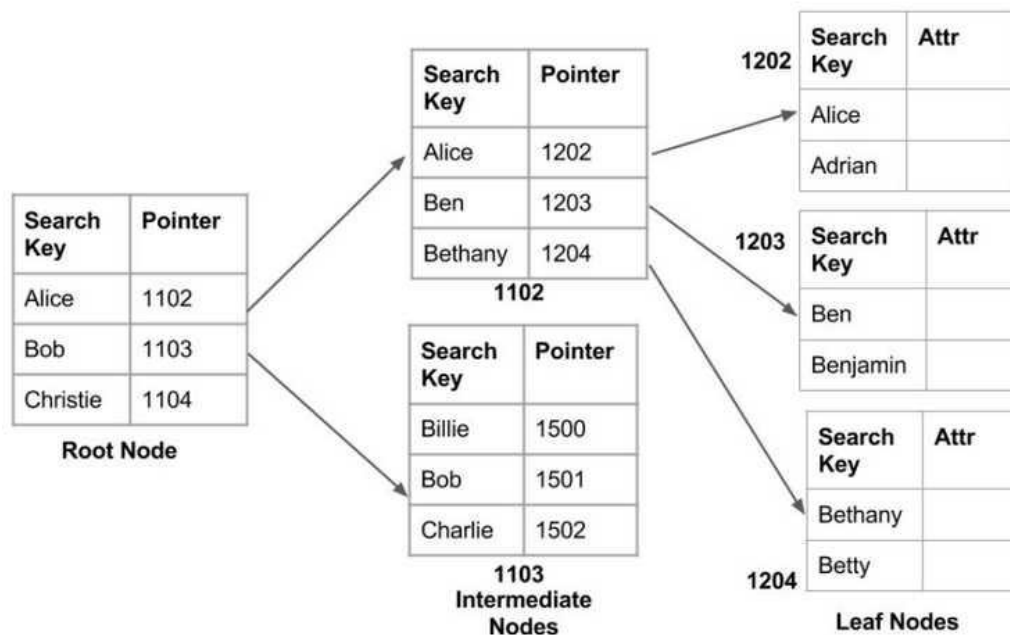
Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned are determined by a function called a hash function. There are primarily three methods of indexing:

- Clustered Indexing:** When more than two records are stored in the same file this type of storing is known as cluster indexing. By using cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored in one place and it also gives the frequent joining of more than two tables (records). The clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create an index out of them. This method is known as the clustering index. Essentially, records with similar properties are grouped together, and indexes for these groupings are formed. Students studying each semester, for example, are grouped together.

First-semester students, second-semester students, third-semester students, and so on are categorized.

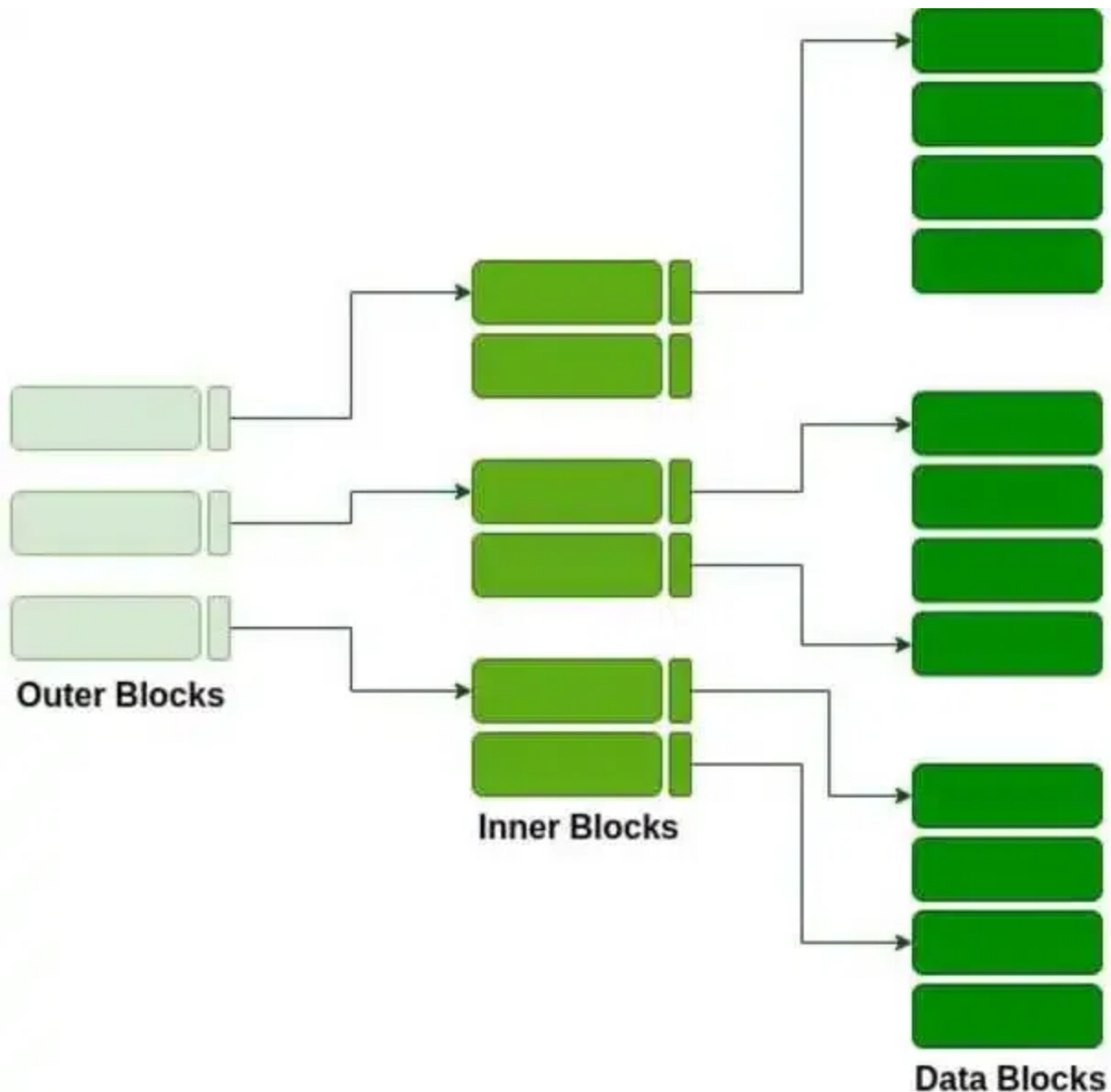
| INDEX FILE | | Data Blocks in Memory | | | | |
|------------|---------------|-----------------------|-----|--------|-------------------|--------|
| SEMESTER | INDEX ADDRESS | | | | | |
| 1 | | → | 100 | Joseph | Alaiedon Township | 20 200 |
| 2 | | | 101 | | | |
| 3 | | | | | | |
| 4 | | | 110 | Allen | Fraser Township | 20 200 |
| 5 | | | 111 | | | |
| | | | 120 | Chris | Clinton Township | 21 200 |
| | | | 121 | | | |
| | | | 200 | Patty | Troy | 22 205 |
| | | | 201 | | | |
| | | | 210 | Jack | Fraser Township | 21 202 |
| | | | 211 | | | |
| | | | 300 | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

- Primary Indexing:** This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces **sequential file organization**. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.
- Non-clustered or Secondary Indexing:** A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here (information on each page of the book) is not organized but we have an ordered reference (contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly. It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



Non clustered index

- Multilevel Indexing:** With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can be stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.



Advantages of Indexing

- **Improved Query Performance:** Indexing enables faster data retrieval from the database. The database may rapidly discover rows that match a specific value or collection of values by generating an index on a column, minimizing the amount of time it takes to perform a query.
- **Efficient Data Access:** Indexing can enhance data access efficiency by lowering the amount of disk I/O required to retrieve data. The database can maintain the data pages for frequently visited columns in memory by

generating an index on those columns, decreasing the requirement to read from disk.

- **Optimized Data Sorting:** Indexing can also improve the performance of sorting operations. By creating an index on the columns used for sorting, the database can avoid sorting the entire table and instead sort only the relevant rows.
- **Consistent Data Performance:** Indexing can assist ensure that the database performs consistently even as the amount of data in the database rises. Without indexing, queries may take longer to run as the number of rows in the table grows, while indexing maintains a roughly consistent speed.
- By ensuring that only unique values are inserted into columns that have been indexed as unique, indexing can also be utilized to ensure the integrity of data. This avoids storing duplicate data in the database, which might lead to issues when performing queries or reports.

Overall, indexing in databases provides significant benefits for improving query performance, efficient data access, optimized data sorting, consistent data performance, and enforced data integrity

Disadvantages of Indexing

- Indexing necessitates more storage space to hold the index data structure, which might increase the total size of the database.
- **Increased database maintenance overhead:** Indexes must be maintained as data is added, destroyed, or modified in the table, which might raise database maintenance overhead.
- Indexing can reduce insert and update performance since the index data structure must be updated each time data is modified.
- **Choosing an index can be difficult:** It can be challenging to choose the right indexes for a specific query or application and may call for a detailed examination of the data and access patterns.

Features of Indexing

- The development of data structures, such as **B-trees** or **hash tables**, that provide quick access to certain data items is known as indexing. The data structures themselves are built on the values of the indexed columns, which are utilized to quickly find the data objects.
- The most important columns for indexing columns are selected based on how frequently they are used and the sorts of queries they are subjected to. The **cardinality**, selectivity, and uniqueness of the indexing columns can be taken into account.
- There are several different index types used by databases, including primary, secondary, clustered, and non-clustered indexes. Based on the particular needs of the database system, each form of index offers benefits and drawbacks.
- For the database system to function at its best, periodic index maintenance is required. According to changes in the data and usage patterns, maintenance work involves building, updating, and removing indexes.
- Database query optimization involves indexing, which is essential. The query optimizer utilizes the indexes to choose the best execution strategy for a particular query based on the cost of accessing the data and the selectivity of the indexing columns.
- Databases make use of a range of indexing strategies, including covering indexes, index-only scans, and partial indexes. These techniques maximize the utilization of indexes for particular types of queries and data access.
- When non-contiguous data blocks are stored in an index, it can result in index fragmentation, which makes the index less effective. Regular index maintenance, such as defragmentation and reorganization, can decrease **fragmentation**.

What is a B-Tree?

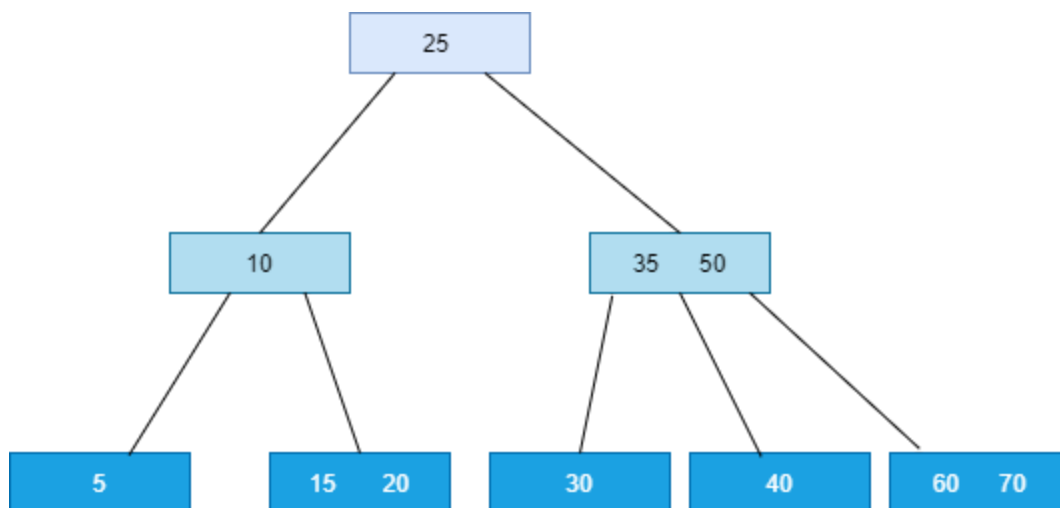
B Tree is a self-balancing tree data structure. It stores and maintains data in a sorted form where the left children of the root are smaller than the root and the right children are larger than the root in value. It makes searching efficient and allows all operations in logarithmic time. It allows nodes with more than two children. B-tree is used for implementing multilevel indexing. Every node of the B-

tree stores the key-value along with the data pointer pointing to the block in the disk file containing that key.

Properties of B-Tree

- Every node has at most m children where m is the order of the B-Tree.
- A node with K children contains $K-1$ keys.
- Every non-leaf node except the root node must have at least $\lceil m/2 \rceil$ child nodes.
- The root must have at least 2 children if it is not the leaf node too.
- All leaves of a B-Tree stays at the same level.
- Unlike other trees, its height increases upwards towards the root, and insertion happens at the leaf node.
- The time complexity of all the operations of a B-Tree is $O(\log n)$, here 'n' is the number of elements in the B-Tree.

Example of a B-Tree of order 4



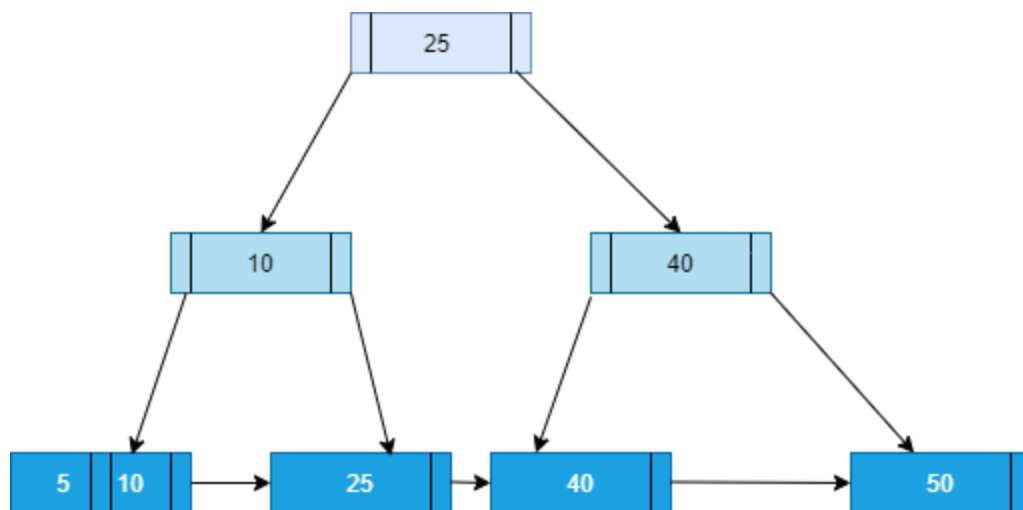
What is a B+ Tree?

A B+ tree is similar to a B-tree, the only difference is that their leaves are linked at the bottom. Unlike B-tree, the nodes of the B+ tree do not store keys along with the pointers to the disk block. The internal nodes contain only keys and the leaf

nodes contain the keys along with the data pointers. All the internal nodes are present at the leaves and are linked together and can be traversed just like a linked list.

B+ tree also removes one drawback of using B-tree. The internal nodes of the B-tree also store keys along with data pointers which take more space and significantly reduce the number of keys that can be stored at a node of the B-tree. Due to this the number of levels increases and searching time also increases. Unlike this, the B+ tree can store more keys than the B-tree of the same level because of their feature of storing pointers only at the leaf nodes. This contributes to storing more entries at fewer levels in the B+ tree and lessens the searching time for a key. This makes the B+ tree very efficient and very quick in accessing the data from the disks.

Example of B+ Tree



Differences between B-Tree and B+ Tree

| B-Tree | B+ Tree |
|--|--|
| All internal nodes and leaf nodes contain data pointers along with keys. | Only leaf nodes contain data pointers along with keys, internal nodes contain keys only. |
| There are no duplicate keys. | Duplicate keys are present in this, all internal nodes are also present at leaves. |
| Leaf nodes are not linked to each other. | Leaf nodes are linked to each other. |

| | |
|---|--|
| Sequential access of nodes is not possible. | All nodes are present at leaves, so sequential access is possible just like a linked list. |
| Searching for a key is slower. | Searching is faster. |
| For a particular number of entries, the height of the B-tree is larger. | The height of the B+ tree is lesser than B-tree for the same number of entries. |

Definition and overview of ODBMS

The data model in which data is kept in the form of objects, which are instances of classes, is known as an object-oriented database management system, or ODBMS. The object-oriented data model is made up of these classes and objects.

A database management system (DBMS) that facilitates the modeling and generation of data as objects is called an object-oriented database management system (ODBMS), which is frequently abbreviated as ODBMS for object database management system. Inheritance of class attributes and methods by subclasses and their objects is also included, as is some sort of support for object classes.

Although the Object Data Management Group (ODMG) produced The Object Data Standard ODMG 3.0 in 2001, which outlines an object model and standards for defining and querying objects, there is no commonly accepted definition of what an OODBMS is. Since then, the group has broken up.

Nowadays, a well-liked substitute for the object database is Not Only SQL (NoSQL) document database systems. NoSQL document databases offer key-based access to semi-structured data as documents, generally using JavaScript Object Notation, even if they lack all the features of a full ODBMS (JSON).

Object-Oriented Data Model Elements

The three main building blocks of the OODBMS are object structure, object classes, and object identity. The following explains them.

Object Structure: An object's structure refers to the components that make up the object. An attribute is a term used to describe certain characteristics of an item. A real-world entity with certain qualities that makes up the object structure is hence referred to as an object. Also, an object contains the data code in a solitary piece, which in turn creates data abstraction by shielding the user from the implementation specifics.

1. Messages - A message operates as a communication channel or as an interface between an entity and the outside world. There are two sorts of messages:
2. Read-only message: The invoking message is referred to as a read-only message if the called method does not alter the value of a variable.
3. Update message: The invoking message is referred to as an update message if the invoked method modifies the value of a variable.
4. Methods: A method is a chunk of code that is executed when a message is given. Every time a method is used, a value is returned as output. There are two categories of methods:
5. Read-only method: A method is referred to as a read-only method when it has no effect on the value of a variable.
6. Update-method: An update method is one that modifies a variable's value in some way.
7. Variables are used to store an object's data. The variables' data allows the objects to be distinguished from one another.

Object Classes: An instance of a class is an object, which is a real-world item. In order to create objects that differ in the data they hold but have the same class definition, a class must first be defined. Messages and variables recorded in the objects themselves relate to the objects in turn.

```
class JOB
{ //variables
    char name;
    string address;
    int id;
    int salary;
    //Messages
    char get_name();
    string get_address();
    int annual_salary();
};
```

As we can see in the sample above, the class CLERK is where the object variables and messages are stored.

Although there may be several classes with comparable methods, variables, and messages in a database, an OODBMS also extensively allows inheritance. As a result, the idea of the class hierarchy is still used to illustrate the commonalities between different classes.

An object-oriented data model also supports the idea of encapsulation, which is data or information concealing. In addition to the built-in data types like char, int, and float, this data architecture also offers the ability for abstract data types. ADTs are user-defined data types that can carry methods in addition to the values they contain.

Hence, ODBMS offers a variety of built-in and user-defined features to its customers. It combines the attributes of an object-oriented data model with those of a database management system, and it supports the notions of programming paradigms like classes and objects in addition to those of encapsulation, inheritance, and user-defined ADTs (abstract data types).

Features of ODBMS

Malcolm Atkinson and colleagues defined an OODBMS in their important work, The Object-Oriented Database Manifesto, as follows:

An object-oriented database system must meet two requirements: it must be a database management system (DBMS) and it must be an object-oriented system, meaning that it must, to the greatest degree feasible, be compatible with the current crop of object-oriented programming languages.

Persistence, secondary storage management, concurrency, recovery, and an ad hoc query capability are the five qualities that correspond to the first criteria.

The second one translates into eight characteristics: extensibility, computational completeness, overriding paired with late binding, inheritance, types or classes, complex objects, object identity, and encapsulation.

RDBMS Vs ODBMS

The type of DBMS that is now used the most frequently is a relational database management system (RDBMS). Most IT workers have a solid understanding of the

relational abstraction of rows and columns accessible via Structured Query Language (SQL).

Yet, object database systems may be more effective in managing and storing complicated data relationships. Accessing data with several relationships spread across various tables in an RDBMS might be more challenging for applications than accessing the same data as an object in an ODBMS.

In addition, a lot of programmers employ object-oriented programming (OOP) languages to create applications, including Java, C++, Python, and others. Conversions between complicated objects and rows from various relational database tables are necessary when using an RDBMS to store and retrieve objects. Tools for object-relational mapping (ORM) can make this process simpler; nonetheless keep the mapping overhead in place.

Several RDBMS companies now provide object-relational database management systems as part of their product lines (ORDBMS). Of fact, adding some object-oriented ideas to relational databases does not give users access to all of an ODBMS's features.

Distributed Database System

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Types:

1. Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

2. Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions.

Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Distributed Data Storage :

There are 2 ways in which data can be stored on different sites. These are:

1. Replication –

In this approach, the entire relationship is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

This is advantageous as it increases the availability of data at different sites. Also, now query requests can be processed in parallel.

However, it has certain disadvantages as well. Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

2. Fragmentation –

In this approach, the relations are fragmented (i.e., they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data).

Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

Fragmentation of relations can be done in two ways:

- **Horizontal fragmentation – Splitting by rows** – The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.
- **Vertical fragmentation – Splitting by columns** – The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure a lossless join.

In certain cases, an approach that is hybrid of fragmentation and replication is used.

Applications of Distributed Database:

- It is used in Corporate Management Information System.
- It is used in multimedia applications.
- Used in Military's control system, Hotel chains etc.
- It is also used in manufacturing control system.

A distributed database system is a type of database management system that stores data across multiple computers or sites that are connected by a network. In a distributed database system, each site has its own database, and the databases are connected to each other to form a single, integrated system.

The main advantage of a distributed database system is that it can provide higher availability and reliability than a centralized database system. Because the data is stored across multiple sites, the system can continue to function even if one or more sites fail. In addition, a distributed database system can provide better performance by distributing the data and processing load across multiple sites.

There are several different architectures for distributed database systems, including:

Client-server architecture: In this architecture, clients connect to a central server, which manages the distributed database system. The server is responsible for coordinating transactions, managing data storage, and providing access control.

Peer-to-peer architecture: In this architecture, each site in the distributed database system is connected to all other sites. Each site is responsible for managing its own data and coordinating transactions with other sites.

Federated architecture: In this architecture, each site in the distributed database system maintains its own independent database, but the databases are integrated through a middleware layer that provides a common interface for accessing and querying the data.

Distributed database systems can be used in a variety of applications, including e-commerce, financial services, and telecommunications. However, designing and

managing a distributed database system can be complex and requires careful consideration of factors such as data distribution, replication, and consistency.

Advantages of Distributed Database System :

- 1) There is fast data processing as several sites participate in request processing.
- 2) Reliability and availability of this system is high.
- 3) It possess reduced operating cost.
- 4) It is easier to expand the system by adding more sites.
- 5) It has improved sharing ability and local autonomy.

Disadvantages of Distributed Database System :

- 1) The system becomes complex to manage and control.
- 2) The security issues must be carefully managed.
- 3) The system require deadlock handling during the transaction processing otherwise
the entire system may be in inconsistent state.
- 4) There is need of some standardization for processing of distributed database system.