Unit-4

Secondary Indexing in Databases

Databases are a critical component of modern applications, storing vast amounts of data and serving as a source of information for various functions. One of the primary challenges in managing databases is providing efficient access to the stored data. To meet this challenge, database management systems use various techniques, including indexing, to improve the performance of data retrieval operations. Indexing is a method that creates a separate structure, referred to as an index, from the data stored in a database. The purpose of an index is to allow for fast access to data without having to search through the entire dataset. There are several types of indexes, including primary indexes and secondary indexes.

What is Secondary Indexing in Databases?

Secondary indexing is a database management technique used to create additional indexes on data stored in a database. The main purpose of secondary indexing is to improve the performance of queries and to simplify the search for specific records within a database. A secondary index provides an alternate means of accessing data in a database, in addition to the primary index. The primary index is typically created when the database is created and is used as the primary means of accessing data in the database. Secondary indexes, on the other hand, can be created and dropped at any time, allowing for greater flexibility in managing the database.

Benefits

- **Improved Query Performance:** Secondary indexes can improve the performance of queries by reducing the amount of data that needs to be scanned to find the desired records. With a secondary index, the database can directly access the required records, rather than having to scan the entire table.
- **Flexibility:** Secondary indexes provide greater flexibility in managing a database, as they can be created and dropped at any time. This allows for a more dynamic approach to database management, as the needs of the database can change over time.
- **Simplified Search:** Secondary indexes simplify the search for specific records within a database, making it easier to find the desired data.
- **Reduced Data Storage Overhead:** Secondary indexes use a compact data structure that requires less space to store compared to the original data. This means that you can store more data in a database while reducing the amount of storage space required.

Types of Secondary Indexes

- **B-tree Index:** A B-tree index is a type of index that stores data in a balanced tree structure. B-tree indexes are commonly used in relational databases and provide efficient search, insert, and delete operations.
- **Hash Index**: A hash index is a type of index that uses a hash function to map data to a specific location within the index. Hash indexes are commonly used in non-relational databases, such as NoSQL databases, and provide fast access to data.

- **Bitmap Index:** A bitmap index is a type of index that uses a bitmap to represent the data in a database. Each bit in the bitmap represents a specific record in the database, and the value of the bit indicates whether the record is present or not. Bitmap indexes are commonly used in data warehousing and business intelligence applications, as they provide efficient access to large amounts of data.

When to Use Secondary Indexing

Secondary indexing should be used in database management systems when there is a need to improve the performance of data retrieval operations that search for data based on specific conditions. Secondary indexing is particularly useful in the following scenarios:

- **Queries with Complex Search Criteria:** Secondary indexes can be used to support complex queries that search for data based on multiple conditions. By creating a secondary index based on the columns used in the search criteria, database management systems can access the data more efficiently.
- **Large Data Sets:** Secondary indexing can be beneficial for large data sets where the time and resources required for data retrieval operations can be significant. By creating a secondary index, database management systems can access the data more quickly, reducing the time and resources required for data retrieval operations.
- **Frequently Accessed Data:** Secondary indexing should be used for frequently accessed data to reduce the time and resources required for data retrieval operations. This is because secondary indexes provide a fast and efficient way to access data stored in a database.
- **Sorting and Aggregating Data:** Secondary indexing can be used to support sorting and aggregating data based on specific columns. By creating a secondary index based on the columns used for sorting and aggregating, database management systems can access the data more efficiently, reducing the time and resources required for data retrieval operations.
- **Data Structure:** The data structure of a database can also affect the decision to use secondary indexing. For example, if the data is structured as a B-tree, a B-tree index may be the most appropriate type of secondary index.

File Organization in DBMS

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

File – A file is named 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 certain order is called file Organization. **File**

Structure refers to the format of the label and data blocks and of any logical control record.

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 :

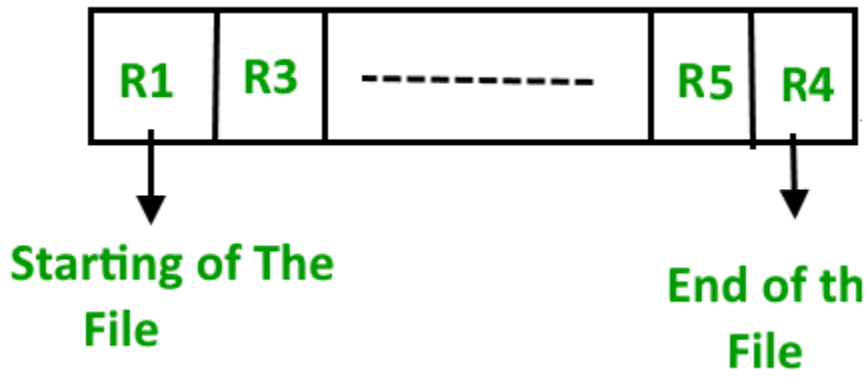
- Sequential File Organization
- Heap File Organization
- Hash File Organization
- B+ Tree File Organization
- Clustered File Organization

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

Sequential File Organization –

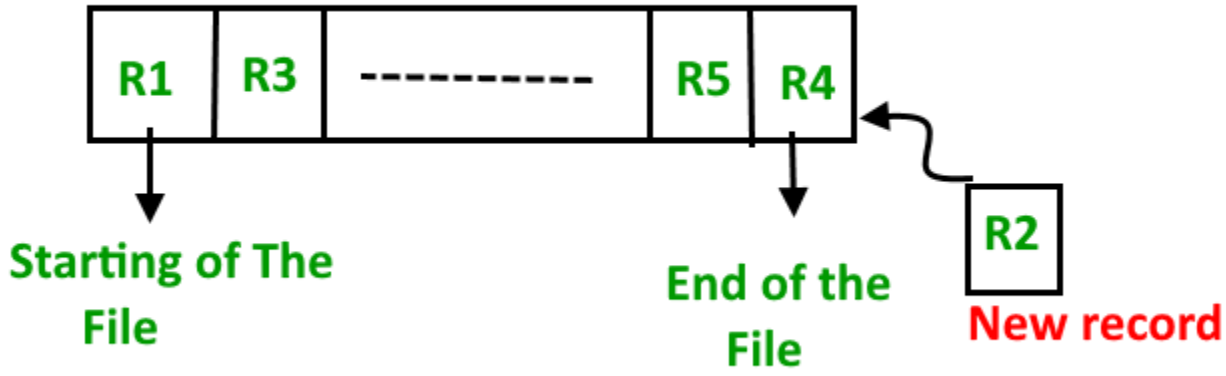
The easiest method for file Organization is Sequential method. In this method the file are stored one after another in a sequential manner. There are two ways to implement this method:

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

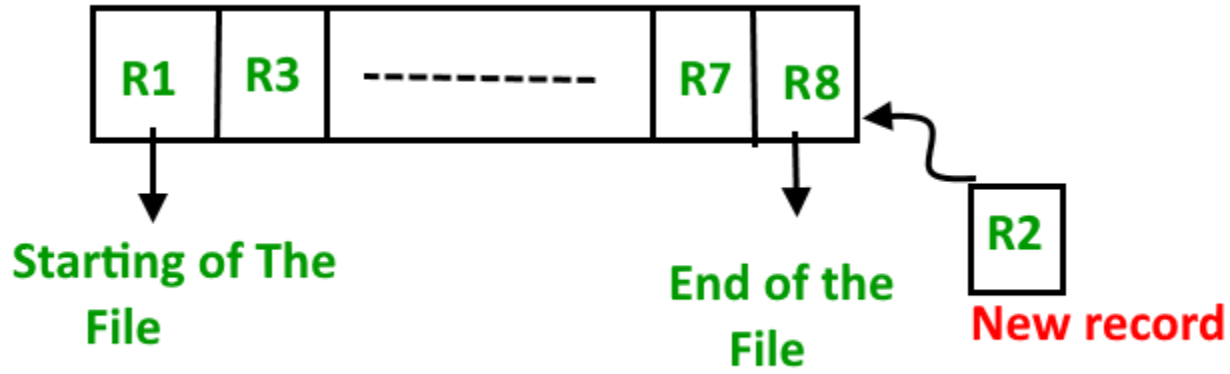


1. **Insertion of new record –**
   Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted

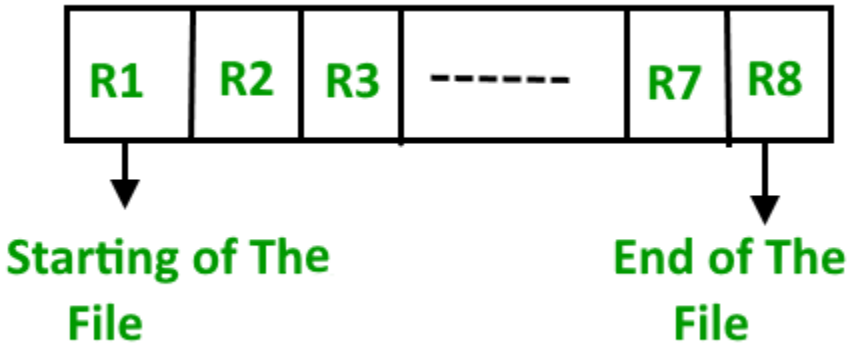
in the sequence, then it is simply placed at the end of the file.



- **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.



1. **Insertion of new record –**
   Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto 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 .



Pros and Cons of Sequential File Organization –
Pros –

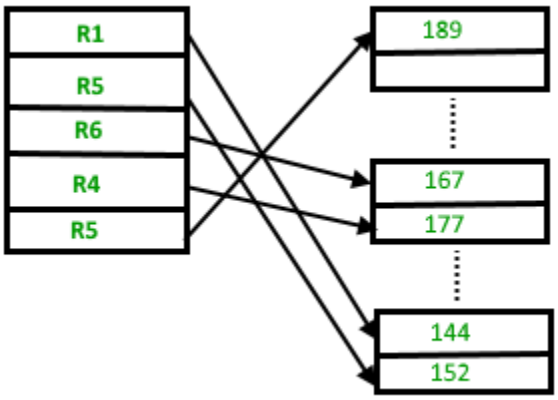
- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Cons –

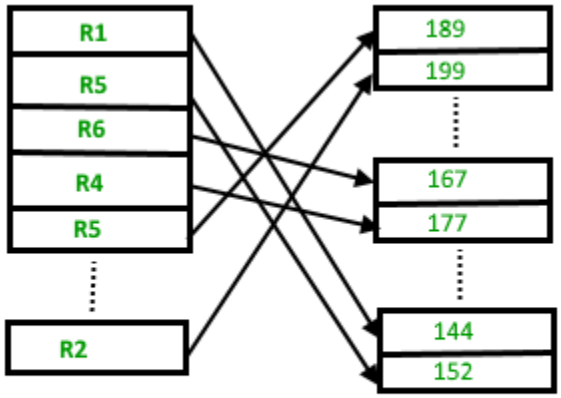
- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

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 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, lets say data block 1.



If we want to search, delete or update data in heap file Organization the 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.

Pros and Cons of Heap File Organization –
Pros –

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

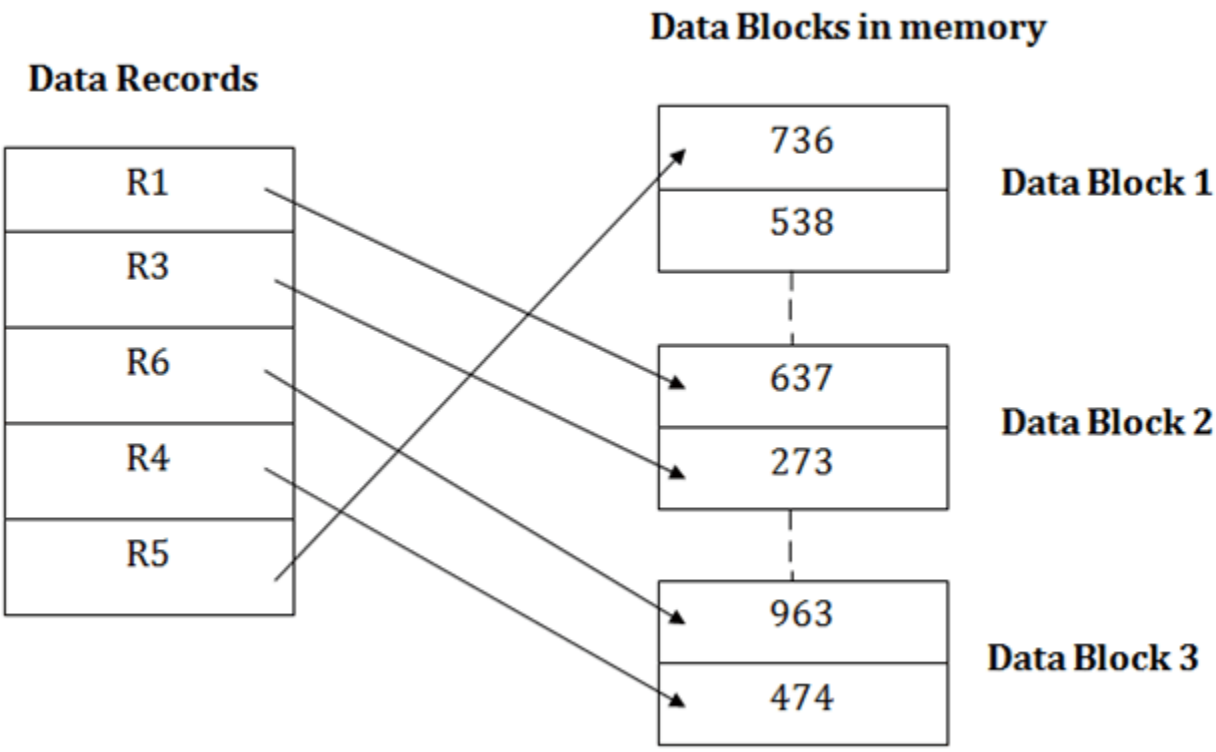
Cons –

- Problem of unused memory blocks.
- Inefficient for larger databases.

Heap file organization

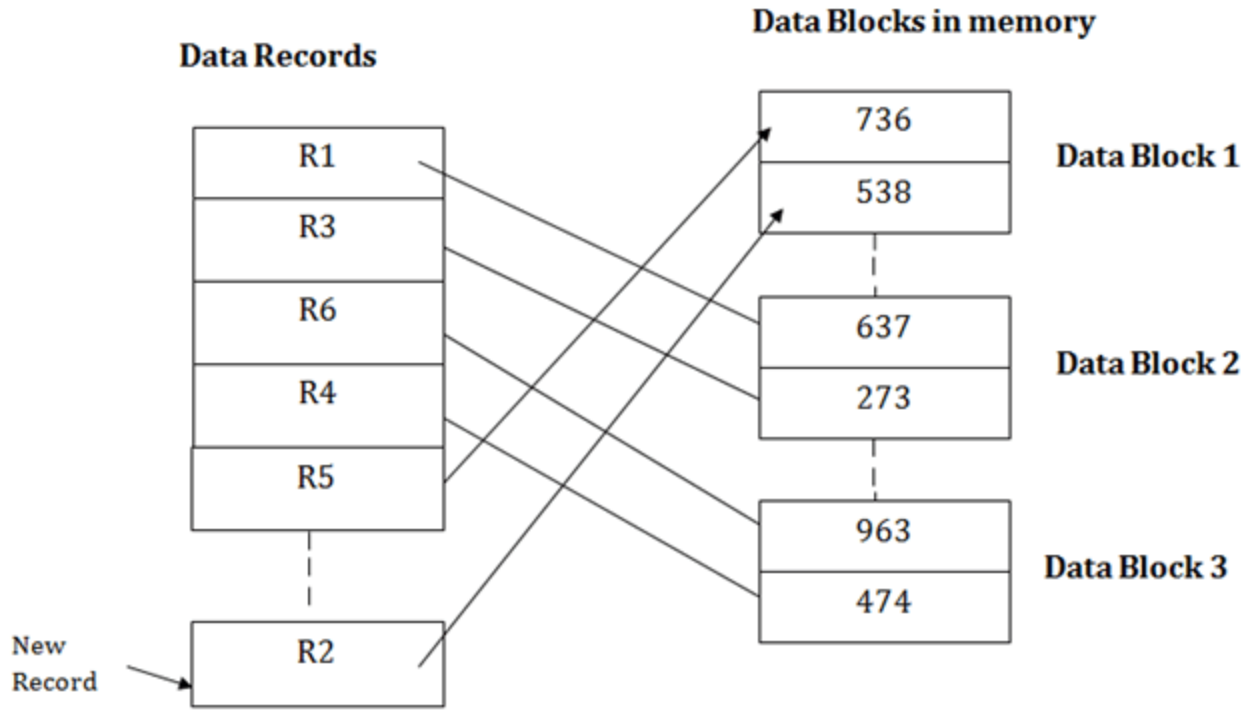
- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.

- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

- This method is inefficient for the large database because it takes time to search or modify the record.
-
- This method is inefficient for large databases.

File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

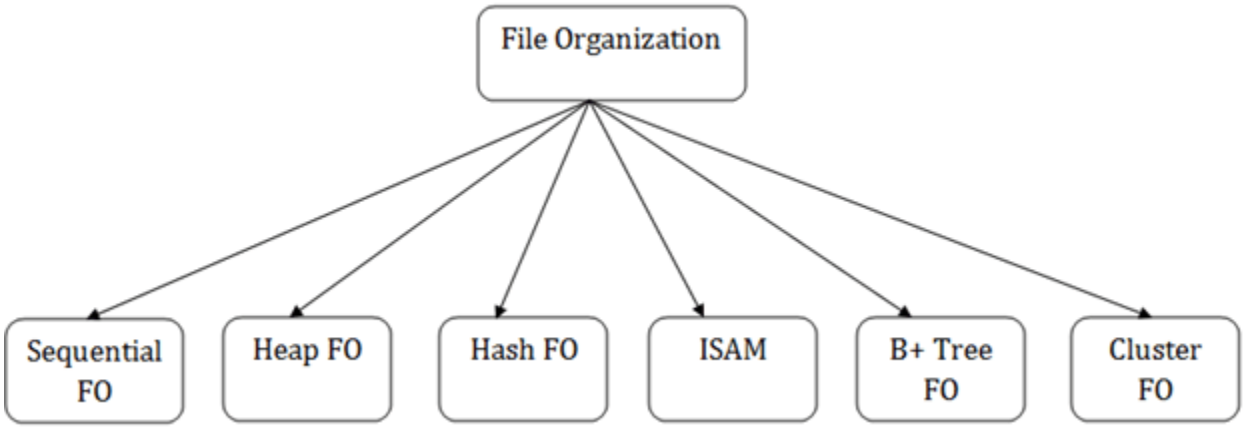
Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



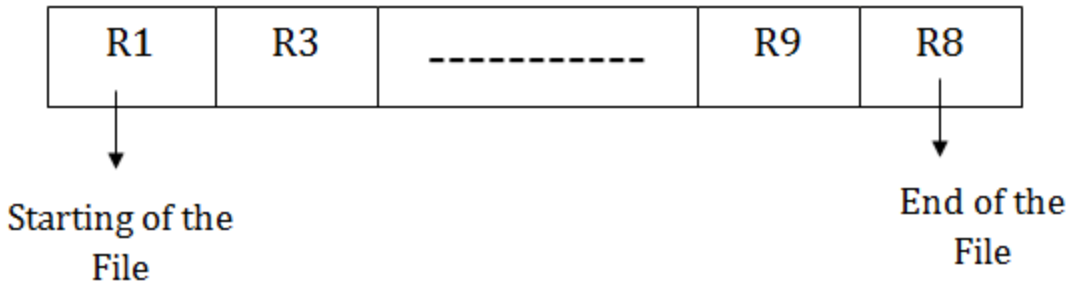
- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

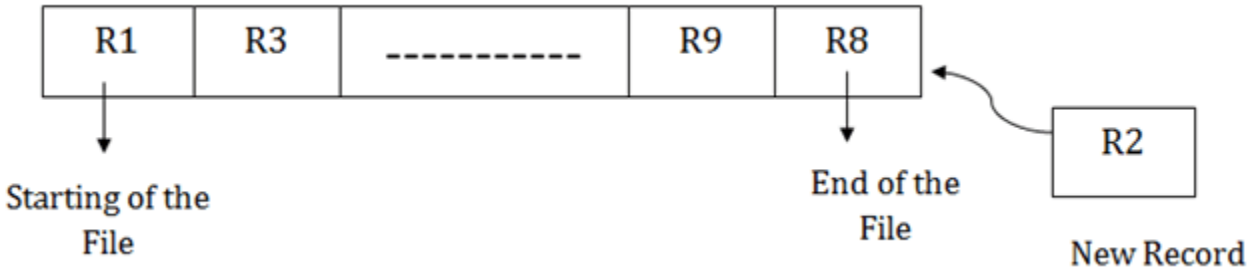
1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



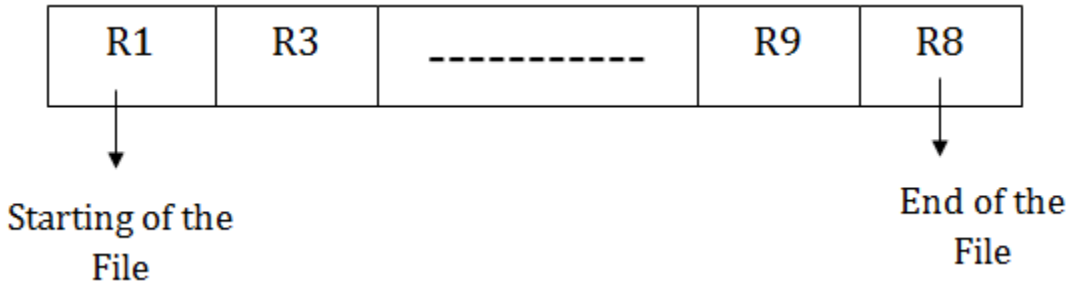
Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



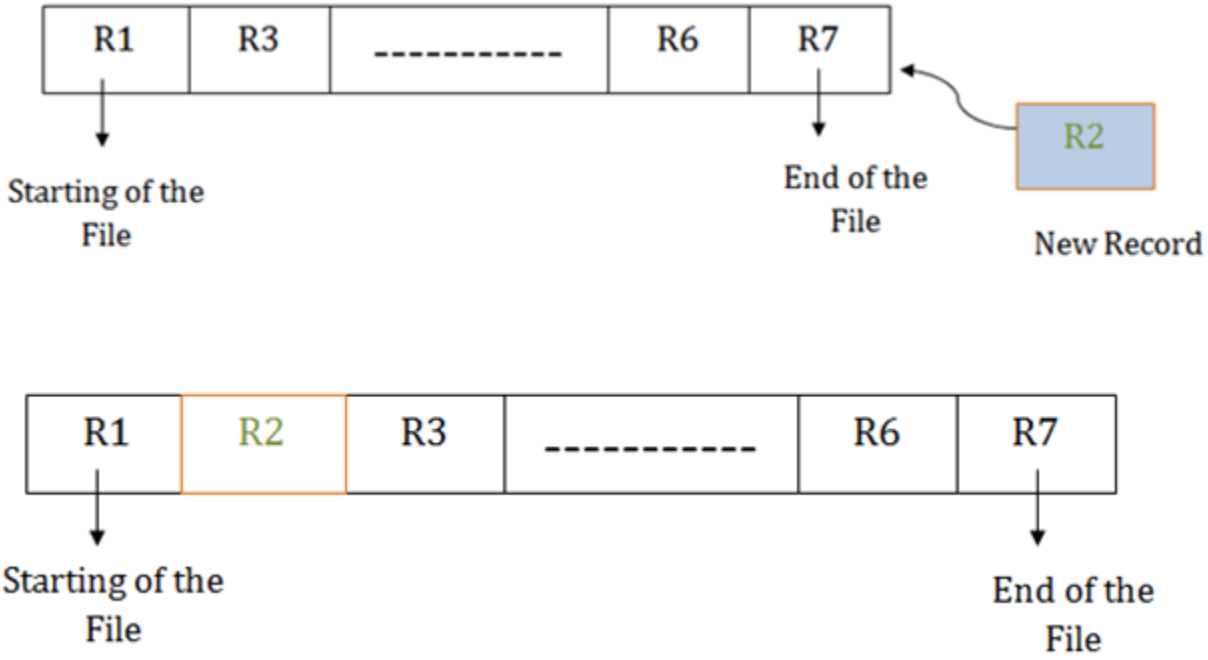
2. Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. 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.



Pros of sequential file organization

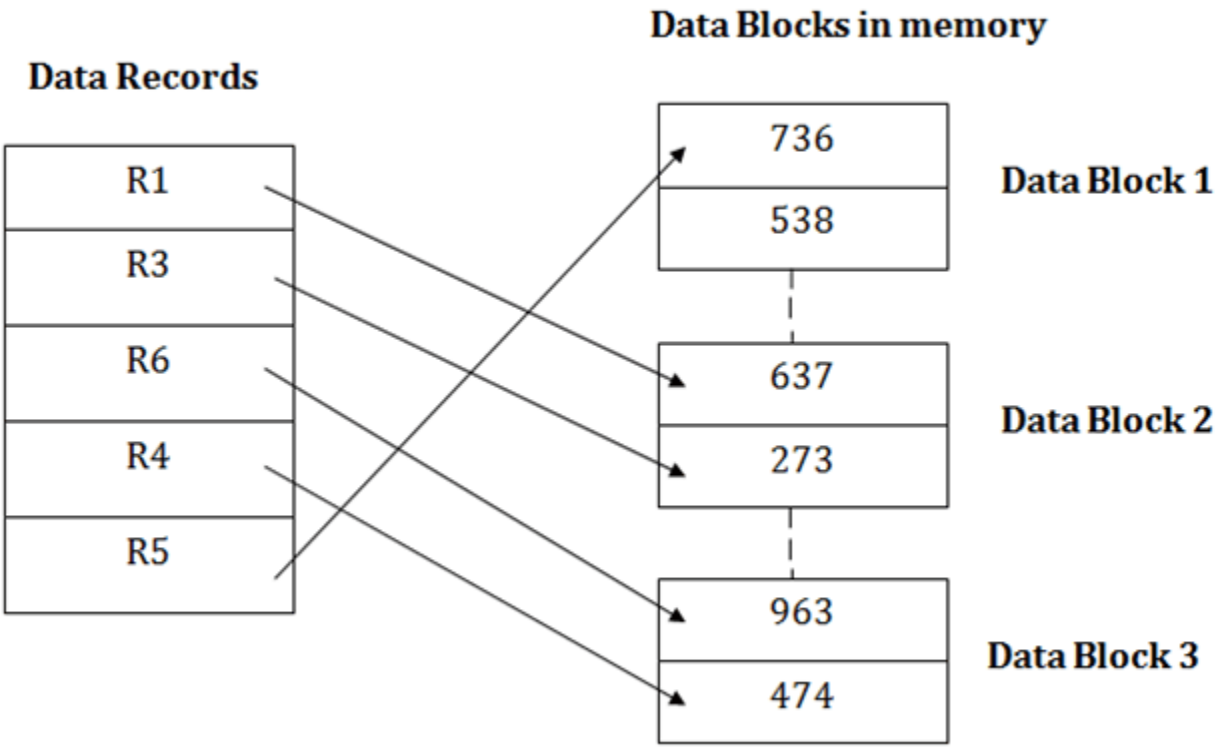
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

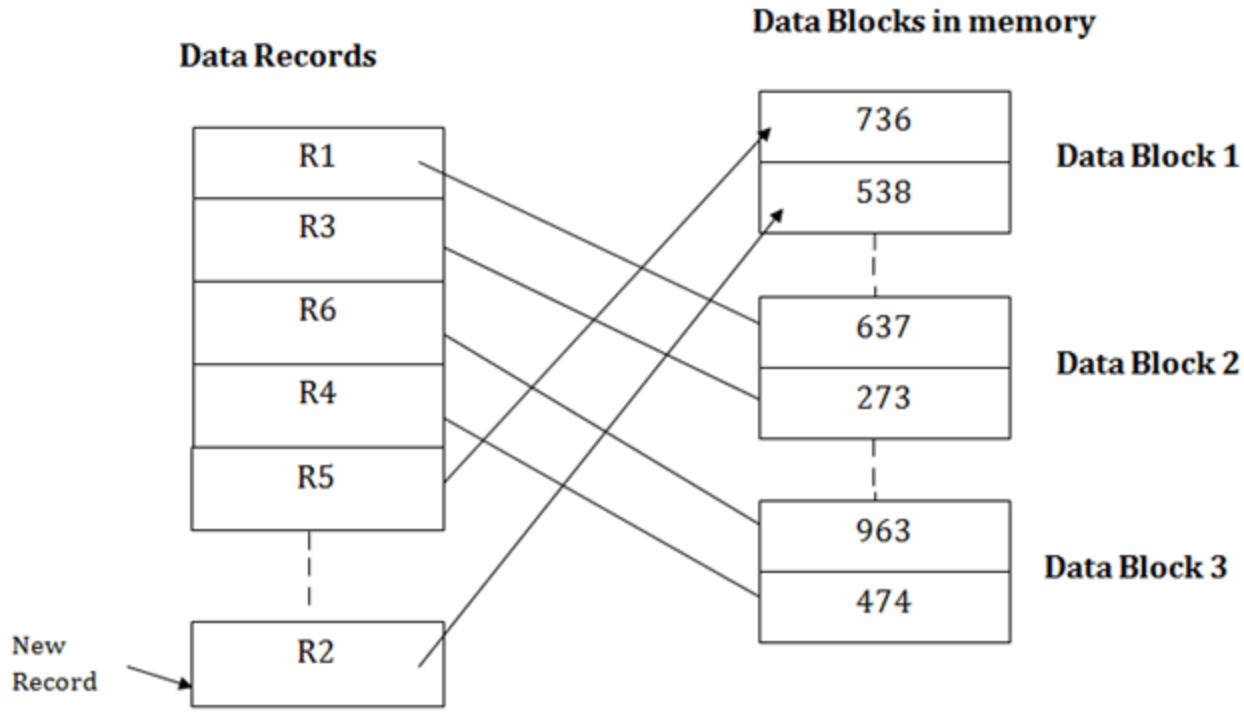
Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from staring of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

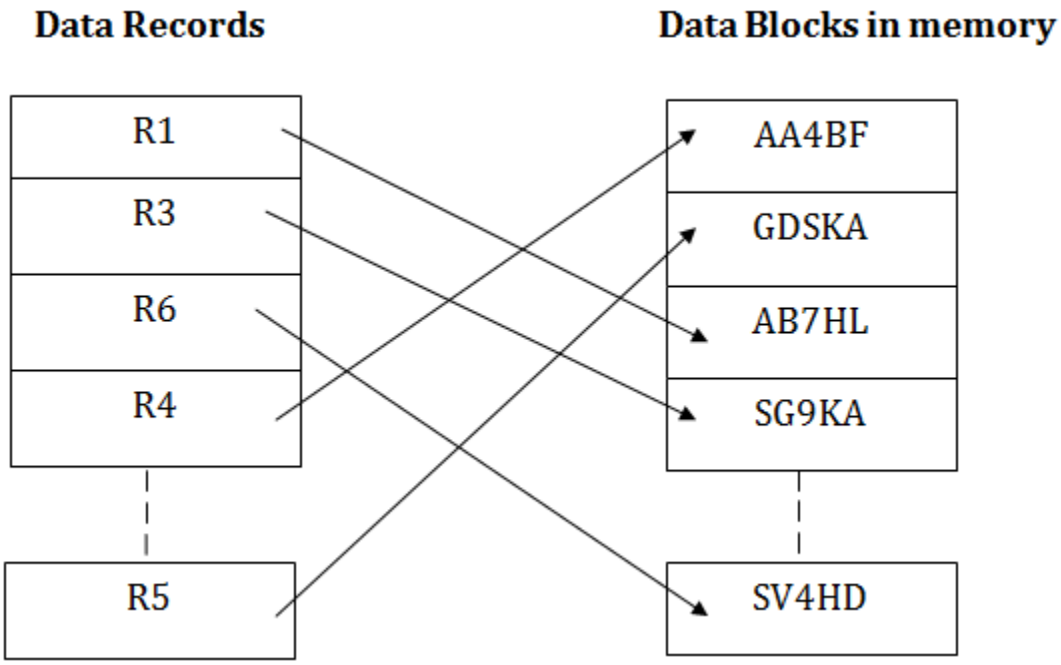
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

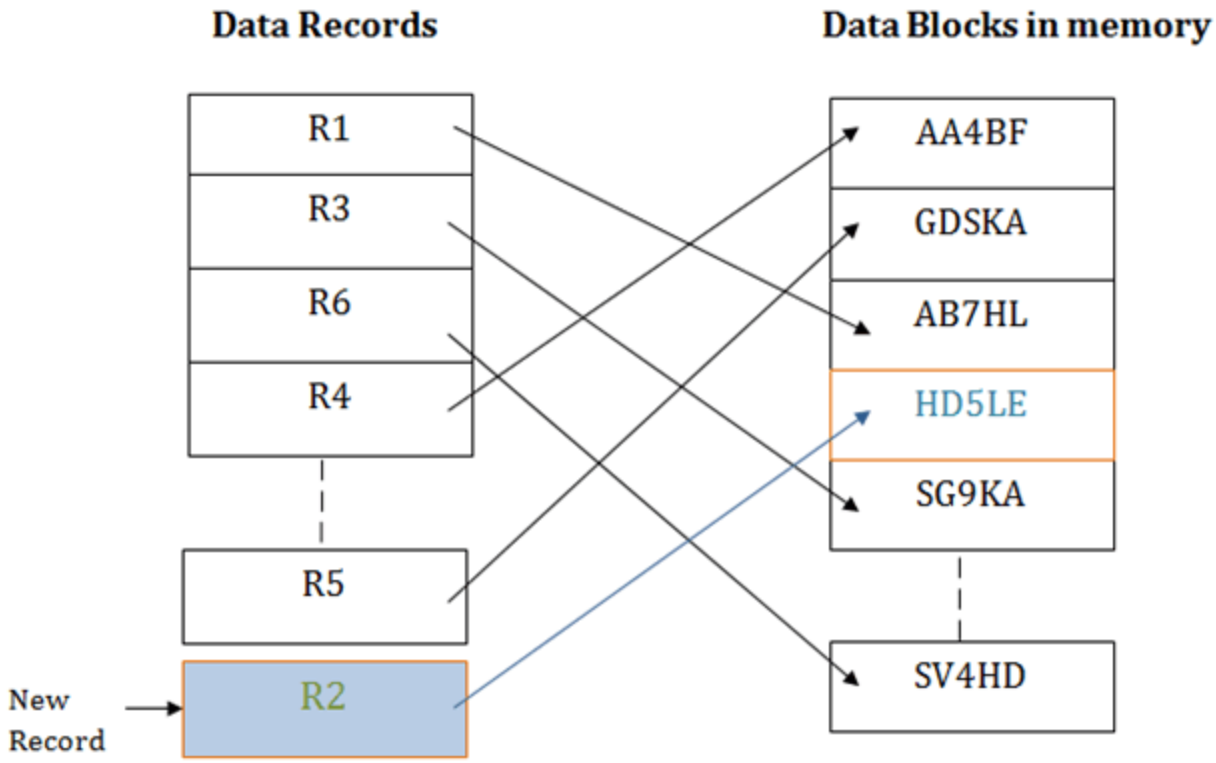
- This method is inefficient for the large database because it takes time to search or modify the record.
-
- This method is inefficient for large databases.

Hash File Organization

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

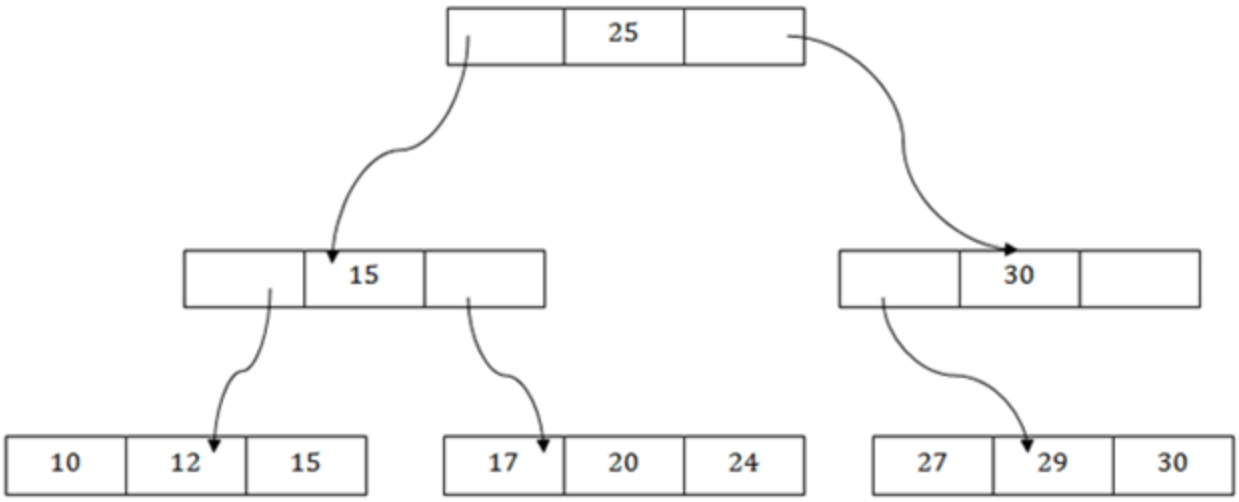


- When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.
- In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

Pros of B+ tree file organization

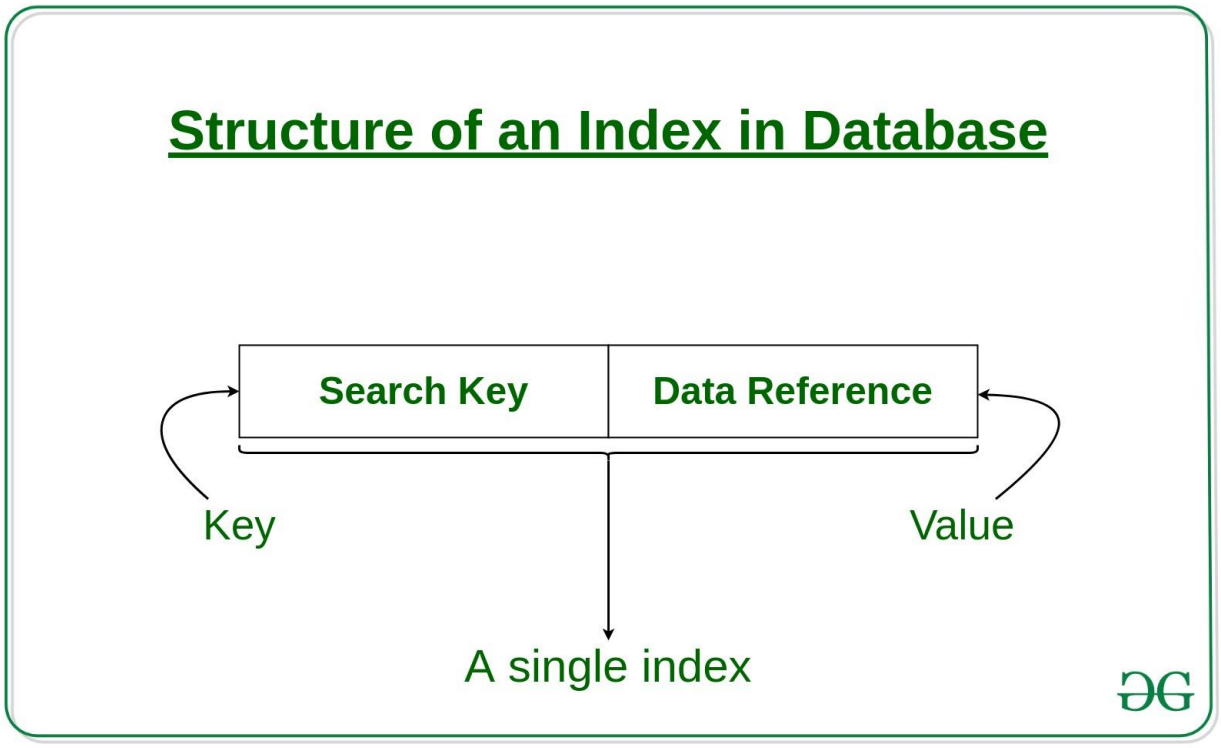
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

Cons of B+ tree file organization

- This method is inefficient for the static method.

Indexing in Databases

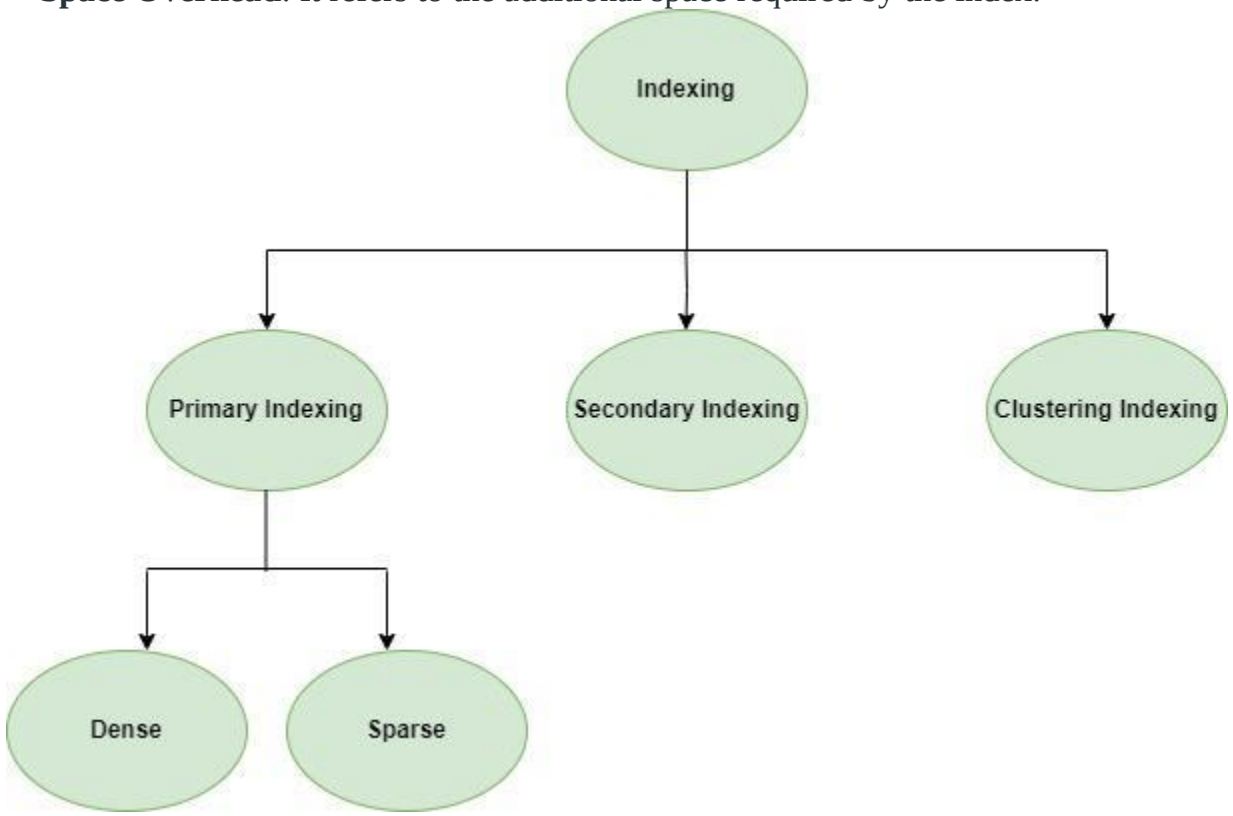
Indexing improves database performance by minimizing the number of disc visits required to fulfil a query. It is a data structure technique used to locate and quickly access data in databases. Several database fields are used to generate indexes. The main key or candidate key of the table is duplicated 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.



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 particular data element or set of elements.
- **Insertion Time**: It refers to the time taken to find the appropriate space and insert a 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.



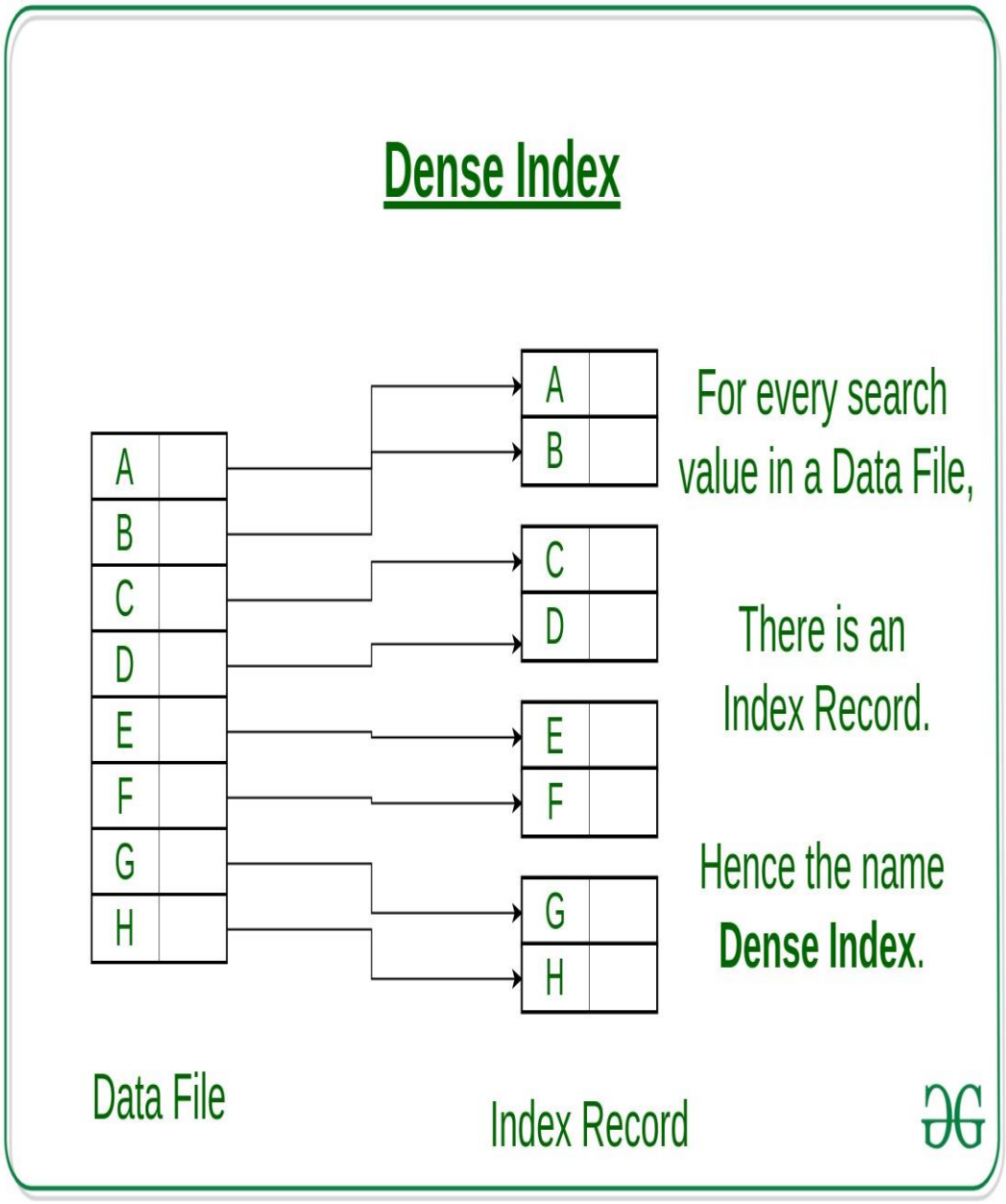
Basic diagram of Indexing in DBMS

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

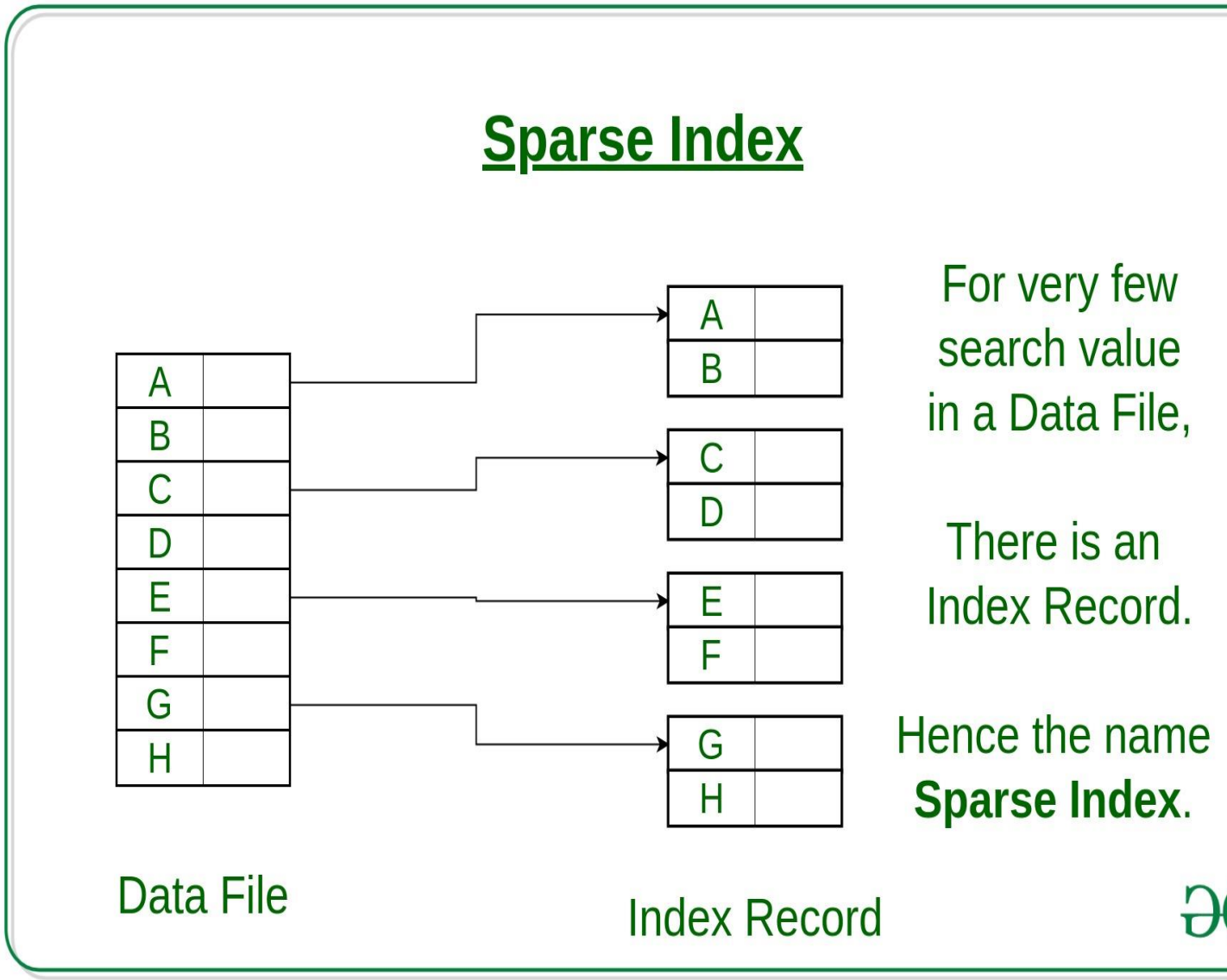
Sequential File Organization or Ordered Index File

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 organization 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.



- **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)

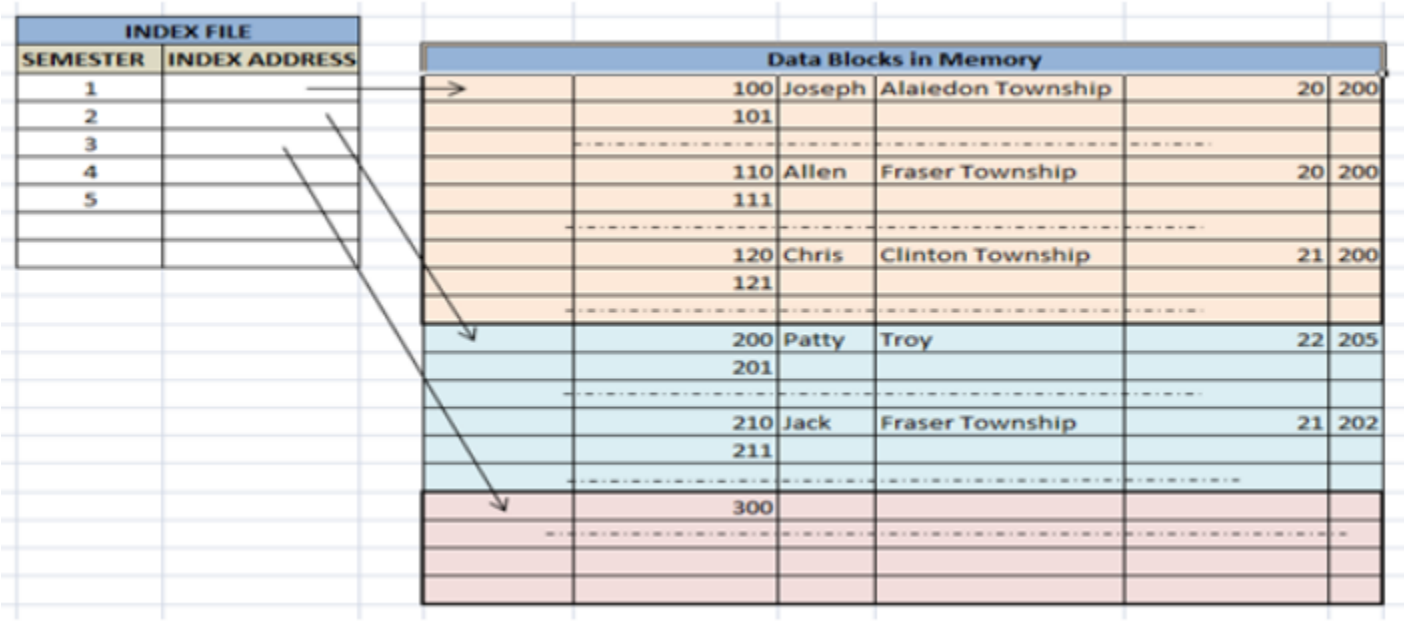


Hash File organization

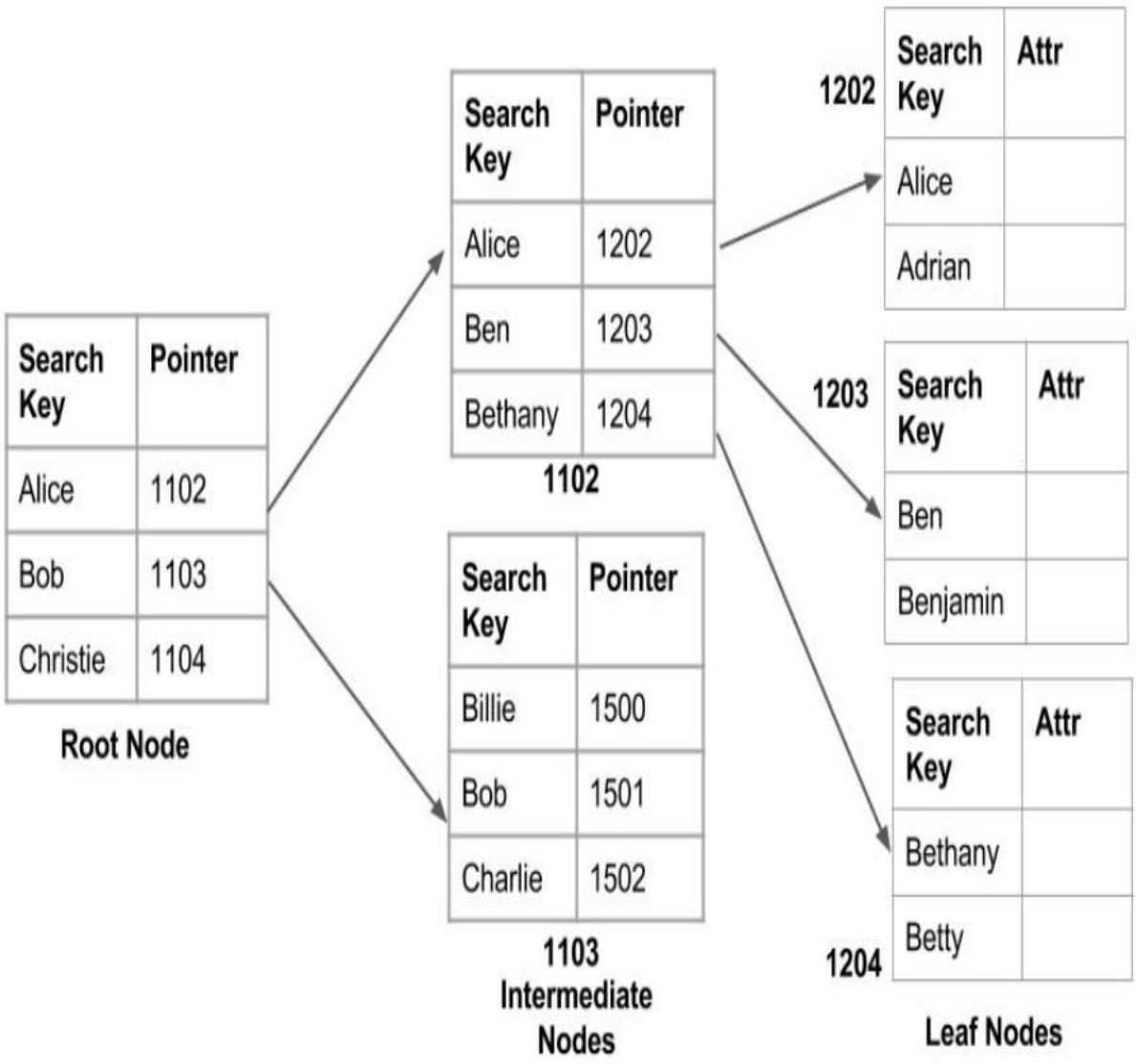
Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by a function called a hash function. There are primarily three methods of indexing:

- **Clustered Indexing**
  When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joining of more than two tables (records). 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 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 in each semester, for example, are grouped together. First semester students, second semester students, third semester students, and so on are categorized.



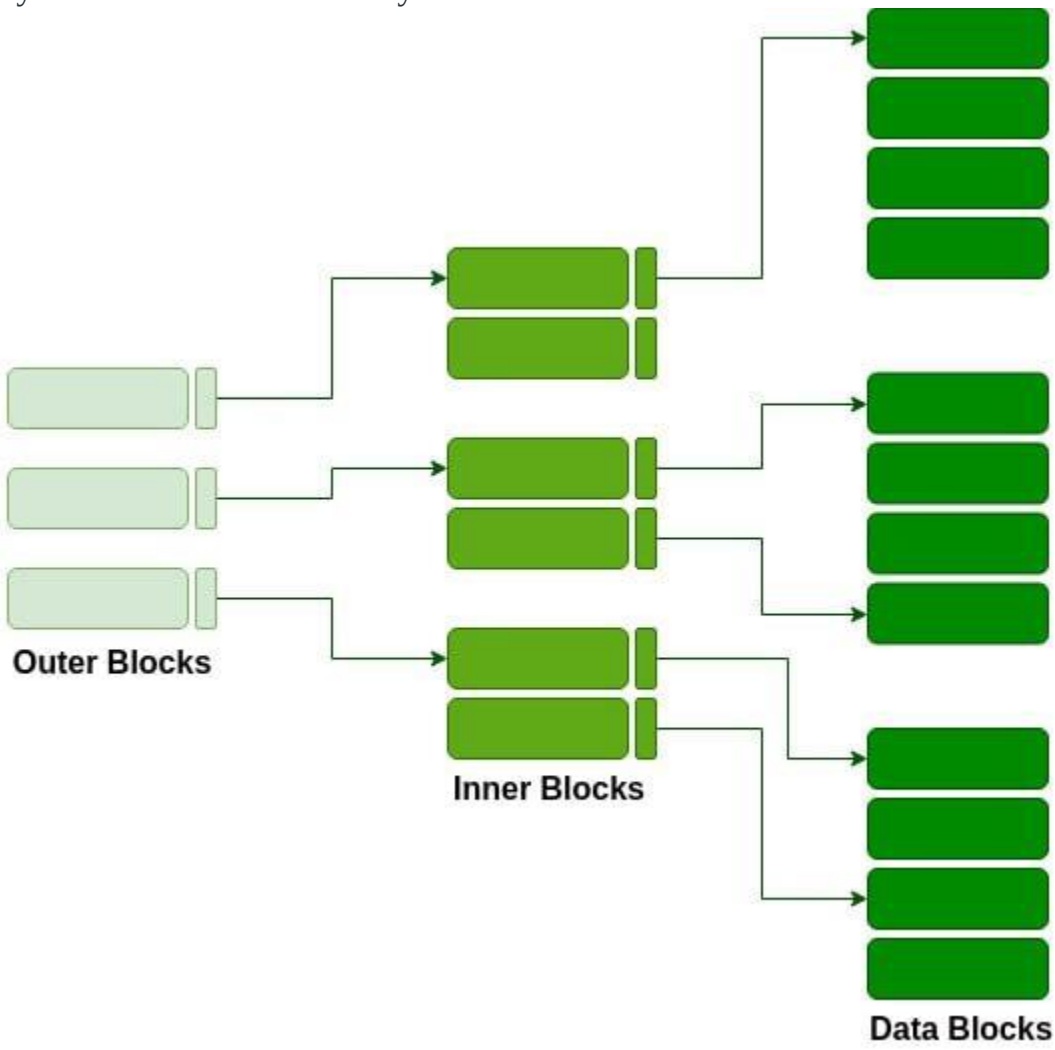
- **Primary Indexing**
  This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and 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 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, minimising 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 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 as 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 reorganisation, can decrease fragmentation.

Definition and Overview of ODBMS

The **ODBMS** which is an abbreviation for **object-oriented database management system** is the data model in which data is stored in form of objects, which are instances of classes. These classes and objects together make an object-oriented data model.

Components of Object-Oriented **Data Model:**

The OODBMS is based on three major components, namely: Object structure, Object classes, and Object identity. These are explained below.

1. Object Structure:

The structure of an object refers to the properties that an object is made up of. These properties of an object are referred to as an attribute. Thus, an object is a real-world entity with certain attributes that makes up the object structure. Also, an object encapsulates the data code into a single unit which in turn provides data abstraction by hiding the implementation details from the user.

The object structure is further composed of three types of components: Messages, Methods, and Variables. These are explained below.

1. **Messages –**
   A message provides an interface or acts as a communication medium between an object and the outside world. A message can be of two types:
   - **Read-only message:** If the invoked method does not change the value of a variable, then the invoking message is said to be a read-only message.
   - **Update message:** If the invoked method changes the value of a variable, then the invoking message is said to be an update message.

2. **Methods –**
   When a message is passed then the body of code that is executed is known as a method. Whenever a method is executed, it returns a value as output. A method can be of two types:
   - **Read-only method:** When the value of a variable is not affected by a method, then it is known as the read-only method.
   - **Update-method:** When the value of a variable change by a method, then it is known as an update method.

3. **Variables –**
   It stores the data of an object. The data stored in the variables makes the object distinguishable from one another.

2. Object Classes:
An object which is a real-world entity is an instance of a class. Hence first we need to define a class and then the objects are made which differ in the values they store but share the same class definition. The objects in turn correspond to various messages and variables stored in them.
Example –
class CLERK

```
{ //variables
   char name;
   string address;
   int id;
   int salary;

  //Messages
   char get_name();
   string get_address();
   int annual_salary();
};
```

In the above example, we can see, CLERK is a class that holds the object variables and messages.

An OODBMS also supports inheritance in an extensive manner as in a database there may be many classes with similar methods, variables and messages. Thus, the concept of the class hierarchy is maintained to depict the similarities among various classes.

The concept of encapsulation that is the data or information hiding is also supported by an object-oriented data model. And this data model also provides the facility of abstract data types apart from the built-in data types like char, int, float. ADT's are the user-defined data types that hold the values within them and can also have methods attached to them.

Thus, OODBMS provides numerous facilities to its users, both built-in and user-defined. It incorporates the properties of an object-oriented data model with a database management system, and supports the concept of programming paradigms like classes

and objects along with the support for other concepts like encapsulation, inheritance, and the user-defined ADT's (abstract data types).

ODBMS stands for Object-Oriented Database Management System, which is a type of database management system that is designed to store and manage object-oriented data. Object-oriented data is data that is represented using objects, which encapsulate data and behavior into a single entity.

An ODBMS stores and manages data as objects, and provides mechanisms for querying, manipulating, and retrieving the data. In an ODBMS, the data is typically stored in the form of classes and objects, which can be related to each other using inheritance and association relationships.

In an ODBMS, the data is managed using an object-oriented programming language or a specialized query language designed for object-oriented databases. Some of the popular object-oriented database languages include Smalltalk, Java, and C++. Some ODBMS also support standard SQL for querying the data.

ODBMS have several advantages over traditional relational databases. One of the main advantages is that they provide a natural way to represent complex data structures and relationships. Since the data is represented using objects, it can be easier to model real-world entities in the database. Additionally, ODBMS can provide better performance and scalability for applications that require a large number of small, complex transactions.

However, there are also some disadvantages to using an ODBMS. One of the main disadvantages is that they can be more complex and harder to use than traditional relational databases. Additionally, ODBMS may not be as widely used and supported as traditional relational databases, which can make it harder to find expertise and support. Finally, some applications may not require the advanced features and performance provided by an ODBMS, and may be better suited for a simpler database solution

Features of ODBMS:

Object-oriented data model: ODBMS uses an object-oriented data model to store and manage data. This allows developers to work with data in a more natural way, as objects are similar to the objects in the programming language they are using.
Complex data types: ODBMS supports complex data types such as arrays, lists, sets, and graphs, allowing developers to store and manage complex data structures in the database.
Automatic schema management: ODBMS automatically manages the schema of the database, as the schema is defined by the classes and objects in the application code. This

eliminates the need for a separate schema definition language and simplifies the development process.
High performance: ODBMS can provide high performance, especially for applications that require complex data access patterns, as objects can be retrieved with a single query.
Data integrity: ODBMS provides strong data integrity, as the relationships between objects are maintained by the database. This ensures that data remains consistent and correct, even in complex applications.
Concurrency control: ODBMS provides concurrency control mechanisms that ensure that multiple users can access and modify the same data without conflicts.
Scalability: ODBMS can scale horizontally by adding more servers to the database cluster, allowing it to handle large volumes of data.
Support for transactions: ODBMS supports transactions, which ensure that multiple operations on the database are atomic and consistent.

Advantages:

Supports Complex Data Structures: ODBMS is designed to handle complex data structures, such as inheritance, polymorphism, and encapsulation. This makes it easier to work with complex data models in an object-oriented programming environment.
Improved Performance: ODBMS provides improved performance compared to traditional relational databases for complex data models. ODBMS can reduce the amount of mapping and translation required between the programming language and the database, which can improve performance.
Reduced Development Time: ODBMS can reduce development time since it eliminates the need to map objects to tables and allows developers to work directly with objects in the database.
Supports Rich Data Types: ODBMS supports rich data types, such as audio, video, images, and spatial data, which can be challenging to store and retrieve in traditional relational databases.
Scalability: ODBMS can scale horizontally and vertically, which means it can handle larger volumes of data and can support more users.

Disadvantages:

Limited Adoption: ODBMS is not as widely adopted as traditional relational databases, which means it may be more challenging to find developers with experience working with ODBMS.
Lack of Standardization: ODBMS lacks standardization, which means that different vendors may implement different features and functionality.
Cost: ODBMS can be more expensive than traditional relational databases since it requires specialized software and hardware.

Integration with Other Systems: ODBMS can be challenging to integrate with other systems, such as business intelligence tools and reporting software.
Scalability Challenges: ODBMS may face scalability challenges due to the complexity of the data models it supports, which can make it challenging to partition data across multiple nodes.

ODBMS Full Form

ODBMS stands for **Object Database Management System**. In ODBMS data is encapsulated and represented in the form of objects. It relates the concept of object-oriented programming with database systems. ODBMS grew out of research during the early 1970s as database support for graph-structured objects. In comparison with RDBMS, where data is stored in tables with rows and columns, ODBMS stores information as objects.

Characteristics

- **Easy to link with programming language:** The programming language and the database schema use the same type definitions, so developers may not need to learn a new database query language.
- **No need for user defined keys:** Object Database Management Systems have an automatically generated OID associated with each of the objects.
- **Easy modeling:** ODBMS can easily model real-world objects, hence, are suitable for applications with complex data.
- **Can store non-textual data** ODBMS can also store audio, video and image data.

Advantages

- **Speed:** Access to data can be faster because an object can be retrieved directly without a search, by following pointers.
- **Improved performance:**These systems are most suitable for applications that use object oriented programming.
- **Extensibility:**Unlike traditional RDBMS where the basic-datatypes are hardcoded, when using ODBMS the user can encode any kind of data-structures to hold the data.
- **Data consistency:** When ODBMS is integrated with an object-based application, there is much greater consistency between the database and the programming language since both use the same model of representation for the data. This helps avoid the impedance mismatch.
- **Capability of handling variety of data:** Unlike other database management systems, ODBMS can also store nn textual data like-: images, videos and audios

Disadvantages

- **No universal standards:** There is no universally agreed standards of operating ODBMS This is the most significant drawback as the user is free to manipulate data model as he wants which can be an issue when handling enormous amounts of data.

- **No security features:**Since use of ODBMS is very limited, there are not adequate security features to store production-grade data.
- **Exponential increase in complexity:**ODBMS become very complex very fast. When there is a lot of data and a lot of relations between data, managing and optimising ODBMS becomes difficult.
- **Scalability:** Unable to support large systems.
- **Query optimization is challenging:** Optimising ODBMS queries requires complete information about the data like-: type and size of data. This compromises the data-encapsulation feature that ODBMS had to offer.

Distributed Database System in DBMS

A distributed database is essentially a database that is dispersed across numerous sites, i.e., on various computers or over a network of computers, and is not restricted to a single system. A distributed database system is spread across several locations with distinct physical components. This can be necessary when different people from all over the world need to access a certain database. It must be handled such that, to users, it seems to be a single database.

Types:

1. **Homogeneous Database:** A homogeneous database stores data uniformly across all locations. All sites utilize the same operating system, database management system, and data structures. They are therefore simple to handle.

2. **Heterogeneous Database:** With a heterogeneous distributed database, many locations may employ various software and schema, which may cause issues with queries and transactions. Moreover, one site could not be even aware of the existence of the other sites. Various operating systems and database applications may be used by various machines. They could even employ separate database data models. Translations are therefore necessary for communication across various sites.

Data may be stored on several places in two ways using distributed data storage:

1. **Replication -** With this strategy, every aspect of the connection is redundantly kept at two or more locations. It is a completely redundant database if the entire database is accessible from every location. Systems preserve copies of the data as a result of replication. This has advantages since it makes more data accessible at many locations. Moreover, query requests can now be handled in parallel. But, there are some drawbacks as well. Data must be updated often. All changes performed at one site must be documented at every site where that relation is stored in order to avoid inconsistent results. There is a tone of overhead here. Moreover, since concurrent

access must now be monitored across several sites, concurrency management becomes far more complicated.

2. **Fragmentation -** In this method, the relationships are broken up into smaller pieces and each fragment is kept in the many locations where it is needed. To ensure there is no data loss, the pieces must be created in a way that allows for the reconstruction of the original relation. As fragmentation doesn't result in duplicate data, consistency is not a concern.

Relationships can be fragmented in one of two ways:

- Separating the relation into groups of tuples using rows results in horizontal fragmentation, where each tuple is allocated to at least one fragment.
- Vertical fragmentation, also known as splitting by columns, occurs when a relation's schema is split up into smaller schemas. A common candidate key must be present in each fragment in order to guarantee a lossless join

Sometimes a strategy that combines fragmentation and replication is employed.

Uses for distributed databases

- The corporate management information system makes use of it.
- Multimedia apps utilize it.
- Used in hotel chains, military command systems, etc.
- The production control system also makes use of it

Characteristics of distributed databases

Distributed databases are logically connected to one another when they are part of a collection, and they frequently form a single logical database. Data is physically stored across several sites and is separately handled in distributed databases. Each site's processors are connected to one another through a network, but they are not set up for multiprocessing.

A widespread misunderstanding is that a distributed database is equivalent to a loosely coupled file system. It's considerably more difficult than that in reality. Although distributed databases use transaction processing, they are not the same as systems that use it.

Generally speaking, distributed databases have the following characteristics:

- Place unrelated
- Spread-out query processing

- The administration of distributed transactions
- Independent of hardware
- Network independent of operating systems
- Transparency of transactions
- DBMS unrelated<

Architecture for a distributed database

Both homogeneous and heterogeneous distributed databases exist.

All of the physical sites in a homogeneous distributed database system use the same operating system and database software, as well as the same underlying hardware. It can be significantly simpler to build and administer homogenous distributed database systems since they seem to the user as a single system. The data structures at each site must either be the same or compatible for a distributed database system to be considered homogeneous. Also, the database program utilized at each site must be compatible or same.

The hardware, operating systems, or database software at each site may vary in a heterogeneous distributed database. Although separate sites may employ various technologies and schemas, a variation in schema might make query and transaction processing challenging.

Various nodes could have dissimilar hardware, software, and data structures, or they might be situated in incompatible places. Users may be able to access data stored at a different place but not upload or modify it. Because heterogeneous distributed databases are sometimes challenging to use, many organizations find them to be economically unviable.

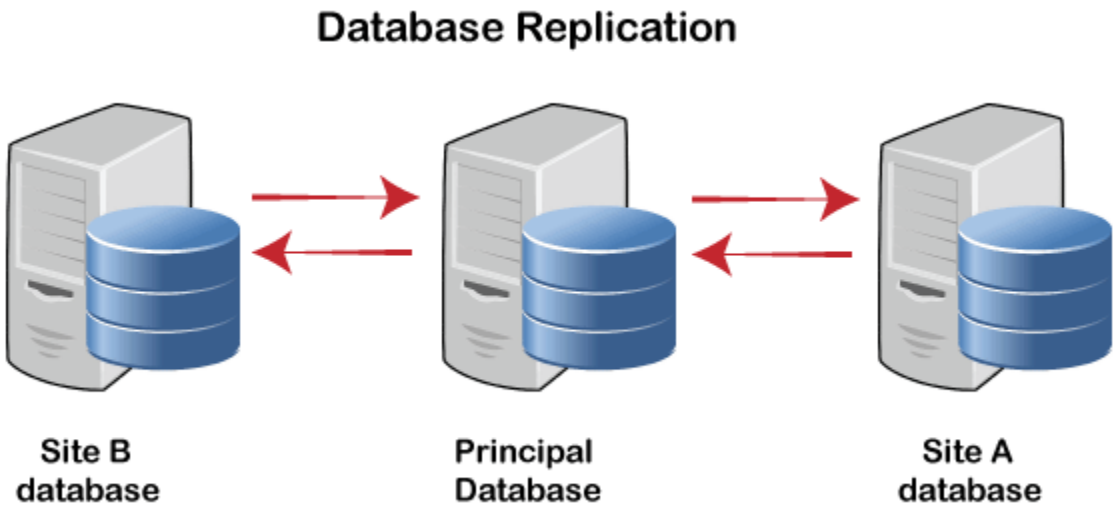
Distributed databases' benefits

Using distributed databases has a lot of benefits.

- As distributed databases provide modular development, systems may be enlarged by putting new computers and local data in a new location and seamlessly connecting them to the distributed system.
- With centralized databases, failures result in a total shutdown of the system. Distributed database systems, however, continue to operate with lower performance when a component fails until the issue is resolved.
- If the data is near to where it is most often utilized, administrators can reduce transmission costs for distributed database systems. Centralized systems are unable to accommodate this<

Types of Distributed Database

- Data instances are created in various areas of the database using replicated data. Distributed databases may access identical data locally by utilizing duplicated data, which reduces bandwidth. Read-only and writable data are the two types of replicated data that may be distinguished.
- Only the initial instance of replicated data can be changed in read-only versions; all subsequent corporate data replications are then updated. Data that is writable can be modified, but only the initial occurrence is affected.



- Primary keys that point to a single database record are used to identify horizontally fragmented data. Horizontal fragmentation is typically used when business locations only want access to the database for their own branch.
- Using primary keys that are duplicates of each other and accessible to each branch of the database is how vertically fragmented data is organized. When a company's branch and central location deal with the same accounts differently, vertically fragmented data is used.
- Data that has been edited or modified for decision support databases is referred to as reorganised data. When two distinct systems are managing transactions and decision support, reorganised data is generally utilised. When there are numerous requests, online transaction processing must be reconfigured, and decision support systems might be challenging to manage.
- In order to accommodate various departments and circumstances, separate schema data separates the database and the software used to access it. Often, there is overlap between many databases and separate schema data

Distributed database examples

- Apache Ignite, Apache Cassandra, Apache HBase, Couchbase Server, Amazon SimpleDB, Clusterpoint, and FoundationDB are just a few examples of the numerous distributed databases available.
- Large data sets may be stored and processed with Apache Ignite across node clusters. GridGain Systems released Ignite as open source in 2014, and it was later approved into the Apache Incubator program. RAM serves as the database's primary processing and storage layer in Apache Ignite.
- Apache Cassandra has its own query language, Cassandra Query Language, and it supports clusters that span several locations (CQL). Replication tactics in Cassandra may also be customized.
- Apache HBase offers a fault-tolerant mechanism to store huge amounts of sparse data on top of the Hadoop Distributed File System. Moreover, it offers per-column Bloom filters, in-memory execution, and compression. Although Apache Phoenix offers a SQL layer for HBase, HBase is not meant to replace SQL databases.
- An interactive application that serves several concurrent users by producing, storing, retrieving, aggregating, altering, and displaying data is best served by Couchbase Server, a NoSQL software package. Scalable key value and JSON document access is provided by Couchbase Server to satisfy these various application demands.
- Along with Amazon S3 and Amazon Elastic Compute Cloud, Amazon SimpleDB is utilised as a web service. Developers may request and store data with Amazon SimpleDB with a minimum of database maintenance and administrative work.
- Relational database designs' complexity, scalability problems, and performance restrictions are all eliminated with Clusterpoint. Open APIs are used to handle data in the XLM or JSON formats. Clusterpoint does not have the scalability or performance difficulties that other relational database systems experience since it is a schema-free document database.