Storage System in DBMS

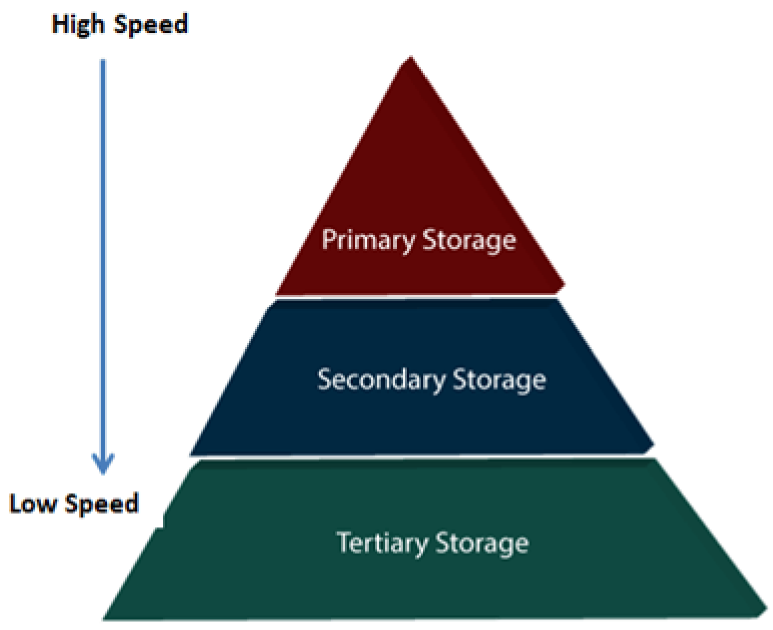
A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices.

In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

Types of Data Storage

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

- Primary Storage
- Secondary Storage
- Tertiary Storage



Primary Storage

It is the primary area that offers quick access to the stored data. We also know the primary storage as volatile storage. It is because this type of memory does not permanently store the data. As soon as the system leads to a power cut or a crash, the data also get lost. Main memory and cache are the types of primary storage.

- **Main Memory:** It is the one that is responsible for operating the data that is available by the storage medium. The main memory handles each instruction of a computer machine. This type of memory can store gigabytes of data on a system but is small enough to carry the entire database. At last, the main memory loses the whole content if the system shuts down because of power failure or other reasons.

1. **Cache:** It is one of the costly storage media. On the other hand, it is the fastest one. A cache is a tiny storage media which is maintained by the computer hardware usually. While designing the algorithms and query processors for the data structures, the designers keep concern on the cache effects.

Secondary Storage

Secondary storage is also called as Online storage. It is the storage area that allows the user to save and store data permanently. This type of memory does not lose the data due to any power failure or system crash. That's why we also call it non-volatile storage.

There are some commonly described secondary storage media which are available in almost every type of computer system:

- **Flash Memory:** A flash memory stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system. These USB keys help transfer data to a computer system, but it varies in size limits. Unlike the main memory, it is possible to get back the stored data which may be lost due to a power cut or other reasons. This type of memory storage is most commonly used in the server systems for caching the frequently used data. This leads the systems towards high performance and is capable of storing large amounts of databases than the main memory.
- **Magnetic Disk Storage:** This type of storage media is also known as online storage media. A magnetic disk is used for storing the data for a long time. It is capable of storing an entire database. It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing. Also, if the system performs any operation over the data, the modified data should be written back to the disk. The tremendous capability of a magnetic disk is that it does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data.

Tertiary Storage

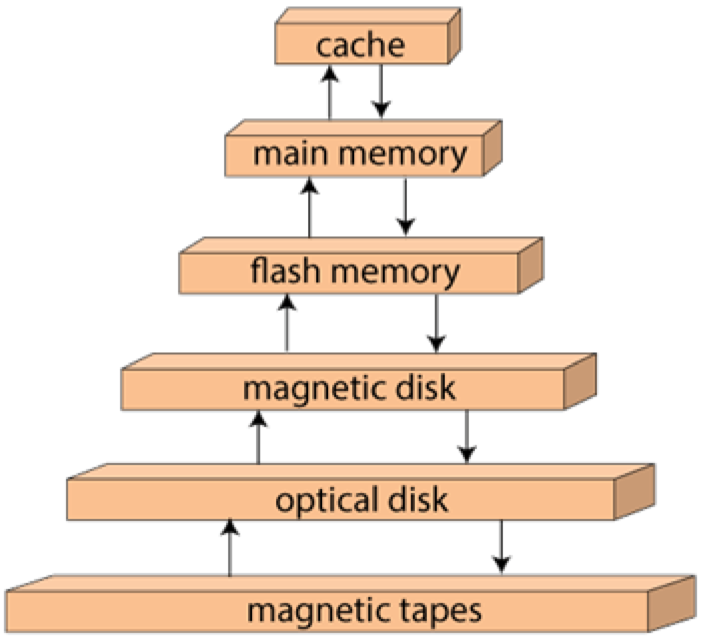
It is the storage type that is external from the computer system. It has the slowest speed. But it is capable of storing a large amount of data. It is also known as Offline storage. Tertiary storage is generally used for data backup. There are following tertiary storage devices available:

- **Optical Storage:** An optical storage can store megabytes or gigabytes of data. A Compact Disk (CD) can store 700 megabytes of data with a playtime of around 80 minutes. On the other hand, a Digital Video Disk or a DVD can store 4.7 or 8.5 gigabytes of data on each side of the disk.
- **Tape Storage:** It is the cheapest storage medium than disks. Generally, tapes are used for archiving or backing up the data. It provides slow access to data as it accesses data sequentially from the start. Thus, tape storage is also known as sequential-access storage. Disk storage is known as direct-access storage as we can directly access the data from any location on disk.

Storage Hierarchy

Besides the above, various other storage devices reside in the computer system. These storage media are organized on the basis of data accessing speed, cost per unit of data to buy the medium, and by medium's reliability. Thus, we can create a hierarchy of storage media on the basis of its cost and speed.

Thus, on arranging the above-described storage media in a hierarchy according to its speed and cost, we conclude the below-described image:

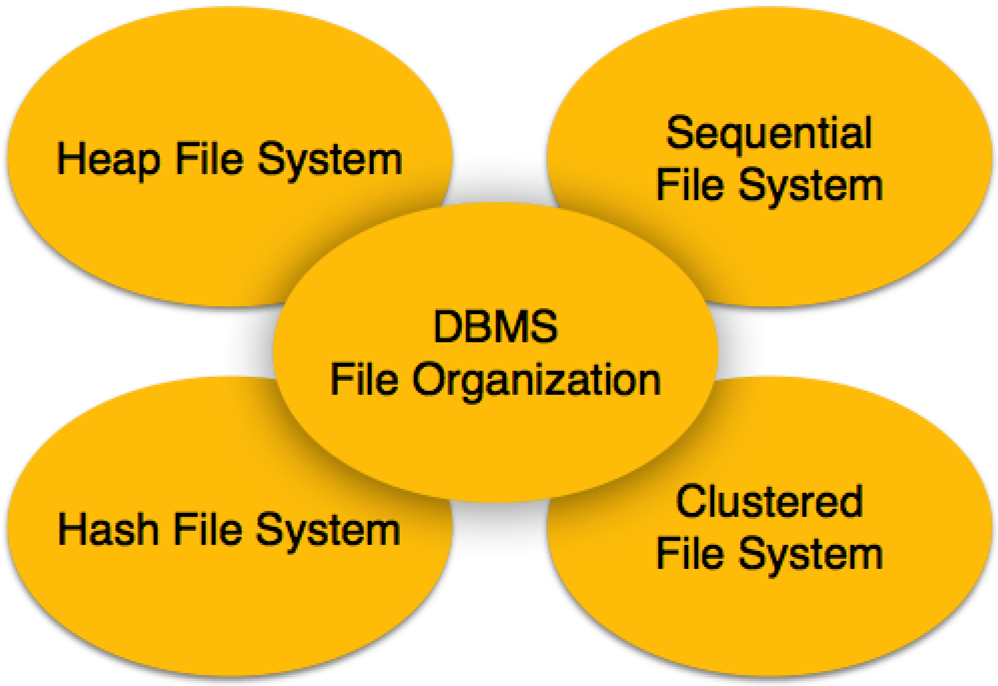


Storage device hierarchy

In the image, the higher levels are expensive but fast. On moving down, the cost per bit is decreasing, and the access time is increasing. Also, the storage media from the main memory to up represents the volatile nature, and below the main memory, all are non-volatile devices.

File Organization

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



Heap File Organization

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

Sequential File Organization

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

Hash File Organization

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

Clustered File Organization

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

File Operations

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

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

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

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

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

Indexing

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

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

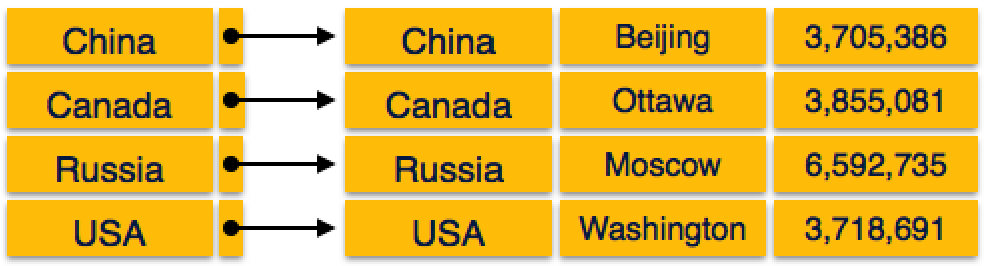
- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every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

Ordered Indexing is of two types –

- Dense Index
- Sparse Index

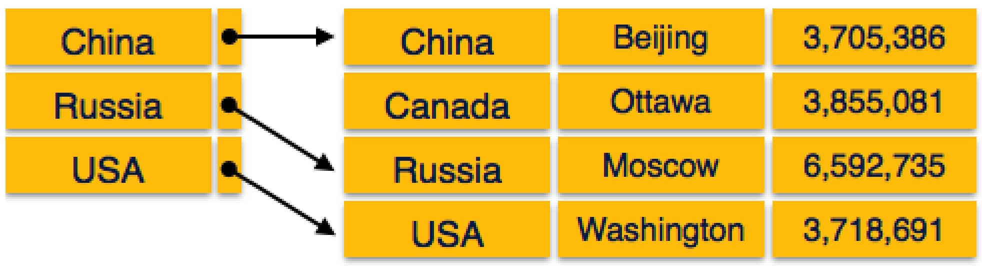
Dense Index

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



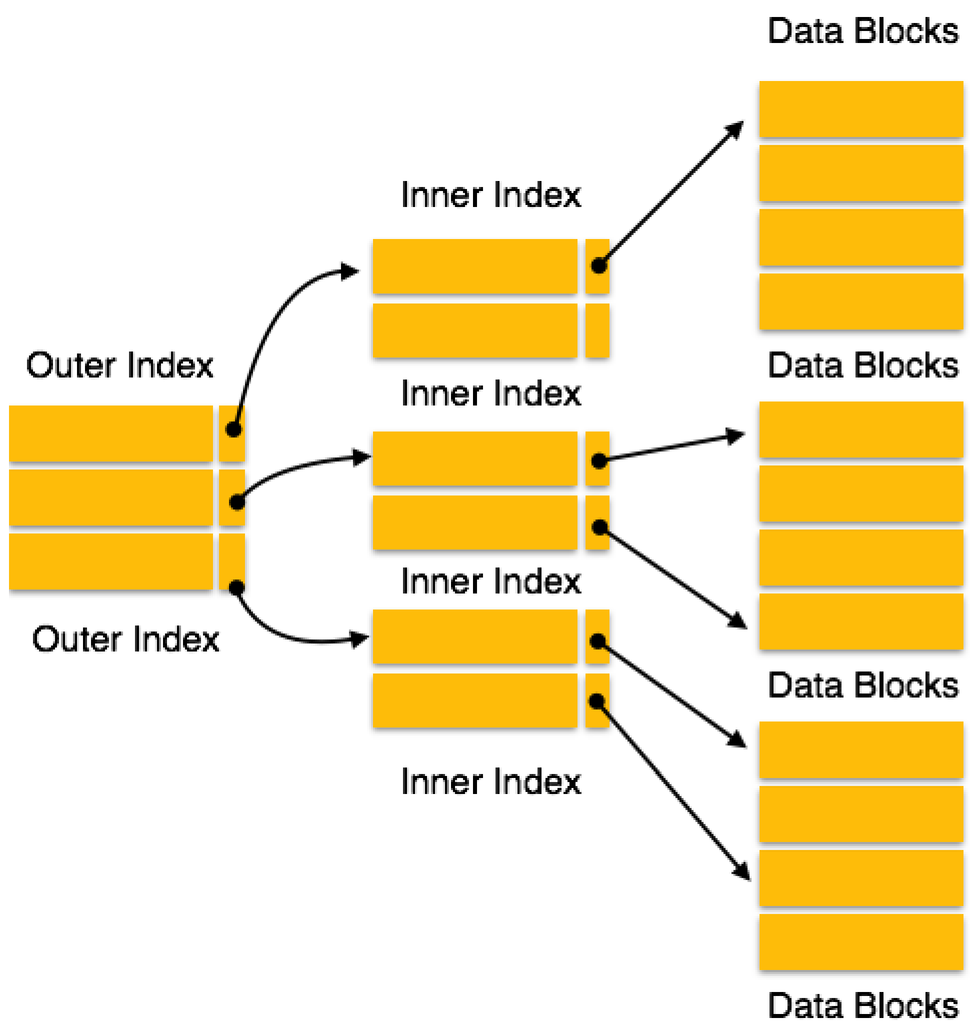
Sparse Index

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



Multilevel Index

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



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

B tree vs B+ tree

Before understanding **B tree** and **B+ tree** differences, we should know the B tree and B+ tree separately.

What is the B tree?

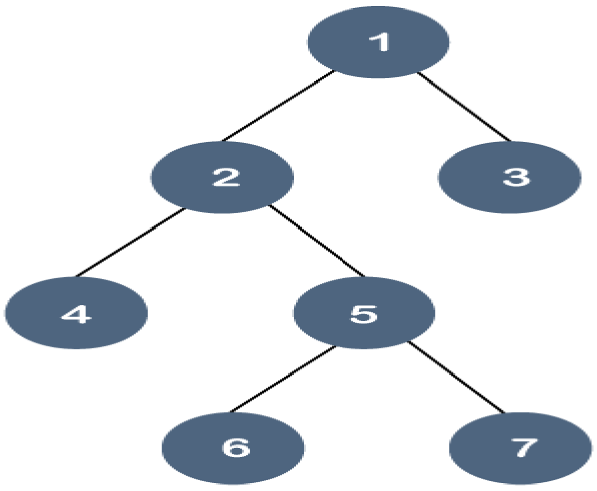
B tree is a self-balancing tree, and it is a m-way tree where m defines the order of the tree. **Btree** is a generalization of the Binary Search tree in which a node can have more than one key and more than two children depending upon the value of **m**. In the B tree, the data is specified in a sorted order having lower values on the left subtree and higher values in the right subtree.

Properties of B tree

The following are the properties of the B tree:

- In the B tree, all the leaf nodes must be at the same level, whereas, in the case of a binary tree, the leaf nodes can be at different levels.

Let's understand this property through an example.

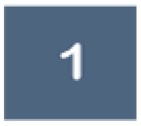


In the above tree, all the leaf nodes are not at the same level, but they have the utmost two children. Therefore, we can say that the above tree is a binary tree but not a B tree.

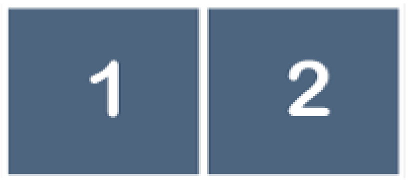
- If the Btree has an order of m, then each node can have a maximum of **m** In the case of minimum children, the leaf nodes have zero children, the root node has two children, and the internal nodes have a ceiling of m/2.
- Each node can have maximum (m-1) keys. For example, if the value of m is 5 then the maximum value of keys is 4.
- The root node has minimum one key, whereas all the other nodes except the root node have (ceiling of m/2 minus - 1) minimum keys.
- If we perform insertion in the B tree, then the node is always inserted in the leaf node.

Suppose we want to create a B tree of order 3 by inserting values from 1 to 10.

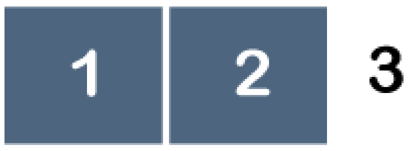
Step 1: First, we create a node with 1 value as shown below:



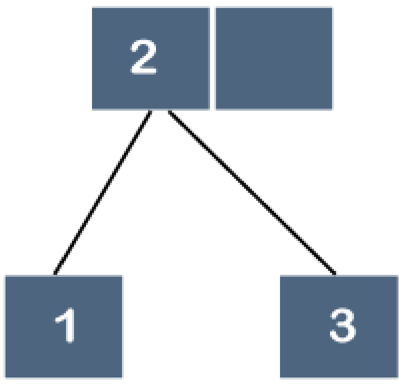
Step 2: The next element is 2, which comes after 1 as shown below:



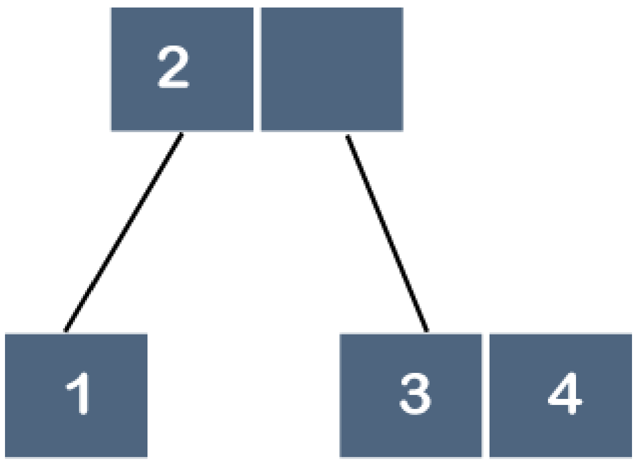
Step 3: The next element is 3, and it is inserted after 2 as shown below:



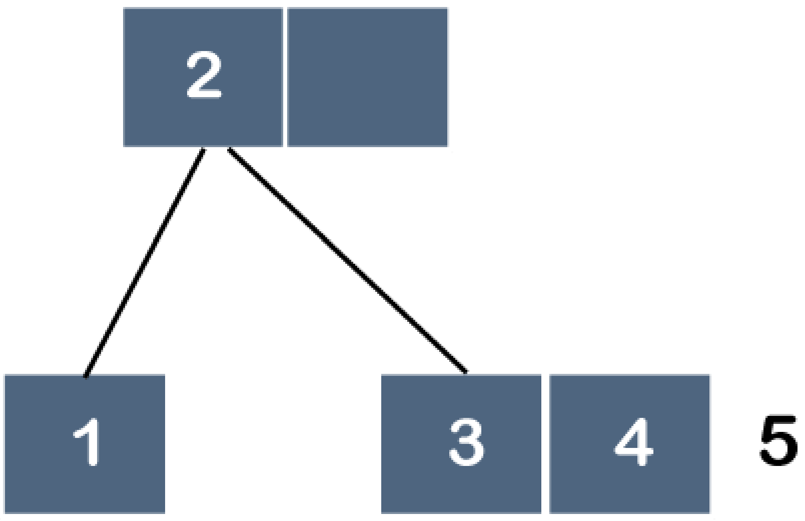
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 2, so it moves to its parent. The node 2 does not have any parent, so it will become the root node as shown below:



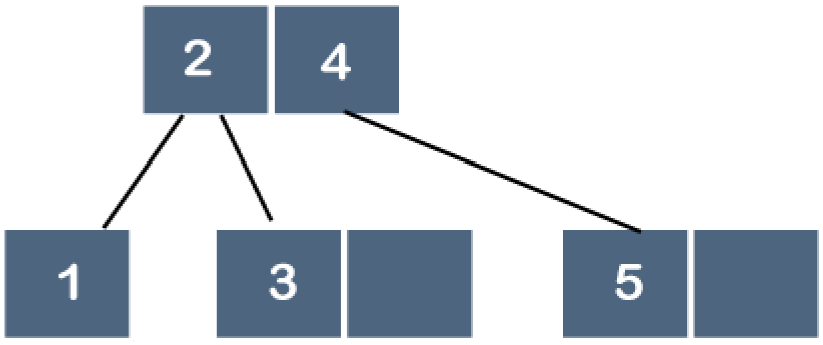
Step 4: The next element is 4. Since 4 is greater than 2 and 3, so it will be added after the 3 as shown below:



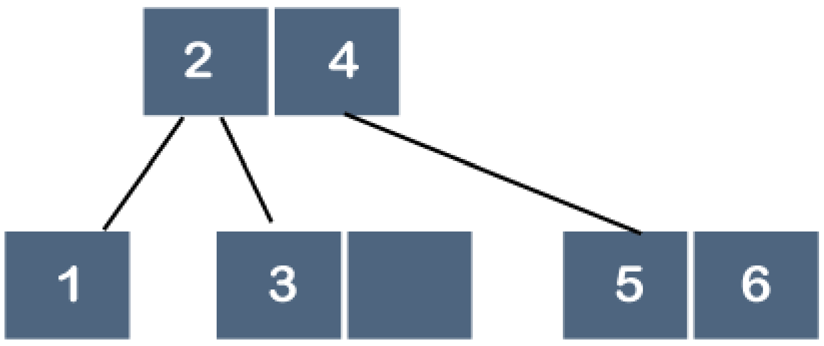
Step 5: The next element is 5. Since 5 is greater than 2, 3 and 4 so it will be added after 4 as shown below:



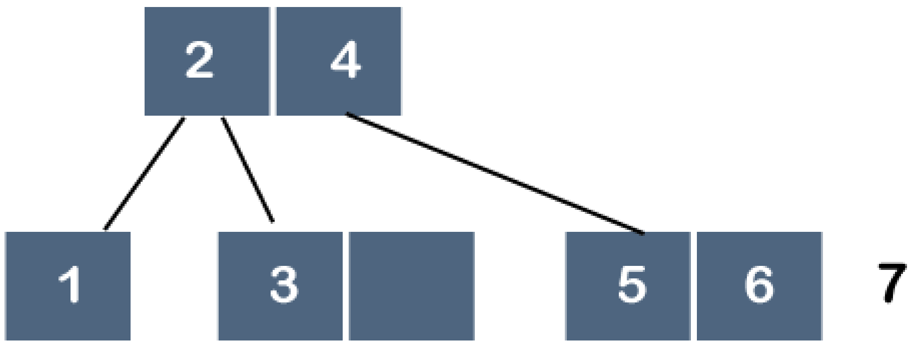
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. The parent is node 2; therefore, 4 will be added after 2 as shown below:



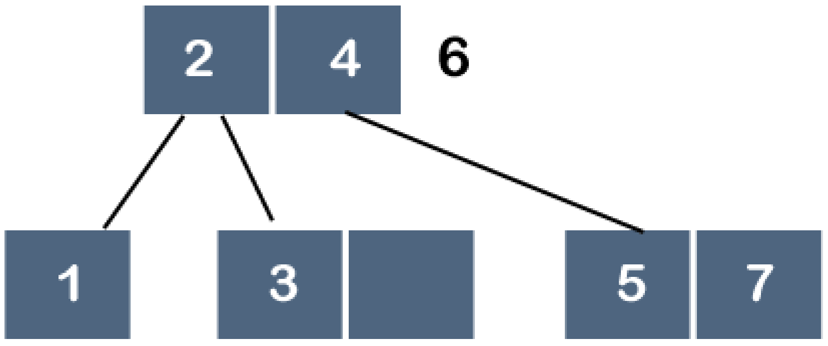
Step 6: The next element is 6. Since 6 is greater than 2, 4 and 5, so 6 will come after 5 as shown below:



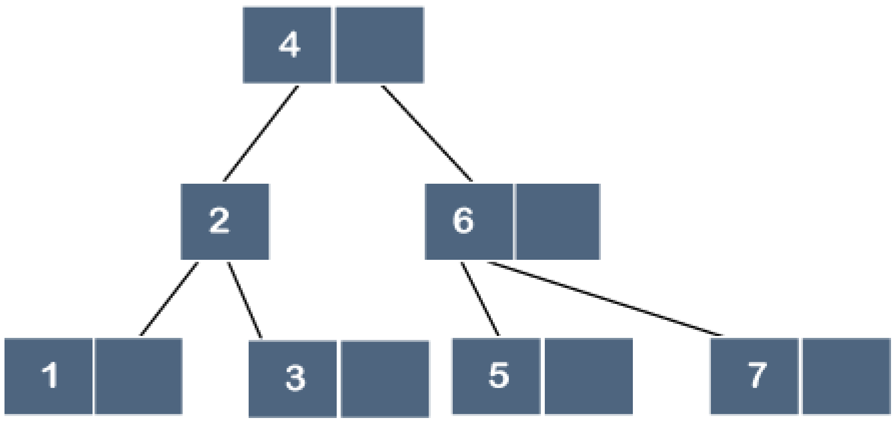
Step 7: The next element is 7. Since 7 is greater than 2, 4, 5 and 6, so 7 will come after 6 as shown below:



As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 6, so it moves to its parent as shown below:



But, 6 cannot be added after 4 because the node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its parent. As node 4 does not have any parent, node 4 will become a root node as shown below:



What is a B+ tree?

The B+ tree is also known as an advanced self-balanced tree because every path from the root of the tree to the leaf of the tree has the same length. Here, the same length means that all the leaf nodes occur at the same level. It will not happen that some of the leaf nodes occur at the third level and some of them at the second level.

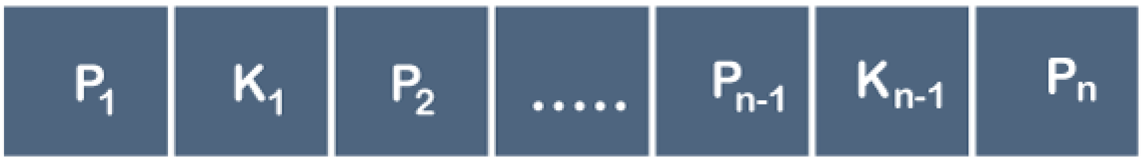
A B+ tree index is considered a multi-level index, but the B+ tree structure is not similar to the multi-level index sequential files.

Why is the B+ tree used?

A B+ tree is used to store the records very efficiently by storing the records in an indexed manner using the B+ tree indexed structure. Due to the multi-level indexing, the data accessing becomes faster and easier.

B+ tree Node Structure

The node structure of the B+ tree contains pointers and key values shown in the below figure:



As we can observe in the above B+ tree node structure that it contains n-1 key values (k_1 to k_{n-1}) and n pointers (p_1 to p_n).

The search key values which are placed in the node are kept in sorted order. Thus, if $i<j$ then $k_i<k_j$.

Constraint on various types of nodes

Let 'b' be the order of the B+ tree.

Non-Leaf node

Let 'm' represents the number of children of a node, then the relation between the order of the tree and the number of children can be represented as:

$$\left[\frac{b}{2}\right] \leq m \leq b$$

Let k represents the search key values. The relation between the order of the tree and search key can be represented as:

As we know that the number of pointers is equal to the search key values plus 1, so mathematically, it can be written as:

Number of Pointers (or children) = Number of Search keys + 1

Therefore, the maximum number of pointers would be 'b', and the minimum number of pointers would be the ceiling function of b/2.

Leaf Node

A leaf node is a node that occurs at the last level of the B+ tree, and each leaf node uses only one pointer to connect with each other to provide the sequential access at the leaf level.

In leaf node, the maximum number of children is:

$$\left[\frac{b}{2}\right] -1 \leq k \leq b-1$$

The maximum number of search keys is:

$$\left[\frac{b}{2}\right] -1 \leq m \leq b-1$$

Root Node

The maximum number of children in the case of the root node is: b

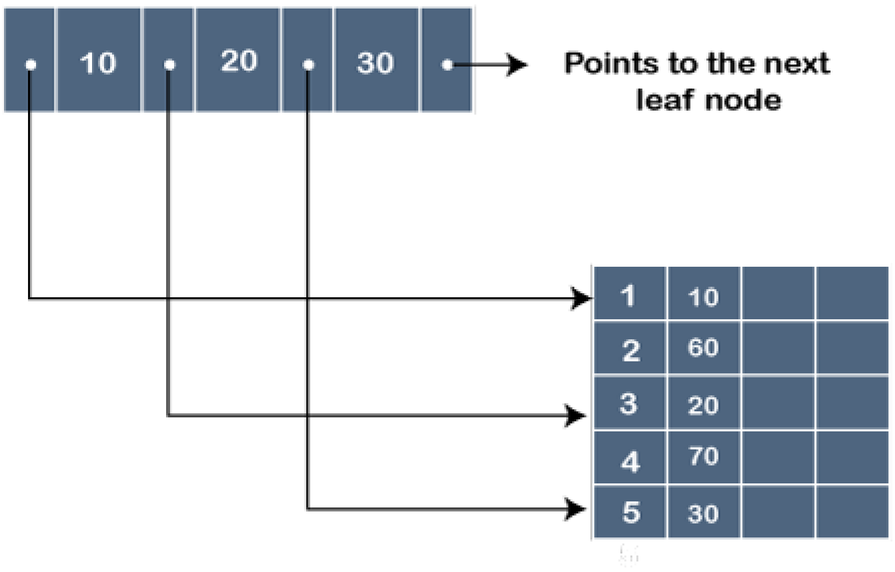
The minimum number of children is: 2

Special cases in B+ tree

Case 1: If the root node is the only node in the tree. In this case, the root node becomes the leaf node.

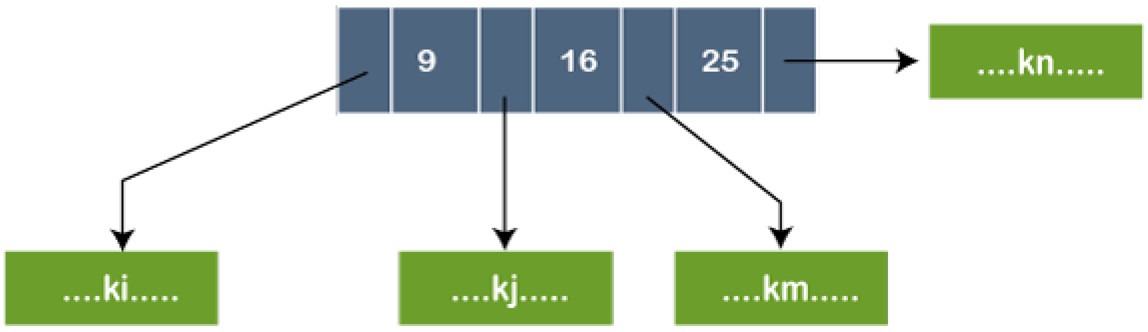
In this case, the maximum number of children is 1, i.e., the root node itself, whereas, the minimum number of children is b-1, which is the same as that of a leaf node.

Representation of a leaf node in B+ tree



In the above figure, '.' represents the pointer, whereas the 10, 20 and 30 are the key values. The pointer contains the address at which the key value is stored, as shown in the above figure.

Example of B+ tree



In the above figure, the node contains three key values, i.e., 9, 16, and 25. The pointer that appears before 9, contains the key values less than 9 represented by k_i. The pointer that appears before 16, contains the key values greater than or equal to 9 but less than 16 represented by kj. The pointer that appears before 25, contains the key values greater than or equal to 16 but less than 25 represented by k_n.

Differences between B tree and B+ tree

The following are the differences between the B tree and B+ tree:

B tree	B+ tree
In the B tree, all the keys and records are stored in both internal as well as leaf nodes.	In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.
In B tree, keys cannot be repeatedly stored, which means that there is no duplication of keys or records.	In the B+ tree, there can be redundancy in the occurrence of the keys. In this case, the records are stored in the leaf nodes, whereas the keys are stored in the internal nodes, so redundant keys can be present in the internal nodes.
In the Btree, leaf nodes are not linked to each other.	In B+ tree, the leaf nodes are linked to each other to provide the sequential access.
In Btree, searching is not very efficient because the records are either stored in leaf or internal nodes.	In B+ tree, searching is very efficient or quicker because all the records are stored in the leaf nodes.
Deletion of internal nodes is very slow and a time-consuming process as we need to consider the child of the deleted key also.	Deletion in B+ tree is very fast because all the records are stored in the leaf nodes so we do not have to consider the child of the node.

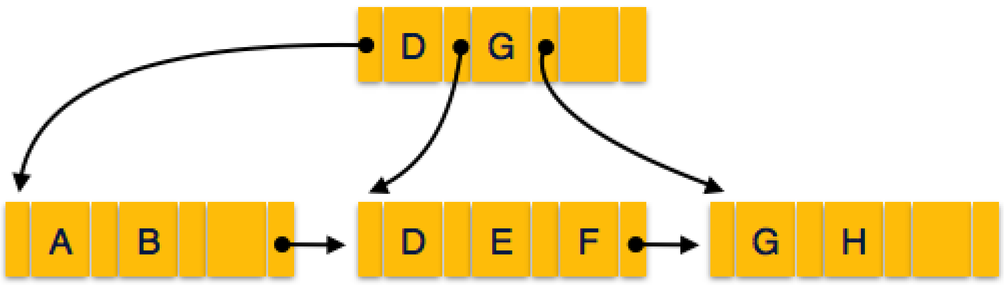
In Btree, sequential access is not possible.	In the B+ tree, all the leaf nodes are connected to each other through a pointer, so sequential access is possible.
In Btree, the more number of splitting operations are performed due to which height increases compared to width,	B+ tree has more width as compared to height.
In Btree, each node has atleast two branches and each node contains some records, so we do not need to traverse till the leaf nodes to get the data.	In B+ tree, internal nodes contain only pointers and leaf nodes contain records. All the leaf nodes are at the same level, so we need to traverse till the leaf nodes to get the data.
The root node contains atleast 2 to m children where m is the order of the tree.	The root node contains atleast 2 to m children where m is the order of the tree.

B^+ Tree

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

Structure of B^+ Tree

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



Internal nodes –

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

Leaf nodes –

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.

B^+ Tree Insertion

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

B^+ Tree Deletion

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

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 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.

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.

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

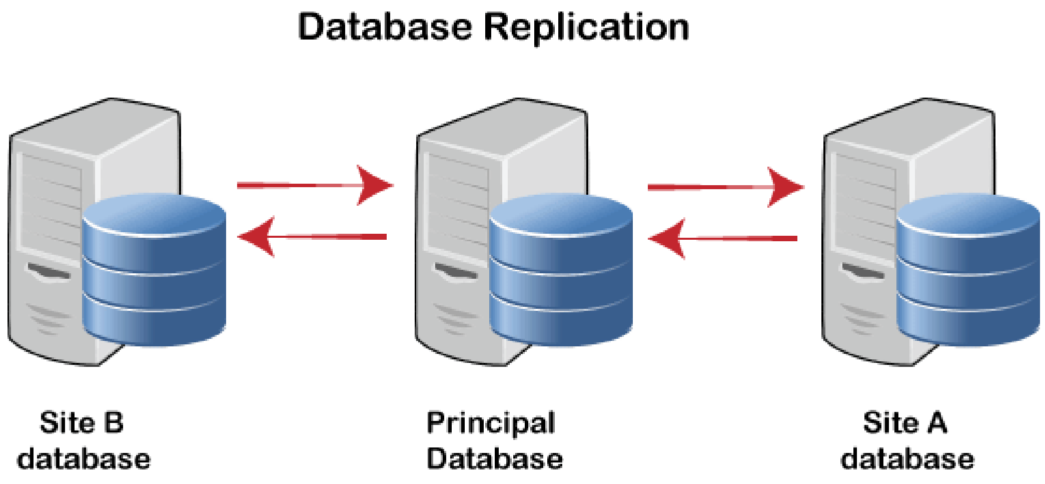
Using distributed databases has a lot of benefits.

- 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.