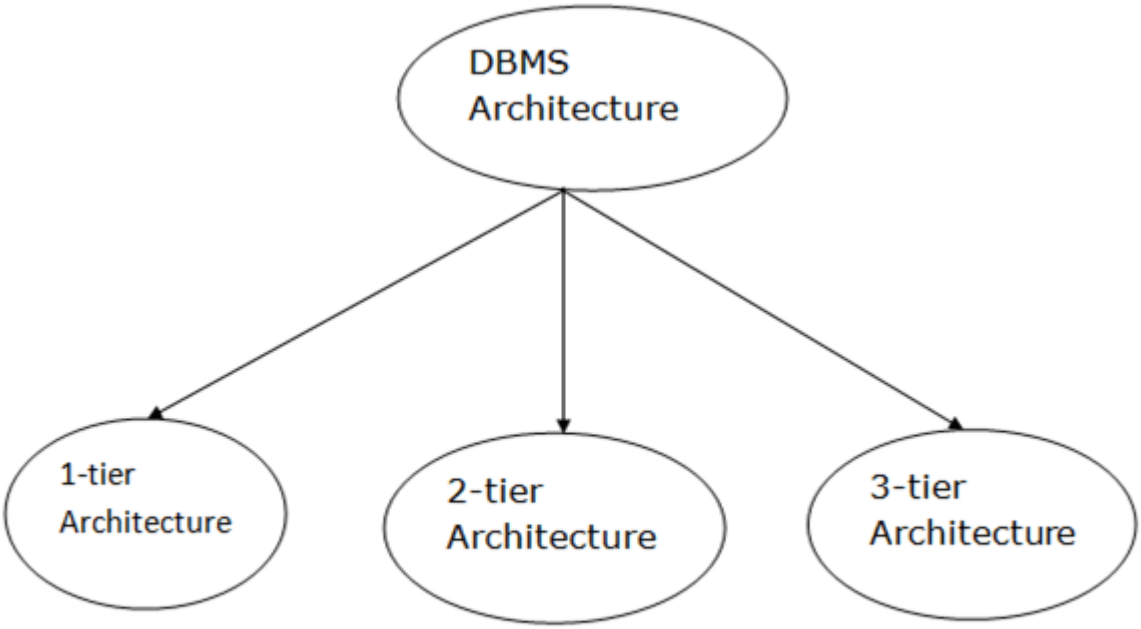
ADVANCED DATABASE MANAGEMENT SYSTEM

UNIT I
Database System Architecture

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.

- To communicate with the DBMS, client-side application establishes a connection with the server side.

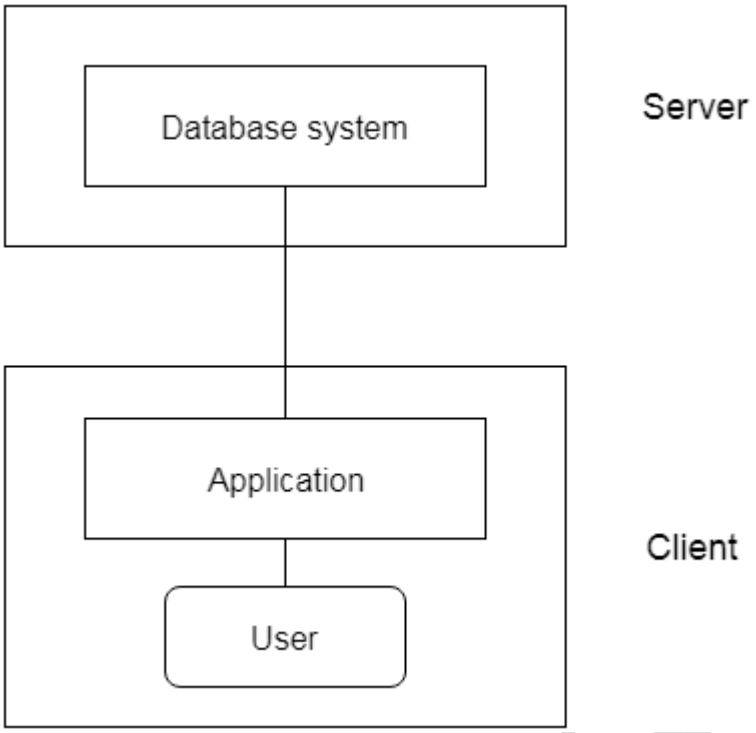


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

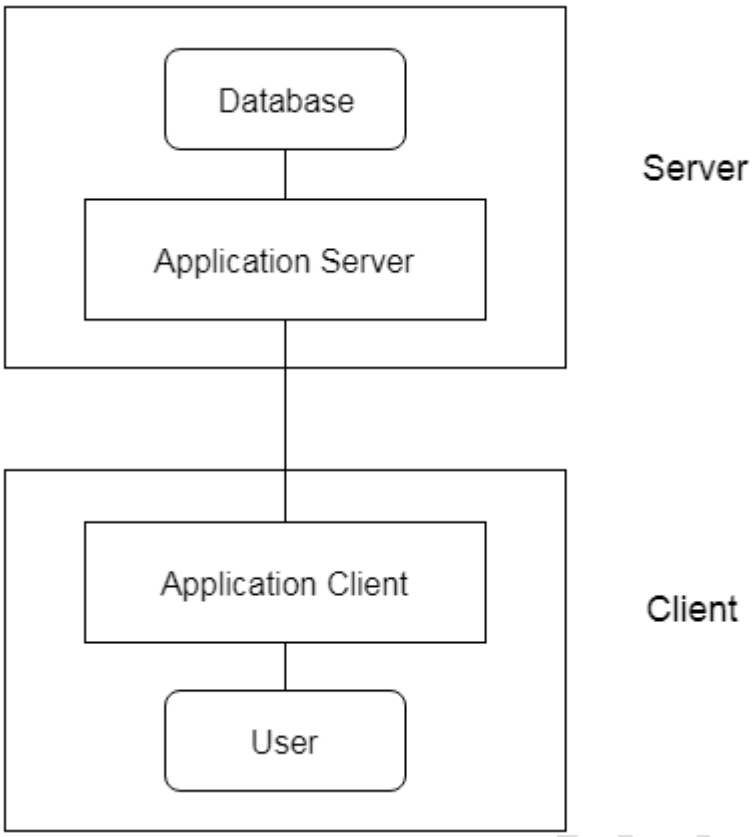


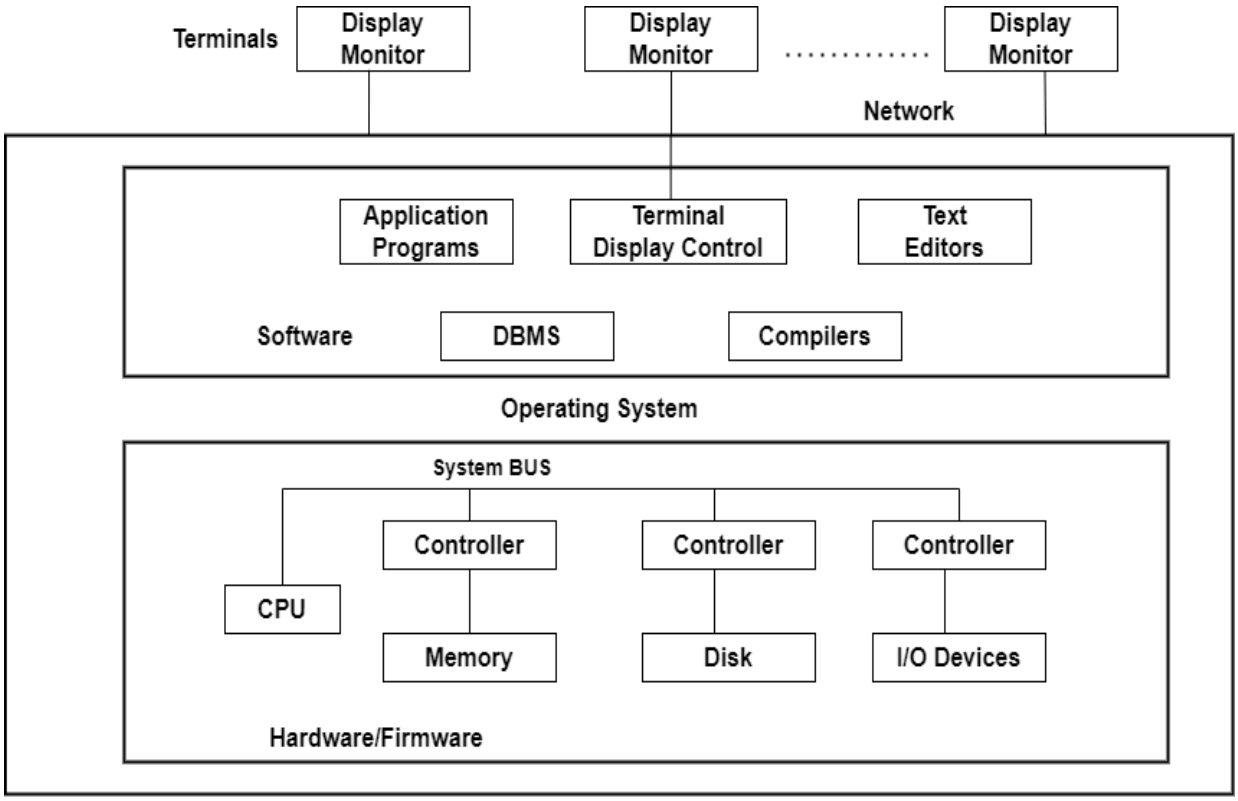
Fig: 3-tier Architecture

Centralized and Client –Server Architectures

Centralized Architecture of DBMS:

Architectures for DBMSs have generally followed trends seen in architectures for larger computer systems. The primary processing for all system functions, including user application programs, user interface programs, and all DBMS capabilities, was handled by mainframe computers in earlier systems. The primary cause of this was that the majority of users accessed such systems using computer terminals with limited processing power and merely display capabilities. Only display data and controls were delivered from the computer system to the display terminals, which were connected to the central node by a variety of communications networks, while all processing was done remotely on the computer system.

The majority of users switched from terminals to PCs and workstations as hardware prices decreased. Initially, Database Systems operated on these computers in a manner akin to how they had operated display terminals. As a result, the DBMS itself continued to operate as a centralized DBMS, where all DBMS functionality, application program execution, and UI processing were done on a single computer. The physical elements of a centralized architecture Client/server DBMS designs emerged as DBMS systems gradually began to take advantage of the user side's computing capability.



Centralized architecture of DBMS

Client-server Architecture of DBMS:

We first talk about client/server architecture in general, and then we look at how DBMSs use it. In order to handle computing settings with a high number of PCs, workstations, file servers, printers, database servers, etc., the client/server architecture was designed.

A network connects various pieces of software and hardware, including email and web server software. To define specialized servers with a particular functionality is the aim. For instance, it is feasible to link a number of PCs or compact workstations to a file server that manages the client machines' files as clients. By having connections to numerous printers, different devices can be designated as a printer server; all print requests from clients are then directed to this machine. The category of specialized servers also includes web servers and email servers. Many client machines can utilize the resources offered by specialized servers. The user is given the proper user interfaces for these servers as well as local processing power to run local applications on the client devices. This idea can be applied to various types of software, where specialist applications, like a CAD (computer-aided design) package, are kept on particular server computers and made available to a variety of clients. Some devices (such as workstations or PCs with discs that only have client software installed) would only be client sites.

The idea of client/server architecture presupposes an underpinning structure made up of several PCs and workstations as well as fewer mainframe computers connected via LANs as well as other types of computer networks. In this system, a client is often a user

machine that offers local processing and user interface capabilities. When a client needs access to extra features-like database access-that are not available on that system, it connects to a server that offers those features. A server is a computer system that includes both hardware and software that can offer client computer services like file access, printing, archiving, or database access. Generally speaking, some workstations install both client and server software, while others just install client software. Client and server software, however, typically run on separate workstations, which is more typical. On this underlying client/server framework, **Two-tier** and **Three-tier** fundamental DBMS architectures were developed.

Two-Tier Client Server Architecture:

Here, the term "two-tier" refers to our architecture's two layers-the Client layer and the Data layer. There are a number of client computers in the client layer that can contact the database server. The API on the client computer will use **JDBC** or some other method to link the computer to the database server. This is due to the possibility of various physical locations for clients and database servers.

Three-Tier Client-Server Architecture:

The Business Logic Layer is an additional layer that serves as a link between the Client layer and the Data layer in this instance. The layer where the application programs are processed is the business logic layer, unlike a Two-tier architecture, where queries are performed in the database server. Here, the application programs are processed in the application server itself.

Sharding

Sharding is a method of distributing a large database across multiple servers. This approach is commonly used in client-server architectures to improve performance and scalability. The data is split into smaller chunks called shards, which are then distributed across multiple servers.

Each shard is a self-contained subset of the data, and clients can connect to any server to access the data they need. This approach allows for horizontal scaling, which means that as the amount of data or the number of clients increases, more servers can be added to the system to handle the load.

Replication

Replication is a method of maintaining multiple copies of a database on different servers. This approach is commonly used in client-server architectures to improve fault-tolerance and performance. There are several types of replication, including master-slave replication, where one server acts as the master and other servers act as slaves, and all changes made on the master are replicated to the slaves.

Another type of replication is called master-master replication, where multiple servers can act as both a master and a slave, allowing data to be written to any server, and changes are replicated to all other servers.

Caching

Caching is a method of storing frequently accessed data in memory for faster access. This approach is commonly used in both centralized and client-server architectures to improve performance. When a client requests data from the server, the server first checks if the data is already in the cache.

If it is, the server returns the data from the cache, which is faster than retrieving it from the main data store. Caching can also be used to temporarily store data that is about to be written to the main data store, which can help to reduce the load on the server and improve write performance.

Load balancing

Load balancing is a method of distributing the load across multiple servers. This approach is commonly used in client-server architectures to improve performance and scalability. Load balancers are typically placed in front of a group of servers and are responsible for distributing incoming requests to the different servers.

This can be done in a number of ways, such as round-robin or least connections, and the goal is to ensure that all servers are used as efficiently as possible. Load balancing also helps to improve fault-tolerance, as if one server goes down, the load balancer can redirect traffic to other servers, keeping the system running smoothly.

These are just a few examples of how different techniques and methods can be used to improve the performance, scalability and availability of database systems. It's important to keep in mind that the architecture of a database system is crucial in ensuring its ability to meet the performance and scalability requirements of the system. Identifying the right architecture and implementing it with the best practices will be crucial to the success of a DBMS.

Conclusion

Both centralized and client-server architectures for DBMS have their own advantages and disadvantages, and the choice of architecture will depend on the specific needs of the application. Centralized architectures are simpler and easier to manage, but they can become a bottleneck as the system grows in size. Client-server architectures are more complex, but they are more scalable and fault-tolerant, making them a better choice for larger and more critical systems.

When it comes to code examples, specific DBMS also has their own syntax, structure, which is not exactly the same, but it gives you a rough idea on how to connect and create table in DBMS. It's important to consult the documentation of the specific DBMS you are using and test your code before deploying it to a production environment.

Parallel Systems

A parallel database is one which involves multiple processors and working in parallel on the database used to provide the services.

A parallel database system seeks to improve performance through parallelization of various operations like loading data, building index and evaluating queries parallel systems improve processing and I/O speeds by using multiple CPU’s and disks in parallel.

Working of parallel database

Let us discuss how parallel database works in step by step manner −

Step 1 − Parallel processing divides a large task into many smaller tasks and executes the smaller tasks concurrently on several CPU's and completes it more quickly.

Step 2 − The driving force behind parallel database systems is the demand of applications that have to query extremely large databases of the order of terabytes or that have to process a large number of transactions per second.

Step 3 − In parallel processing, many operations are performed simultaneously as opposed to serial processing, in which the computational steps are performed sequentially.

This working of parallel database is explained in the diagram given below −

Performance measures

There are two main resources of performance of a database system, which are explained below −

- **Throughput** − The number of tasks that can be completed in a given time interval. A system that processes a large number of small transactions can improve throughput by processing many transactions in parallel.
- **Response time** − The amount of time it takes to complete a single task from the time it is submitted. A system that processes large transactions can improve response time, as well as throughput by performing subtasks of each transaction in parallel.

Benefits of parallel Database

The benefits of the parallel database are explained below −

Speed

Speed is the main advantage of parallel databases. The server breaks up a request for a user database into parts and sends each part to a separate computer.

We eventually function on the pieces and combine the outputs, returning them to the customer. It speeds up most requests for data so that large databases can be reached more easily.

Capacity

As more users request access to the database, the network administrators are adding more machines to the parallel server, increasing their overall capacity.

For example, a parallel database enables a large online store to have at the same time access to information from thousands of users. With a single server, this level of performance is not feasible.

Reliability

Despite the failure of any computer in the cluster, a properly configured parallel database will continue to work. The database server senses that there is no response from a single computer and redirects its function to the other computers.

Many companies, such as online retailers, want their database to be accessible as fast as possible. This is where a parallel database stands good.

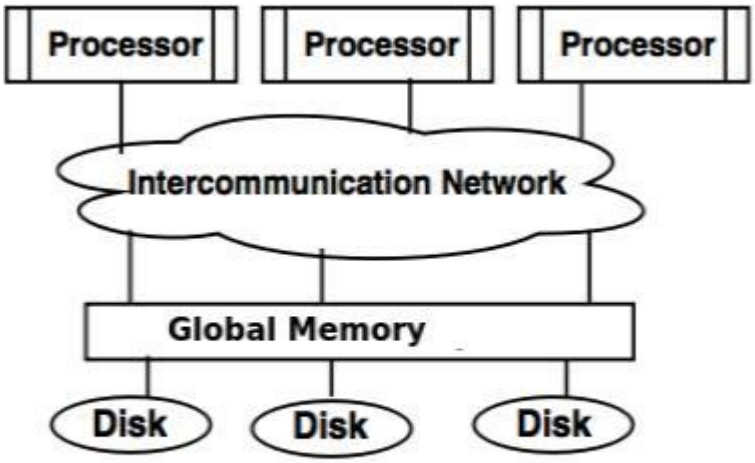
This method also helps in conducting scheduled maintenance on a computer-by-computer technician. They send a server command to uninstall the affected device, then perform the maintenance and update required.

Benefits for queries

Parallel query processing can benefit the following types of queries −

- Select statements that scan large numbers of pages but output a few rows only.
- Select statements that include union, order by, or distinct, since these queries can populate worktables in parallel, and can make use of parallel sorting.
- Select statements that use merge joins can use parallel processing for scanning tables and also for sorting and merging.
- Select statements where the reformatting strategy is chosen by the optimizer, since these can populate worktables in parallel, and can make use of parallel sorting.

- Create index statements, and the alter table - add constraint clauses that create indexes, unique and primary keys.



Shared Memory System in Parallel Databases

Distributed Systems

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

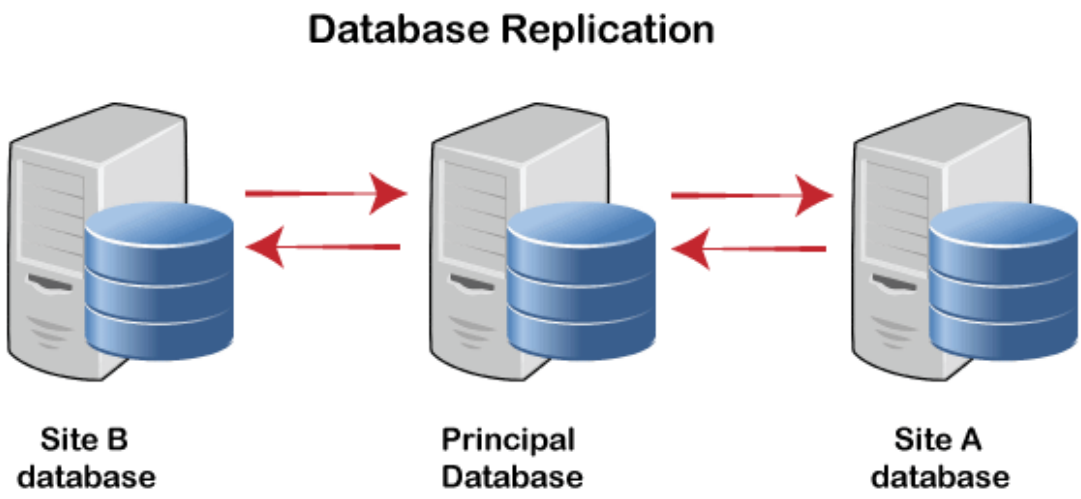
Using distributed databases has a lot of benefits.

- 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.
- 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.