Chapter -4.1
Distributed Database

- Distributed Database Concepts and Advantages
- Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design;
- Types of Distributed Database Systems;
- Distributed Database Architectures

Introduction to Distributed Computing and Distributed Database(DDB)

- A distributed computing system consists of a number of processing sites or nodes that are interconnected by a computer network and that cooperate in performing certain assigned tasks. Distributed databases bring the advantages of distributed computing to the database domain.
- As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it efficiently in a coordinated manner. Thus, more computing power is available to solve a complex task, and the autonomous processing nodes can be managed independently while they cooperate to provide the needed functionalities to solve the problem.

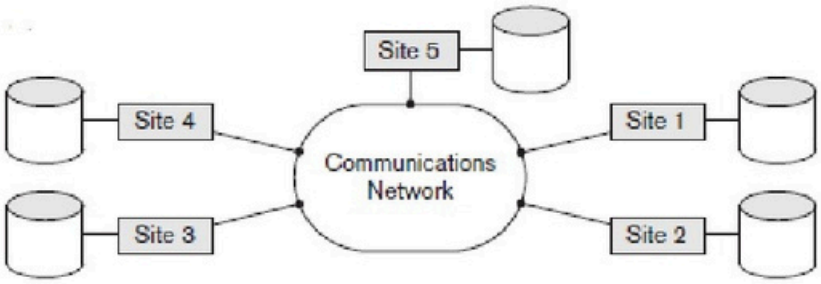

Fig: Distributed Database

- DDB technology resulted from a merger of two technologies: database technology and distributed systems technology.
- A distributed database is a collection of databases which are distributed over different computers of a computer network. Each site has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application which requires accessing data at several sites. For a database to be called distributed, the following minimum conditions should be satisfied:
    1. Connection of database nodes over a computer network. There are multiple computers, called sites or nodes. These sites must be connected by an underlying network to transmit data and commands among sites.
    2. Logical interrelation of the connected databases. It is essential that the information in the various database nodes be logically related.
    3. Possible absence of homogeneity among connected nodes. It is not necessary that all nodes be identical in terms of data, hardware, and software.
- A distributed database management system (DDBMS) as a software system that manages a distributed database while making the distribution transparent to the user.

Advantage of DDB

1. **Transparency:** provides several form of transparency like location transparency, fragmentation transparency, replication transparency. The concept of transparency extends the general idea of hiding implementation details from end users.
2. **Local Autonomy**: permits setting and enforcing local policies regarding the use of local data (suitable for organization that are inherently decentralized). Autonomy determines the extent to which individual nodes or DBs in a connected DDB can operate independently.
3. **Improved Performance:** The regularly used data is proximate to the users and given the parallelism inherent in distributed systems.

4. **Improved Reliability/Availability:** Data replication can be used to obtain higher reliability and availability. Reliability is broadly defined as the probability that a system is running (not down) at a certain time point, whereas availability is the probability that the system is continuously available during a time interval
5. **Incremental Growth:** supports a smooth incremental growth with a minimum degree of impact on the already existing sites.
6. **Shareability:** allows preexisting sites to share data.
7. **Economics:** It cost less to create a network of smaller computers with a power of single large computer.

Disadvantage of DDB

1. **Complexity:** Complex software is required for distributed database environment. As we know that the possible duplication is mainly due to reliability and efficiency considerations. Data redundancy, however, complicates update operations. If some sites fail while an update is being executed, the system must make sure that the effects will be reflected on the data residing at the failing sites as soon as the system can recover from the failure.
2. **Economics:** Increased complexity and requirement of extensive infrastructures requires extra cost.
3. **Security:** Remote database fragments must be secured, and they are not centralized so remote site must be secures as well. The infrastructure must also be secured.
4. **Difficult to maintain integrity:** In a distributed database, enforcing integrity over a network may require too much of the network resources to be feasible.
5. **Inexperience:** Distributed database are difficult to work with, and as a young field there is much not readily available experience on proper practice.
6. **Lack of standard:** There are no tools yet to help users convert a centralized DBMS into a distributed DBMS.
7. **Database design more complex:** Beside the normal difficulties, the design of the distributed database has to consider fragmentation of data, allocation of fragments to specific site and data replication.

Data Fragmentation, Replication and Allocation Techniques for Distributed Database Design

- Data Fragmentation techniques are used to break up the database into logical units, called fragments, which may be assigned for storage at the various nodes.
- Data replication techniques permits certain data to be stored in more than one site to increase availability and reliability.
- There are two approaches for allocating relation in distributed database. These Allocation techniques deals with the process of allocating **fragments or replicas of fragment,** for storage at the various nodes.
- Combinedly, these techniques are used during the process of distributed database design.
- The information concerning data fragmentation, allocation, and replication is stored in a global directory that is accessed by the DDBS applications as needed.

Data Fragmentation

- In a Distributed Database (DDB), fragmentation refers to the division of a database into smaller parts or fragments that are distributed across multiple nodes or locations in a network.
- Data fragmentation describes how the global relations are divided into fragments. Relation is partitioned into several fragments stored in distinct sites.
- Division of relation r into fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation r.
- Three approaches
  1. Horizontal Fragmentation
  2. Vertical Fragmentation
  3. Hybrid Fragmentation

Horizontal fragmentation

- A horizontal fragment or shard of a relation is a subset of the tuples in that relation. The tuples that belong to the horizontal fragment can be specified by a condition on one or more attributes of the relation, or by some other mechanism.
- Each tuple of relation r is assigned to one or more fragments.

- In horizontal fragmentation, a relation r is partitioned into a number of subsets, r1, r2, . . . , rn. Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.
- In general, a horizontal fragment can be defined as a selection on the global relation r. That is, we use a predicate Pi to construct fragment ri :

$$r_i = \sigma\ Pi\ (r\ )$$

- The below example shows the horizontal fragmentation of account relation

branch_name	account_number	balance
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$account_1 = \sigma_{branch\_name="Hillside"}(account\ )$

branch_name	account_number	balance
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_2 = \sigma_{branch\_name="Valleyview"}(account\ )$

- We reconstruct the relation r by taking the union of all fragments; that is:

$$r = r1 \cup r2 \cup \cdots \cup rn$$

Vertical fragmentation

- Each site may not need all the attributes of a relation, which would indicate the need for a different type of fragmentation. Vertical fragmentation divides a relation "vertically" by columns. A vertical fragment of a relation keeps only certain attributes of the relation.
- The schema for relation *r* is split into several smaller schemas.
- Vertical fragmentation of r(R) involves the definition of several subsets of attributes R1, R2, . . . , Rn of the schema R so that: $R = R1 \cup R2 \cup \cdots \cup Rn$
- Each fragment r_i of r is defined by:

$$ri = \Pi_{Ri}\ (r\ )$$

- The below example shows the vertical fragmentation of employ_info relation

branch_name	customer_name	tuple_id
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}(employee\_info\ )$

account_number	balance	tuple_id
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(employee\_info\ )$

- The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join:

$$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \cdots \bowtie r_n$$

Mixed (Hybrid) Fragmentation

- Hybrid fragmentation is the combination of both vertical and horizontal fragmentation. Hybrid fragmentation can be done using one of the following two techniques.
  i. First generate horizontal fragments, then generate vertical fragments from one or more horizontal fragments.
  ii. First generate vertical fragments, then generate horizontal fragments from one or more horizontal fragments.

Another Example:

J

JNO	JNAME	BUDGET	LOC
J1	Instrumental	150,000	Montreal
J2	Database Dev.	135,000	New York
J3	CAD/CAM	250,000	New York
J4	Maintenance	350,000	Paris

Horizontal Partitioning

J1

JNO	JNAME	BUDGET	LOC
J1	Instrumental	150,000	Montreal
J2	Database Dev.	135,000	New York

J2

JNO	JNAME	BUDGET	LOC
J3	CAD/CAM	150,000	Montreal
J4	Maintenance.	310,000	Paris

Vertical Partitioning

JNO	BUDGET
J1	150,000
J2	135,000
J3	250,000
J4	310,000

JNO	JNAME	LOC
J1	Instrumentation	Montreal
J2	Database Devl	New York
J3	CAD/CAM	New York
J4	Maintenance	Paris

Advantage of fragmentation

- **Improved Performance:** By distributing the database across multiple nodes, fragmentation can enhance performance in terms of query execution time and response time. Queries can be processed concurrently on different fragments, enabling parallel processing and reducing the overall execution time.
- **Increased Scalability:** Fragmentation allows for horizontal scalability by adding more nodes to the network. As the database grows, new fragments can be created and distributed among additional nodes, ensuring that the system can handle larger amounts of data and increasing the capacity to serve more users.
- **Enhanced Availability:** Distributed fragments provide redundancy and fault tolerance. If one node or fragment becomes unavailable due to a failure or maintenance, other nodes can still process queries using the available fragments. This improves the overall availability and reliability of the system.
- **Localized Data Access:** Fragmentation can be used to store data closer to the users or applications that frequently access it. By placing fragments on nodes geographically closer to the users, the latency and network traffic can be reduced, resulting in faster data access.
- **Simplified Management:** Fragmentation can simplify database management tasks. Each fragment can be managed independently, allowing for easier backup and recovery operations, load balancing, and data partitioning strategies. It can also enable efficient data archiving or purging by selectively targeting specific fragments.

Disadvantage of fragmentation

- **Increased Complexity:** Fragmentation introduces additional complexity in the design, implementation, and management of the database. It requires careful planning and coordination to determine the optimal fragmentation scheme, ensuring that data is appropriately distributed and organized across the network.

- **Data Consistency and Integrity:** As data is distributed across multiple fragments, maintaining the integrity and consistency of the database becomes complex. Synchronization mechanisms and protocols need to be implemented to handle concurrent updates, data replication, and distributed transactions.
- **Increased Network Traffic:** Fragmentation can result in increased network traffic. As queries are executed on distributed fragments, data from multiple fragments may need to be accessed and exchanged to satisfy a query. This can lead to higher network utilization and increased latency, particularly if the fragments are located in different geographical regions or if the network bandwidth is limited.
- **Fragmentation Overhead:** Managing and maintaining fragmented data incurs additional overhead. Moving data between fragments or reorganizing fragments can be time-consuming and resource-intensive, impacting overall system performance.
- **Fragmentation Design and Optimization:** Selecting an appropriate fragmentation scheme is crucial. An inefficient fragmentation design can result in unbalanced data distribution, leading to performance degradation
- **Limited Joins and Complex Queries:** Fragmentation can complicate queries that involve joins or involve data from multiple fragments. Coordinating and executing distributed joins across fragments requires additional processing and coordination overhead

Replication

- Replication is the process of maintaining multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
- Replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, thus creating a **fully replicated distributed database**. This can improve **availability** remarkably because the system can continue to operate as long as at least one site is up.
- It also **improves performance of** retrieval (read performance) for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at **the local site where it is submitted.**
- The **disadvantage of full replication** is that it can slow down update operations (write performance) drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent.
- Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication.
- **The other extreme from full replication involves having no replication**—that is, each fragment is stored at exactly one site. In this case, all fragments must be disjoint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called **nonredundant allocation.**
- Between these two extremes, we have a wide spectrum of **partial replication of the** data—that is, some fragments of the database may be replicated whereas others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system.
- A **special case of partial replication is occurring heavily in applications where mobile workers**—such as sales forces, financial planners, and claims adjustors—carry partially replicated databases with them on laptops and PDAs and **synchronize them periodically with the server database**.

Advantage of replication

- Availability: failure of site containing relation r does not result in unavailability of r is replicas exist.
- Parallelism: queries on r may be processed by several nodes in parallel.
- Reduced data transfer: relation r is available locally at each site containing a replica of r.

Disadvantage of replication

- Increased cost of updates: each replica of relation r must be updated.
- Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.

Allocation

- There are basically two approaches for allocating data fragments in distributed database. These Allocation techniques deals with the process of allocating **fragments or replicas of fragment,** for storage at the various nodes.

- Each fragment—or each copy of a fragment—must be assigned to a particular site in the distributed system. This process is **called data distribution (or data allocation).**
- The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.
- For example, **if high availability** is required, transactions can be submitted at any site, and most transactions are retrieval only, a **fully replicated database is a good choice.**
- However, if certain transactions that access **particular parts** of the database are mostly submitted at a particular site, the **corresponding set of fragments can be allocated at that site only.** Data that is accessed at multiple sites can be replicated at those sites.
- If **many updates are performed**, it may be useful to **limit replication.**
- Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

Types of Distributed Database Systems

- Broadly DDS is classified into two types as below
  1. Homogeneous DDS
  2. Heterogeneous DDS

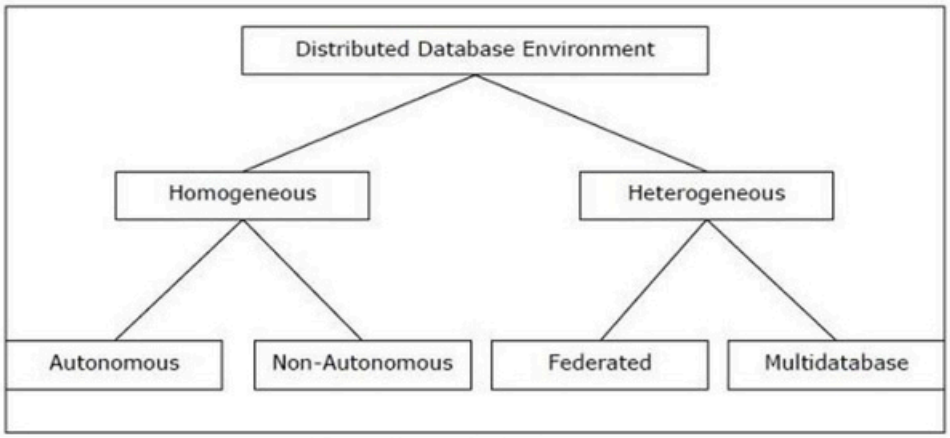

Fig: Types of DDS

Homogeneous DDS

- If data is distributed but all servers run the same DBMS software, we have a homogeneous distributed database system. i.e. all sites of the database system have identical setup, i.e., same database system software.
- The underlying operating system may be same or different. The underlying operating systems can be a mixture of Linux, Window, Unix, etc.

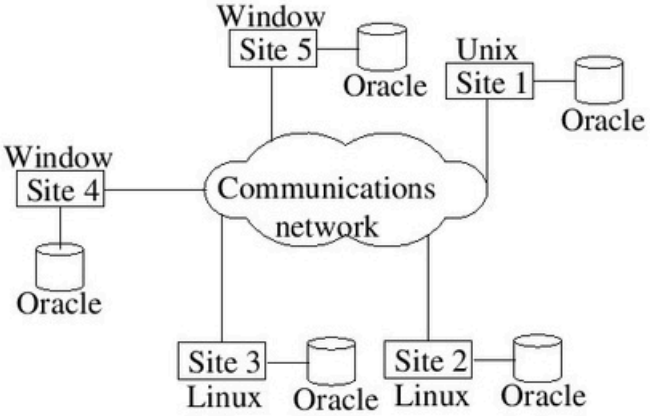

Fig: Homogeneous DDS

- Its properties are −
  1. The sites use very similar software.
  2. The sites use identical DBMS or DBMS from the same vendor.
  3. Each site is aware of all other sites and cooperates with other sites to process user requests.

4. The database is accessed through a single interface as if it is a single database.

- There are two types of Homogeneous DDS: Autonomous and Non-Autonomous.
- If direct access by local transactions to a server is permitted, the system has some degree of local autonomy. local autonomy refers to the ability of individual databases or nodes to manage and control their own data, operations, and resources without relying on a centralized authority. Each database is independent that functions on its own. They are integrated by a controlling application and uses message passing to share data and updates.
- If there is no provision for the local site to function as a standalone DBMS, then the system has **no local autonomy.** Here data is distributed across the homogeneous nodes and **master DBMS coordinates** data sharing and updates.

- If different sites run under the control of different DBMSs, essentially autonomously, and are connected somehow to enable access to data from multiple sites, we have a heterogeneous distributed database system.

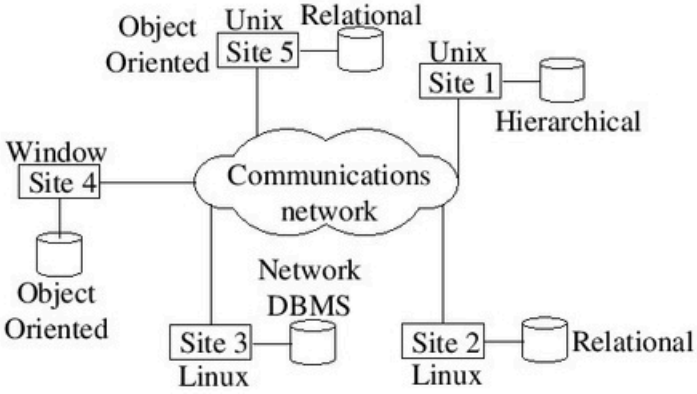

Fig: Heterogeneous DDS

- In a heterogeneous distributed database, **different sites have different operating systems, DBMS products and data models**
- Its properties are –
  - Different sites use dissimilar schemas and software.
  - The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
  - Query processing is complex due to dissimilar schemas.
  - Transaction processing is complex due to dissimilar software.
  - A site may not be aware of other sites and so there is limited co-operation in processing user requests.
- Further Heterogeneous DDS is classified into two categories i.e., Federated and Multi-database
- In Federated DDS, the heterogeneous database systems are independent in nature and **integrated together** so that they function as a single database system. A federated database system consists of **multiple autonomous databases** that are geographically distributed but interconnected. Each database retains its own autonomy, but there is **a middleware layer** that provides a unified interface for accessing and querying data across the distributed databases. Federation enables data integration and allows applications to access data from multiple sources seamlessly. The term federated database system (FDBS) is used when there is some **global view or schema** of the federation of databases that is shared by the applications. A **middleware layer, known as a federated database management system (FDBMS),** provides a unified interface and coordinates the access and integration of data across the distributed databases.
- Note, the global schema defines the organization of data, including the tables, attributes, relationships, and constraints that are shared or accessible across multiple databases. It provides a high-level description of the entire distributed database, allowing users and applications to understand and interact with the system as a unified unit.
- On the other hand, a multi-database system or peer to peer database system **has full local autonomy** in that it does not have a global schema but interactively constructs one as needed by the application. There is no central authority or dedicated server. Each node in the network acts as both a client and a server, contributing resources and participating in the data storage and retrieval process.

Federated DBMS issues

- **Differences in data models**. Databases in an organization come from a variety of data models, including the so-called legacy models (hierarchical and network), the relational data model, the object data model, and even files. The modeling

capabilities of the models vary. Hence, to deal with them uniformly via a single global schema or to process them in a single language is challenging

- **Differences in constraints**: Constraint facilities for specification and implementation vary from system to system. There are comparable features that must be reconciled in the construction of a global schema. For example, the relationships from ER models are represented as referential integrity constraints in the relational model. Triggers may have to be used to implement certain constraints in the relational model. The global schema must also deal with potential conflicts among constraints.
- **Differences in query languages:** Even with the same data model, the languages and their versions vary. For example, SQL has multiple versions like SQL-89, SQL-92, SQL-99, and SQL:2008, and each system has its own set of data types, comparison operators, string manipulation features, and so on.

Distributed Database Architectures

Parallel vs Distributed Architecture

- Parallel and distributed architectures are two different approaches to designing and implementing systems for processing large amounts of data or executing computational tasks.
- Parallel Architecture: Parallel architecture involves the use of multiple processors or computing units that work together to perform a task or process data simultaneously. In a parallel architecture, the processors operate on different parts of the data or execute different subtasks concurrently. The key features of parallel architecture include:
    - **Shared Memory:** In parallel architectures, multiple processors typically have access to a shared memory space. This allows them to exchange data and communicate directly, simplifying data sharing and coordination.
    - **Task Partitioning:** The workload or data is divided into smaller units or tasks, which are assigned to different processors for parallel execution. Each processor works on its allocated task independently and synchronizes with other processors as needed.
    - **High Performance:** Parallel architecture aims to achieve high performance by leveraging the combined computational power of multiple processors. By dividing the workload and executing tasks in parallel, it can significantly reduce the overall processing time.
    - **Scalability:** Parallel architectures are inherently scalable as additional processors can be added to the system to handle larger workloads or increase computational capacity.
- Distributed Architecture: Distributed architecture, on the other hand, involves a network of interconnected nodes or computing resources that work together to achieve a common goal. In a distributed architecture, each node operates independently **and has its own memory and processing capabilities**. The key features of distributed architecture include:
    - **Message Passing:** Nodes in a distributed architecture communicate with each other by exchanging messages. Data and computation are typically distributed across the nodes, and communication happens explicitly through message passing protocols.
    - **Data Distribution:** Data is distributed across different nodes in a distributed architecture, allowing for parallel processing and fault tolerance. Each node manages its own portion of the data, and coordination is required to ensure consistency and synchronization.
    - **Scalability and Fault Tolerance:** Distributed architectures are highly scalable as new nodes can be added to the network to handle increased workloads or provide additional computational resources. They also provide fault tolerance, as the failure of one node does not necessarily lead to the failure of the entire system.
      There are two kinds of scalability in distributed systems: horizontal and vertical. Horizontal scalability is, where the distributed system is expanded by adding more nodes for data storage and processing as the volume of data grows. Vertical scalability, on the other hand, refers to expanding the storage and computing power of existing nodes

- Parallel architecture focuses on achieving high performance through parallel execution of tasks using shared memory, while distributed architecture emphasizes scalability, fault tolerance, and distribution of data and computation across independent nodes connected through a network.

General Architecture of Pure Distributed Database

- Figure below shows the generic schema architecture of a DDB.

- As shown in the figure, the enterprise is presented with a consistent, unified view showing the logical structure of underlying data across all nodes. This view is represented by the global conceptual schema (GCS), which provides network transparency. To accommodate potential heterogeneity in the DDB, each node is shown as having its own **local internal schema (LIS) based on physical organization** details at that particular site. The **logical organization of data at each site is specified by the local conceptual schema (LCS).** The GCS, LCS, and their underlying mappings provide the fragmentation and replication transparency. Figure

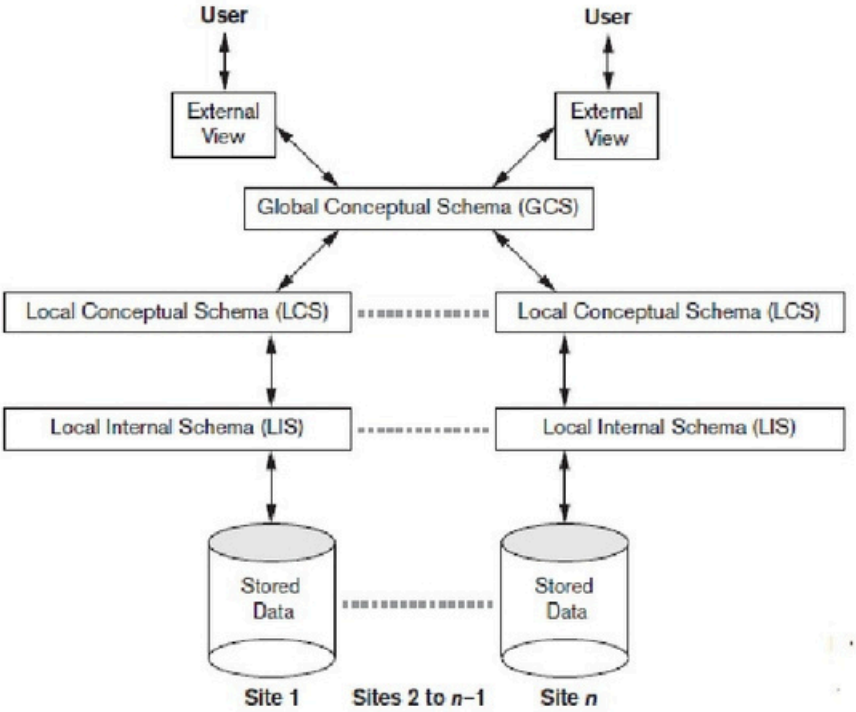

Fig: Schema Architecture of distributed database

- The global query compiler references the global conceptual schema from the global system catalog to verify and impose defined constraints. The global query optimizer references both global and local conceptual schemas and generates optimized local queries from global queries. It evaluates all candidate strategies using a cost function that estimates cost based on response time (CPU, I/O, and network latencies) and estimated sizes of intermediate results.
- Each local DBMS would have its local query optimizer, transaction manager, and execution engines as well as the local system catalog, which houses the local schemas. The global transaction manager is responsible for coordinating the execution across multiple sites in conjunction with the local transaction manager at those sites.

Federated Database Schema Architecture

- Typical five-level schema architecture that supports global applications in the FDBS environment is shown in Figure below.

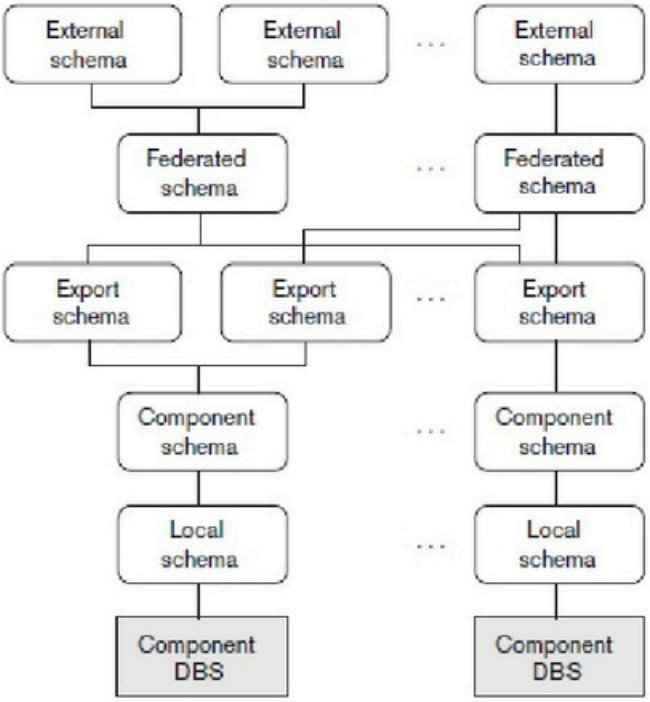

Fig: Five level schema architecture in federated database system(FDBS)

- In this architecture, the local schema is the conceptual schema (full database definition) of a component database, and the component schema is derived by translating the local schema into a canonical data model or common data model (CDM) for the FDBS. Schema translation from the local schema to the component schema is accompanied by generating mappings to **transform commands on a component schema into commands on the corresponding local schema.**
- The export schema represents the subset of a component schema that is available to the FDBS. The federated schema is the global schema or view, which is the result of integrating all the shareable export schemas.
- The external schemas define the schema for a user group or an application, as in the three-level schema architecture.

Three Tier Client/Server architecture

- It consists of clients running client software, a set of servers which provide all database functionalities and a reliable communication infrastructure.

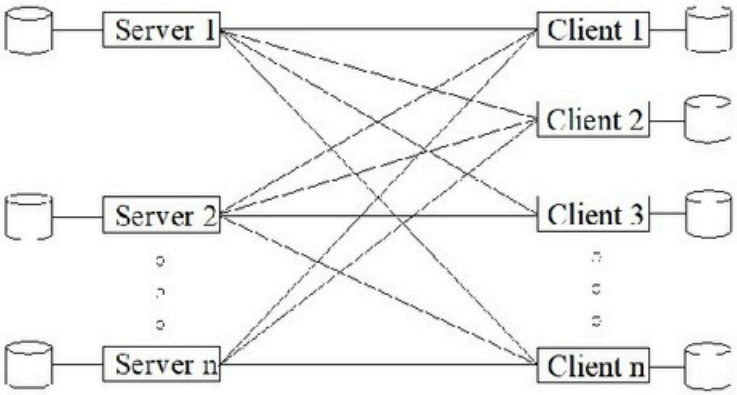

Fig : Client/Server Architecture

- A Client-Server system has one or more's client processes and one or more server processes, and a client process can send a query to anyone server process.
- Clients are responsible for user-interface issues, and servers manage data and execute transactions. Thus, a client process could run on a low-end processing node such as personal computer and send queries to a server running on high end processing node such as mainframe.
- The query processing follows following steps
    - Client parses a user query and decomposes it into a number of independent sub-queries. Each subquery is sent to appropriate site for execution.
    - Each server processes its query and sends the result to the client.

- The client combines the results of subqueries and produces the final result.

- The below figure shows the general three tier client/server architecture of DBS.

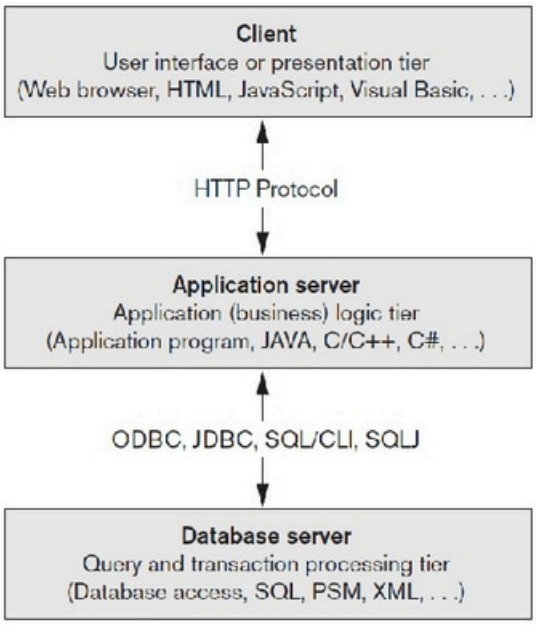

Fig: The Three Tier Client/Server Architecture

In three tier client/server architecture of DBS, the following layer exists.

Presentation layer (client)

- This provides the user interface and interacts with the user. The programs at this layer present Web interfaces or forms to the client in order to interface with the application. Web browsers are often utilized, and the languages and specifications used include HTML, XHTML, CSS, Flash, MathML, Scalable Vector Graphics (SVG), Java, JavaScript, Adobe Flex, and others.
- This layer handles user input, output, and navigation by accepting user commands and displaying the needed information, usually in the form of static or dynamic Web pages. The latter are employed when the interaction involves database access.
- When a Web interface is used, this layer typically communicates with the application layer via the HTTP protocol.

Application layer (business logic):

- This layer programs the application logic. For example, queries can be formulated based on user input from the client, or query results can be formatted and sent to the client for presentation.
- Additional application functionality can be handled at this layer, such as security checks, identity verification, and other functions.
- The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC, JDBC, SQL/CLI, or other database access techniques.

Database server

- This layer handles query and update requests from the application layer, processes the requests, and sends the results.
- Usually, SQL is used to access the database if it is relational or object-relational, and stored database procedures may also be invoked.
- Query results (and queries) may be formatted into XML when transmitted between the application server and the database server.