UNIT- I

Introduction; Distributed Data Processing, Distributed Database System, Promises of DDBSs, Problem areas.
Distributed DBMS Architecture: Architectural Models for Distributed DBMS, DDMBS Architecture. **Distributed Database Design**: Alternative Design Strategies, Distribution Design issues, Fragmentation, Allocation.

Introduction; Distributed Data Processing

Distributed data processing is a computer-networking method in which multiple computers across different locations share computer-processing capability. This is in contrast to a single, centralized server managing and providing processing capability to all connected systems.

The term distributed processing is probably the most abused term in computer science of the last couple of the year. It has been used to refer to such diverse system as multiprocessing systems, distributed data processing, and computer networks. This abuse has gone on such an extent that the term distributed processing has sometime been called a concept in search of a definition and a name .Here are some of the other term that have been synonymously with distributed processing distributed / multicomputers , satellite processing /satellite computers , backend processing , dedicated/special-purpose computers, time-shared systems and functionally modular system.

Distributed Database System:-

Distributed database systems can be defined as follows:
Distributed database system (DDBS) = Databases + Computers + Computer Network + Distributed database management system (DDBMS)
DDBS (distributed database system) = DDB + DDBMS

The database system manages data; the computer network makes the communication possible; and the DDBMS (distributed database management system) is the software that realises the mechanisms and policies for manipulating distributed data.

A distributed database system can be simply defined as a collection of multiple logically interrelated databases distributed over a computer network and managed by a distributed database management system.

A distributed database management system (DDBMS) is the software system that permits the management of the DDBS and makes the distribution transparent to users.

From the architectural point of view, a distributed database system consists of a collection of sites, connected together via a communication network, in which

➢ Each site is a database system site in its own right, but

➢ The sites have agreed to work together (if necessary), so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site.

Promises of DDBSs:

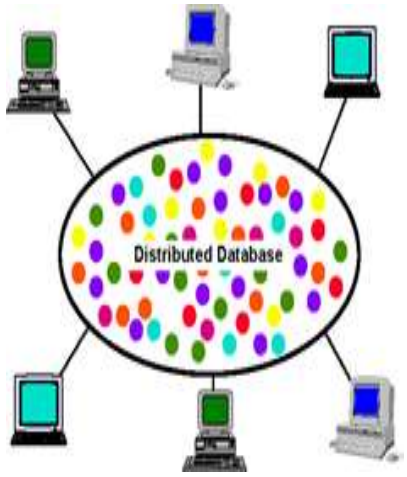
Distributed Database Systems provide the following advantages:

-Transparency of distributed and replicated data
-Higher reliability
-Improved performance
-Easier system expansion

Transparency:
Refers to the separation of the higher-level semantics of the system from the lower-level implementation issues

A transparent system "hides" implementation details from users.

A fully transparent DBMS provides high-level support for the development of complex applications.



The goal of transparency is to provide a system where users do not have to worry about representation, fragmentation, location, and replication of data.

Various forms of transparency can be distinguished for DDBSs:

Network transparency (also called distribution transparency)
-Location transparency
-Naming transparency

Replication transparency
Fragmentation transparency

Problem areas of distributed databases:

Distributed Concurrency Control: The integrity of the database is maintained by specifying the synchronization of access to the distributed database to manage concurrency different locking techniques uses based on the mutual exclusion of access to data.

Replication Control: Replication techniques only applicable to distributed systems where a database is supposed to be replicated if the whole database or a percentage of it is copied and the copies are stockpiled at dissimilar sites. Having more than one copy of a database, the issue is continuing the communal uniformity of the copies ensuring with all copies are identical schema and data.

Deadlock Handling: Numerous users request for the same resources from the database if the resources are obtainable at that time, then database allow permission for the resources to that user if not obtainable the user has to wait until the resources are released by another user. Sometimes the users do not release the resources blocked by some other users.

OS Environment: To implement distributed database environment a specific operating system is required as per organizational requirements. The operating system plays an important role to manage the distributed database because sometimes it doesn't support for distributed database.

Transparent Management: The major problem area in the distributed database where data located in numerous locations and number of users are used that database. So, the transparent management of data is important to maintain the integrity of distributed database.

Security and Privacy: A great issue in the distributed system that how to apply the security policies to the interdependent system. Since distributed systems contract with sensitive data and information ensuring with the strong security and privacy measurement system exists. The important issues in the distributed system are the protection of distributed system assets together with the fundamental resources, storage, communications and user-interface I/O, higher-level compounds of these resources, like processes, files, messages, display windows and more complex objects, etc.

Resource Management: Resources are in different places in the distributed system. Routing is an issue at the network layer of the distributed system and at the application layer. To cooperate with these resource in a distributed system.

Distributed DBMS Architecture:

Architectural Models for Distributed DBMS:

Some of the common architectural models are −

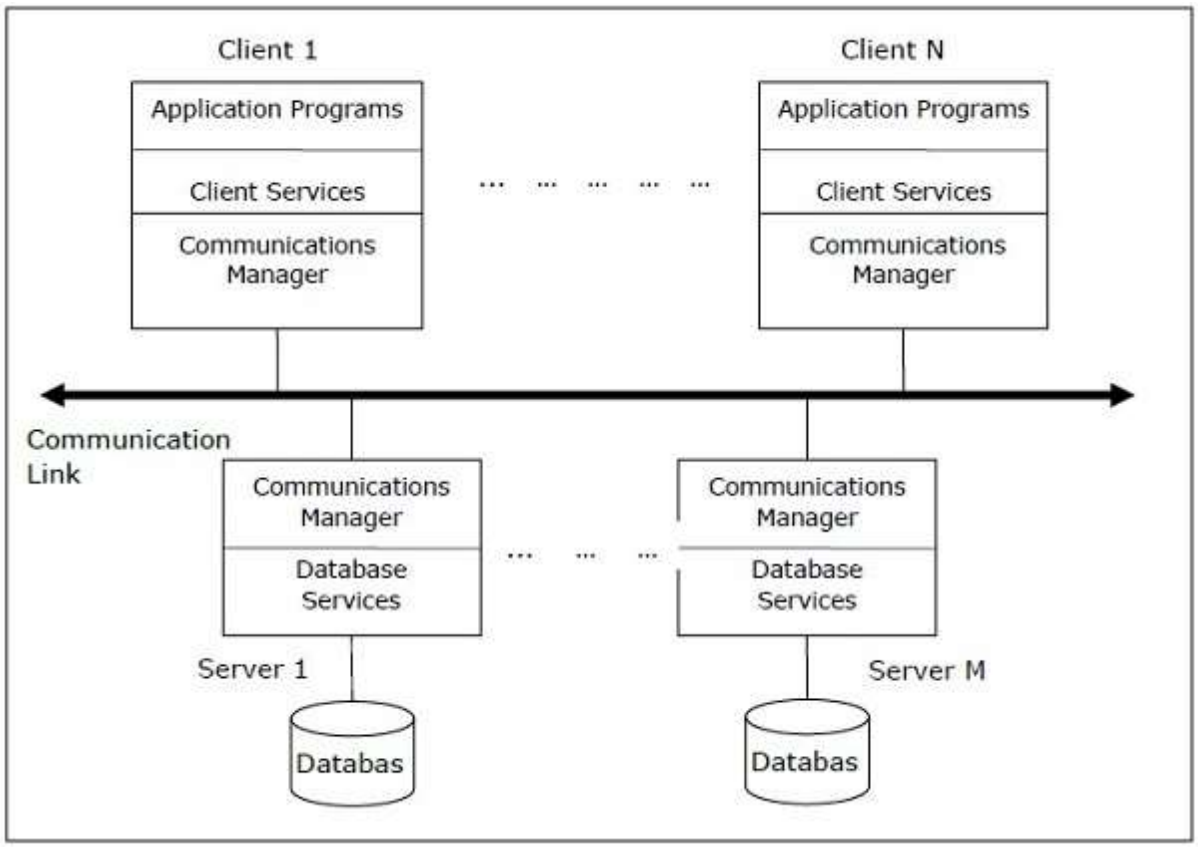
Client - Server Architecture for DDBMS
Peer - to - Peer Architecture for DDBMS
Multi - DBMS Architecture

Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The two different clients - server architecture are −

-Single Server Multiple Client
-Multiple Server Multiple Client (shown in the following diagram)

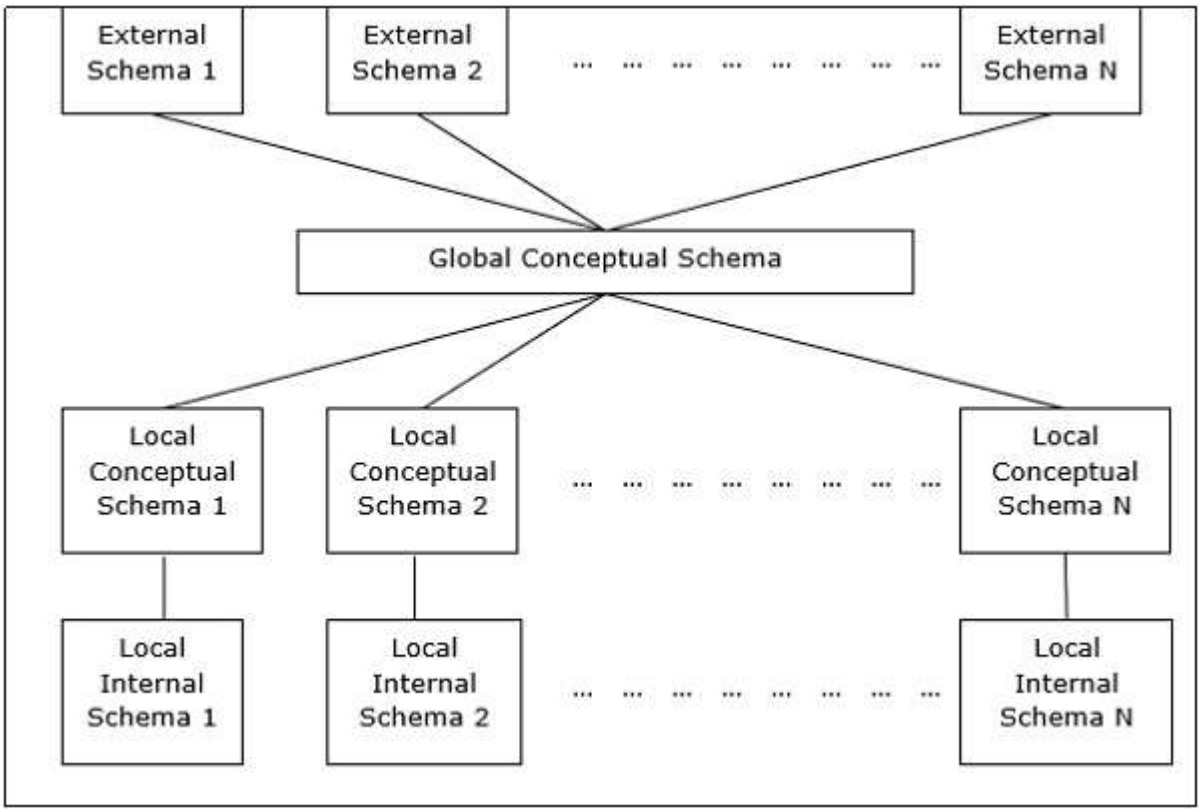


Peer- to-Peer Architecture for DDBMS:

In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas –

-Global Conceptual Schema − Depicts the global logical view of data.
-Local Conceptual Schema − Depicts logical data organization at each site.
-Local Internal Schema − Depicts physical data organization at each site.
-External Schema − Depicts user view of data.



Multi - DBMS Architectures

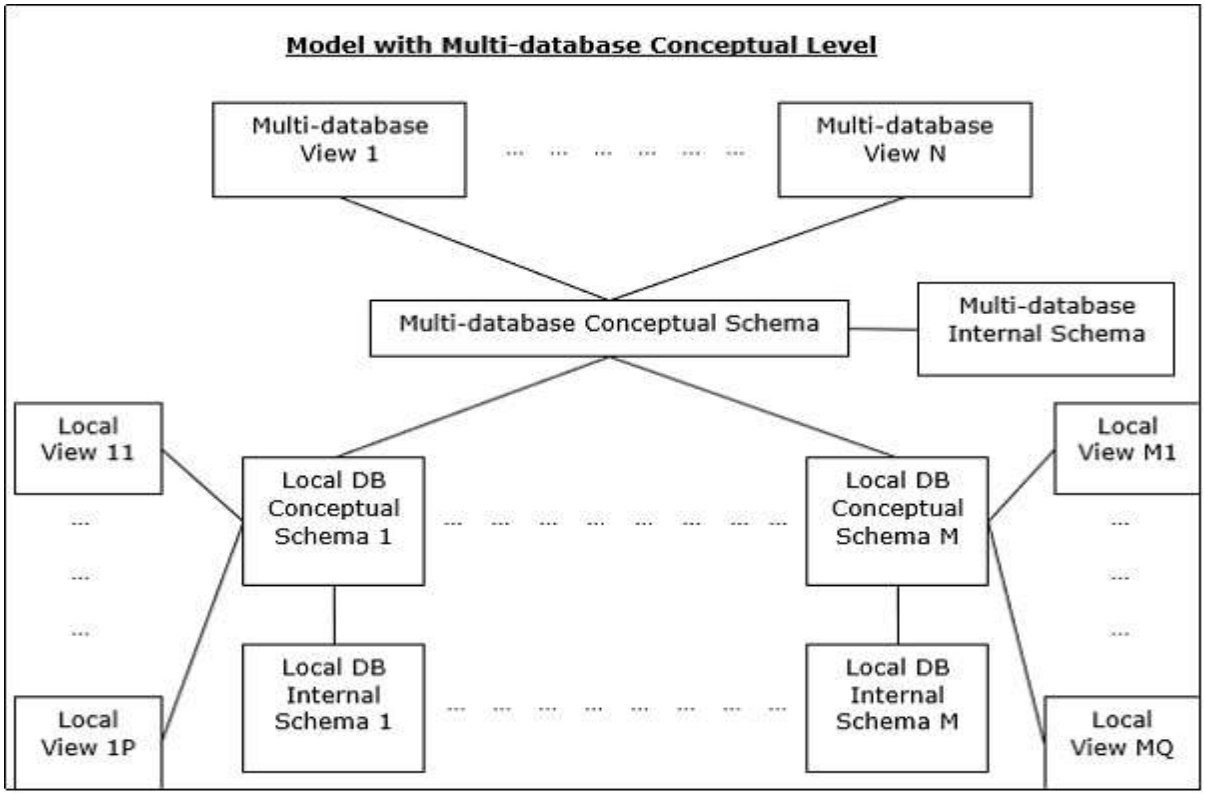
This is an integrated database system formed by a collection of two or more autonomous database systems.

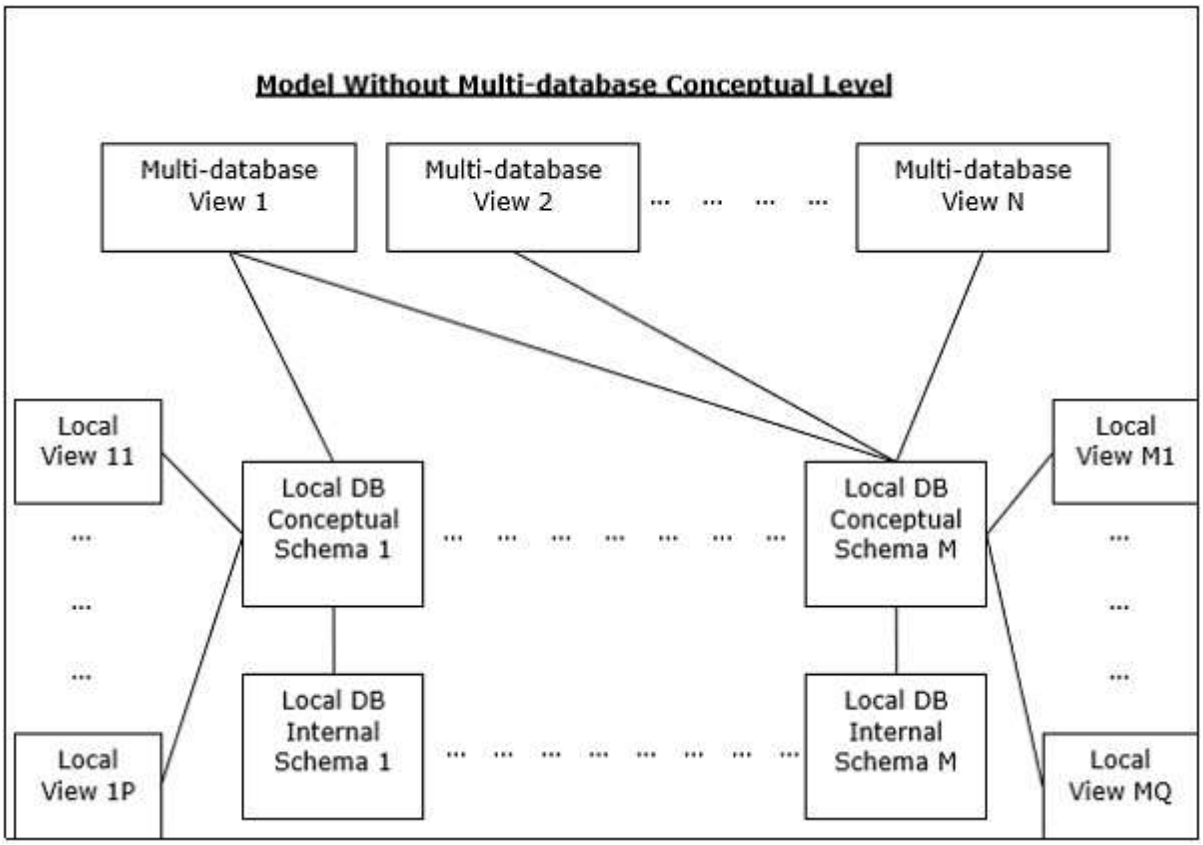
Multi-DBMS can be expressed through six levels of schemas −

-**Multi-database View Level** − Depicts multiple user views comprising of subsets of the integrated distributed database.
-**Multi-database Conceptual Level** − Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
-**Multi-database Internal Level** − Depicts the data distribution across different sites and multi-database to local data mapping.
-Local database View Level − Depicts public view of local data.
-Local database Conceptual Level − Depicts local data organization at each site.
-Local database Internal Level − Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS −

Model with multi-database conceptual level.
Model without multi-database conceptual level.





DDMBS Architecture:

Distributed DBMS Architectures

DDBMS architectures are generally developed depending on three parameters −

- Distribution − It states the physical distribution of data across the different sites.

-Autonomy − It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.

-Heterogeneity − It refers to the uniformity or dissimilarity of the data models, system components and databases.

Distributed Database Design:

Distributed database design refers to the following problem: given a database and its workload, how the database should be split and allocated to sites so as to optimize certain objective function (e.g., to minimize the resource consumption in processing the query workload).

Alternative Design Strategies:

The distribution design alternatives for the tables in a DDBMS are as follows –

-Non-replicated and non-fragmented

-Fully replicated

-Partially replicated
-Fragmented
-Mixed

Non-replicated & Non-fragmented
In this design alternative, different tables are placed at different sites. Data is placed so that it is at a close proximity to the site where it is used most. It is most suitable for database systems where the percentage of queries needed to join information in tables placed at different sites is low. If an appropriate distribution strategy is adopted, then this design alternative helps to reduce the communication cost during data processing.

Fully Replicated
In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database, queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update operations. Hence, this is suitable for systems where a large number of queries is required to be handled whereas the number of database updates is low.
Partially Replicated
Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.
Fragmented
In this design, a table is divided into two or more pieces referred to as fragments or partitions, and each fragment can be stored at different sites. This considers the fact that it seldom happens that all data stored in a table is required at a given site. Moreover, fragmentation increases parallelism and provides better disaster recovery. Here, there is only one copy of each fragment in the system, i.e. no redundant data.
The three fragmentation techniques are −
-Vertical fragmentation
-Horizontal fragmentation
-Hybrid fragmentation
Mixed Distribution
This is a combination of fragmentation and partial replications. Here, the tables are initially fragmented in any form (horizontal or vertical), and then these fragments are partially replicated across the different sites according to the frequency of accessing the fragments.

Distribution Design issues:

-Distributed Database Design.
-Distributed Directory Management.
-Distributed Query Processing.
-Distributed Concurrency Control.
-Distributed Deadlock Management.
-Reliability of Distributed DBMS.
-Replication.

-Distributed Database Design

• One of the main questions that is being addressed is how database and the applications that run against it should be placed across the sites.

• There are two basic alternatives to placing data: partitioned (or no-replicated) and replicated.

• In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at different site. Replicated designs can be either fully replicated (also called fully duplicated) where entire database is stored at each site, or partially replicated (or partially duplicated) where each partition of the database is stored at more than one site, but not at all the sites.

• The two fundamental design issues are fragmentation, the separation of the database into partitions called fragments, and distribution, the optimum distribution of fragments. The research in this area mostly involve mathematical programming in order to minimize the combined cost of storing the database, processing transactions against it, and message communication among site.

Distributed Directory Management

• A directory contains information (such as descriptions and locations) about data items in the database. Problems related to directory management are similar in nature to the database placement problem discussed in the preceding section.

• A directory may be global to the entire DDBS or local to each site; it can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

Distributed Query Processing

• Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each query over the network in the most cost-effective way, however cost is defined.

• The factors to be considered are the distribution of data, communication cost, and lack of sufficient locally-available information. The objective is to optimize where the inherent parallelism is used to improve the performance of executing the transaction, subject to the abovementioned constraints.

Distributed Concurrency Control

• Concurrency control involves the synchronization of access to the distributed database, such that the integrity of the database is maintained. It is, without any doubt, one of the most extensively studied problems in the DDBS field.

• The concurrency control problem in a distributed context is somewhat different that in a centralized framework. One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database. The condition that requires all

values of multiple copies of every data item to converge to the same value is called mutual consistency.

• Let us only mention that the two general classes are pessimistic, synchronizing the execution of the user request before the execution starts, and optimistic, executing requests and then checking if the execution has compromised the consistency of the database.

• Two fundamental primitives that can be used with both approaches are locking, which is based on the mutual exclusion of access to data items, and time-stamping, where transactions executions are ordered based on timestamps.

• There are variations of these schemes as well as hybrid algorithms that attempt to combine the two basic mechanisms.

Distributed Deadlock Management

• The deadlock problem in DDBSs is similar in nature to that encountered in operating systems.

• The competition among users for access to a set of resources (data, in this case) can result in a deadlock if the synchronization mechanism is based on locking. The well-known alternatives of prevention, avoidance, and detection/recovery also apply to DDBSs.

Reliability of Distributed DBMS

• It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them. The implication for DDBSs is that when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date.

• Furthermore, when the computer system or network recovers from the failure, the DDBSs should be able to recover and bring the databases at the failed sites up-to date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them.

Replication

• If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas, i.e. copies of the same data item have the same value.

• These protocols can be eager in that they force the updates to be applied to all the replicas before the transactions completes, or they may be lazy so that the transactions updates one copy (called the master) from which updates are propagated to the others after the transaction completes.

Fragmentation:

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "reconstructiveness."

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

STUDENT

Regd_No	Name	Course	Address	Semester	Fees	Marks

Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows −

```
CREATE TABLE STD_FEES AS
  SELECT Regd_No, Fees
  FROM STUDENT;
```

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows −

```
CREATE COMP_STD AS
  SELECT * FROM STUDENT
  WHERE COURSE = "Computer Science";
```

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways −

At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.

At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Allocation:-

Allocation of data is one of the key design issues of distributed database. The main objective of a data allocation in distributed database is to place the data fragments at different sites in such a way, so that the total data transfer cost can be minimized while executing a set of queries.

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.