

Distributed Database – SCS1613

UNIT - I

INTRODUCTION TO DISTRIBUTED DATABASE

Introduction of Distributed Databases-Features of Distributed Databases-Distributed databases versus Centralized Databases- Principles—Levels Of Distribution-Transparency-Reference Architecture- Types of Data Fragmentation- Integrity Constraints in Distributed Databases- Architectural Issues- Alternative Client/Server Architecture.

A **database** is an ordered collection of related data that is built for a specific purpose. A database may be organized as a collection of multiple tables, where a table represents a real world element or entity. Each table has several different fields that represent the characteristic features of the entity.

For example, a company database may include tables for projects, employees, departments, products and financial records. The fields in the Employee table may be Name, Company_Id, Date_of_Joining, and so forth.

A **database management system** is a collection of programs that enables creation and maintenance of a database. DBMS is available as a software package that facilitates definition, construction, manipulation and sharing of data in a database. Definition of a database includes description of the structure of a database. Construction of a database involves actual storing of the data in any storage medium. Manipulation refers to the retrieving information from the database, updating the database and generating reports. Sharing of data facilitates data to be accessed by different users or programs.

Examples of DBMS Application Areas

Automatic Teller Machines Train Reservation System Employee Management System
Student Information System

Examples of DBMS Packages

MySQL

Oracle

SQL Server dBASE

FoxPro PostgreSQL, etc.

Database Schemas

A database schema is a description of the database which is specified during database design and subject to infrequent alterations. It defines the organization of the data, the relationships among them, and the constraints associated with them. Databases are often represented through the three-schema architecture or ANSISPARC architecture. The goal of this architecture is to separate the user application from the physical database.

The three levels are –

Internal Level having Internal Schema – It describes the physical structure, details of internal storage and access paths for the database.

Conceptual Level having Conceptual Schema – It describes the structure of the whole database while hiding the details of physical storage of data. This illustrates the entities, attributes with their data types and constraints, user operations and relationships.

External or View Level having External Schemas or Views – It describes the portion of a database relevant to a particular user or a group of users while hiding the rest of database.

Types of DBMS

Hierarchical DBMS

In hierarchical DBMS, the relationships among data in the database are established so that one data element exists as a subordinate of another. The data elements have parent-child relationships and are modelled using the “tree” data structure. These are very fast and simple.

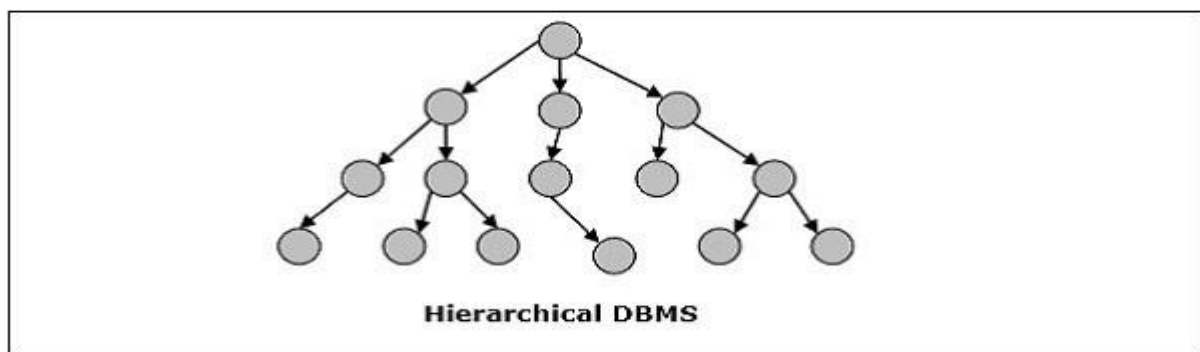


Figure 1 Hierarchical DBMS

Network DBMS

Network DBMS is one where the relationships among data in the database are of type many-to-many in the form of a network. The structure is generally complicated due to the existence of numerous many-to-many relationships. Network DBMS is modelled using “graph” data structure.

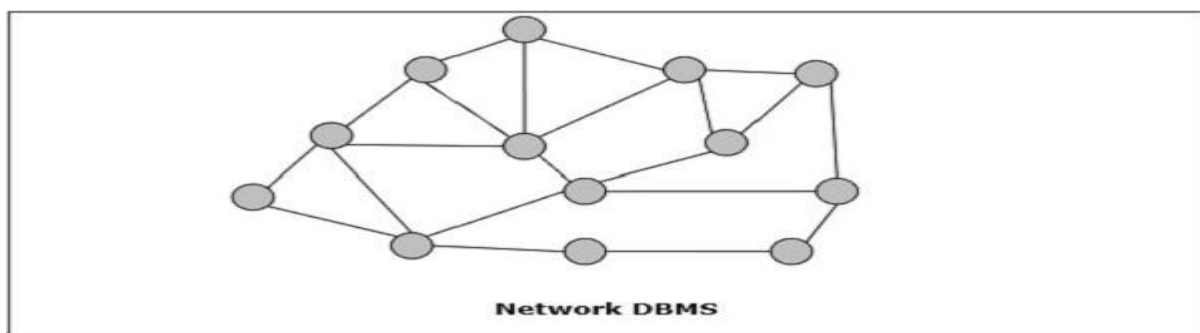
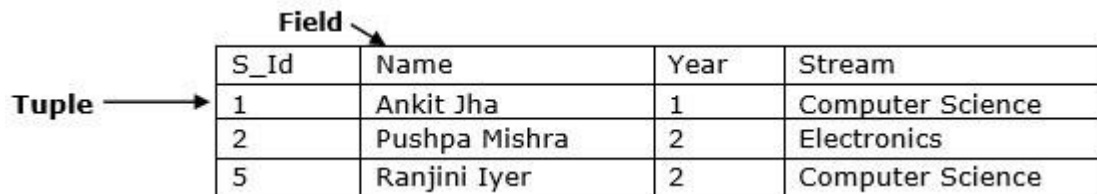


Figure 2 Network DBMS

Relational DBMS

In relational databases, the database is represented in the form of relations. Each relation models an entity and is represented as a table of values. In the relation or table, a row is called a tuple and denotes a single record. A column is called a field or an attribute and denotes a characteristic property of the entity. RDBMS is the most popular database management system.

For example – A Student Relation –



S_Id	Name	Year	Stream
1	Ankit Jha	1	Computer Science
2	Pushpa Mishra	2	Electronics
5	Ranjini Iyer	2	Computer Science

Figure 3 A Student Relation

Object Oriented DBMS

Object-oriented DBMS is derived from the model of the object-oriented programming paradigm. They are helpful in representing both consistent data as stored in databases, as well as transient data, as found in executing programs. They use small, reusable elements called objects. Each object contains a data part and a set of operations which works upon the data. The object and its attributes are accessed through pointers instead of being stored in relational table models.

For example – A simplified Bank Account object-oriented database –

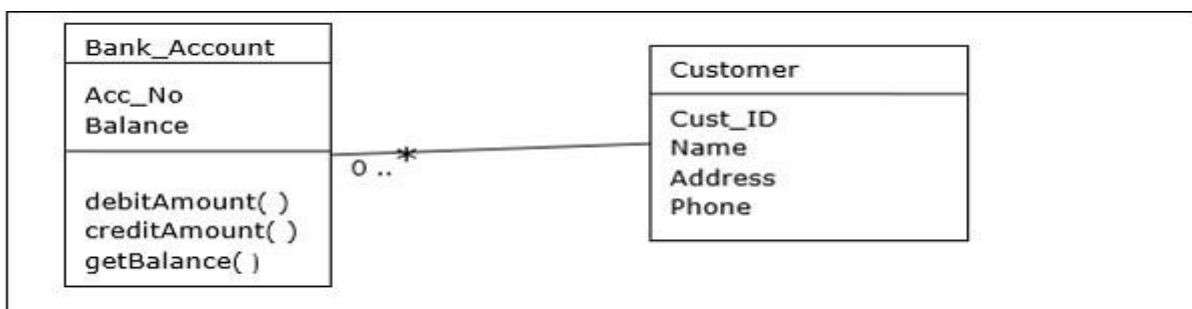


Figure 4 A simplified Bank Account object-oriented database

Distributed DBMS

A distributed database is a set of interconnected databases that is distributed over the computer network or internet. A Distributed Database Management System (DDBMS) manages the distributed database and provides mechanisms so as to make the databases transparent to the users. In these systems, data is intentionally distributed among multiple nodes so that all computing resources of the organization can be optimally used.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

- It is used to create, retrieve, update and delete distributed databases.
- It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
- It ensures that the data modified at any site is universally updated.
- It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
- It is designed for heterogeneous database platforms.
- It maintains confidentiality and data integrity of the databases.

Factors Encouraging DDBMS

- Distributed Nature of Organizational Units – Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.
- Need for Sharing of Data – The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.
- Support for Both OLTP and OLAP – Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.
- Database Recovery – One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the

damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.

- Support for Multiple Application Software – Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Advantages of Distributed Databases

- Modular Development – If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.
- More Reliable – In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.
- Better Response – If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.
- Lower Communication Cost – In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Adversities of Distributed Databases

- Need for complex and expensive software – DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
- Processing overhead – Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
- Data integrity – The need for updating data in multiple sites pose problems of data integrity.
- Overheads for improper data distribution – Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to very slow response to user requests.

Distributed Database Vs Centralized Database

<i>Centralized DBMS</i>	<i>Distributed DBMS</i>
In Centralized DBMS the database are stored in a only one site	In Distributed DBMS the database are stored in different site and help of network it can access it
If the data is stored at a single computer site,which can be used by multiple users	Database and DBMS software distributed over many sites,connected by a computer network
Database is maintained at one site	Database is maintained at a number of different sites
If centralized system fails,entire system is halted	If one system fails,system continues work with other site
It is a less reliable	It is a more reliable

Centralized database

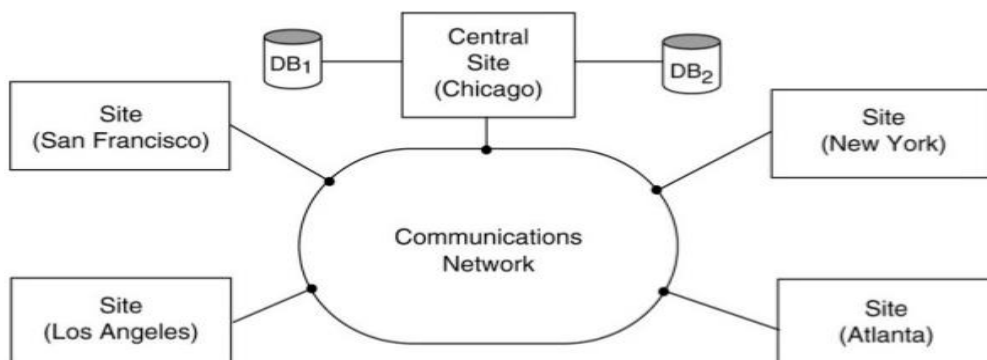


Figure 5 Centralized database

Distributed database

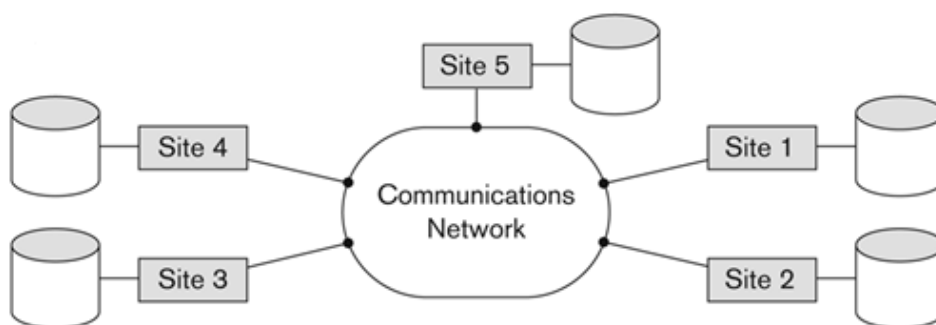


Figure 6 Distributed database

Types of Distributed Databases

Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments.

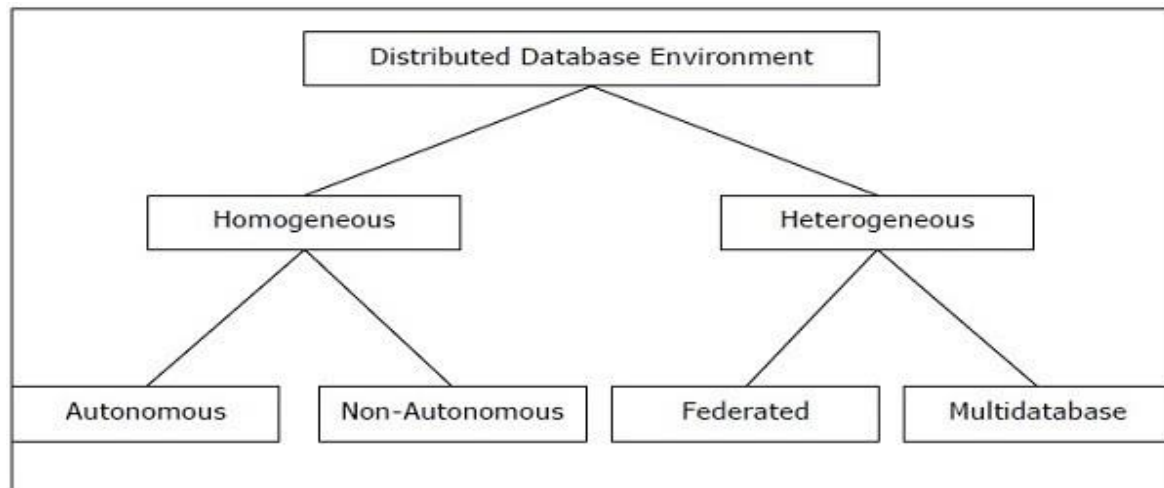


Figure 7 Types of Distributed Databases

Homogeneous Distributed Databases

In a homogeneous distributed database, all the sites use identical DBMS and operating systems. Its properties are –

- The sites use very similar software.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.

Types of Homogeneous Distributed Database

There are two types of homogeneous distributed database –

Autonomous – Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.

Non-autonomous – Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Databases

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.

Types of Heterogeneous Distributed Databases

Federated – The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.

Un-federated – The database systems employ a central coordinating module through which the databases are accessed.

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.

Architectural Models

- 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.

Distinguish the functionality and divide these functions into two classes, server functions and client functions.

Server does most of the data management work

- query processing
- data management
- Optimization
- Transaction management etc

Client performs

- Application
- User interface
- DBMS Client model

The two different client - server architecture are –

Single Server Multiple Client

Single Server accessed by multiple clients

- A client server architecture has a number of clients and a few servers connected in a network.
- A client sends a query to one of the servers. The earliest available server solves it and replies.
- A Client-server architecture is simple to implement and execute due to centralized server system.

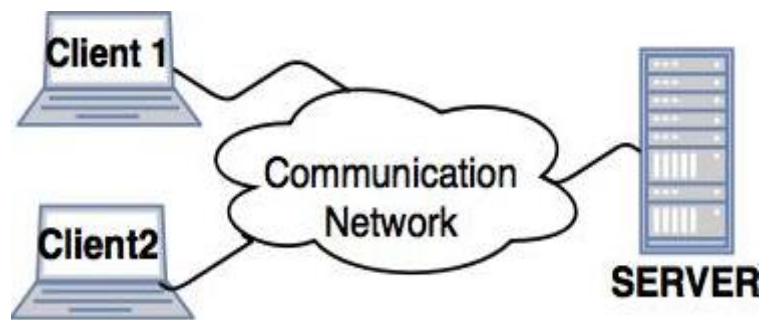


Figure 8 Single Server Multiple Client

Multiple Server Multiple Client

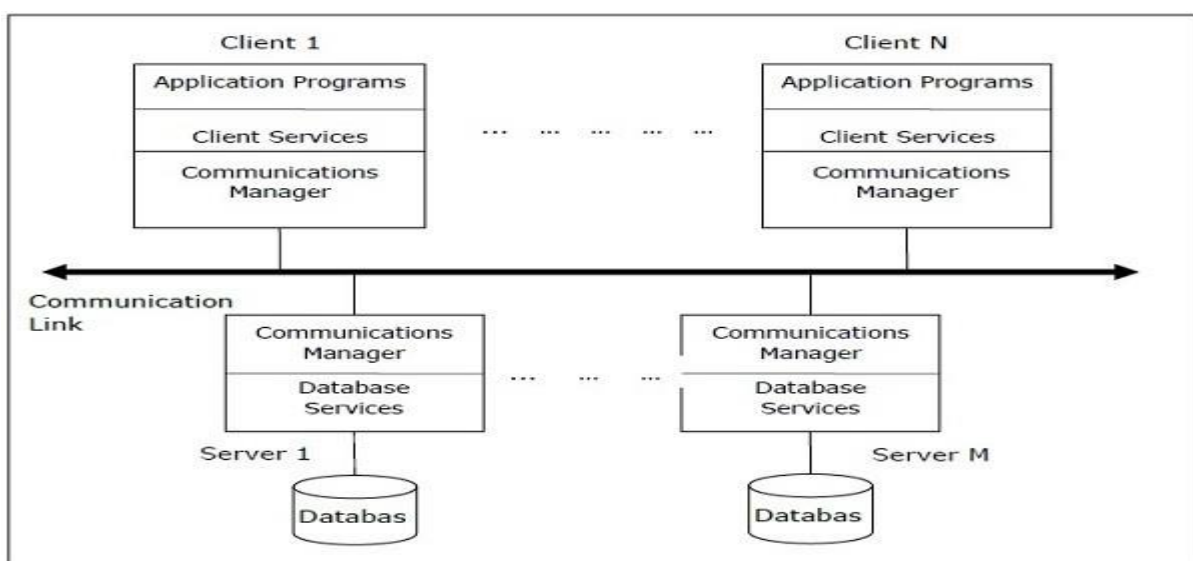


Figure 9 Multiple Servers accessed by multiple clients

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 –

Schemas Present

Individual internal schema definition at each site, *local internal schema*

Enterprise view of data is described the *global conceptual schema*.

Local organization of data at each site is describe in the *local conceptual schema*.

User applications and user access to the database is supported by *external schemas*

Local conceptual schemas are mappings of the global schema onto each site.

Databases are typically designed in a top-down fashion, and, therefore all external view definitions are made globally.

Major Components of a Peer-to-Peer System

- User Processor
- Data processor

User Processor

- User-interface handler
- responsible for interpreting user commands, and formatting the result data
- Semantic data controller
- checks if the user query can be processed.
- Global Query optimizer and decomposer
- determines an execution strategy
- Translates global queries into local one.
- Distributed execution
- Coordinates the distributed execution of the user request

Data processor

- Local query optimizer
- Acts as the access path selector
- Responsible for choosing the best access path
- Local Recovery Manager
- Makes sure local database remains consistent
- Run-time support processor
- Is the interface to the operating system and contains the database buffer

- Responsible for maintaining the main memory buffers and managing the data access.

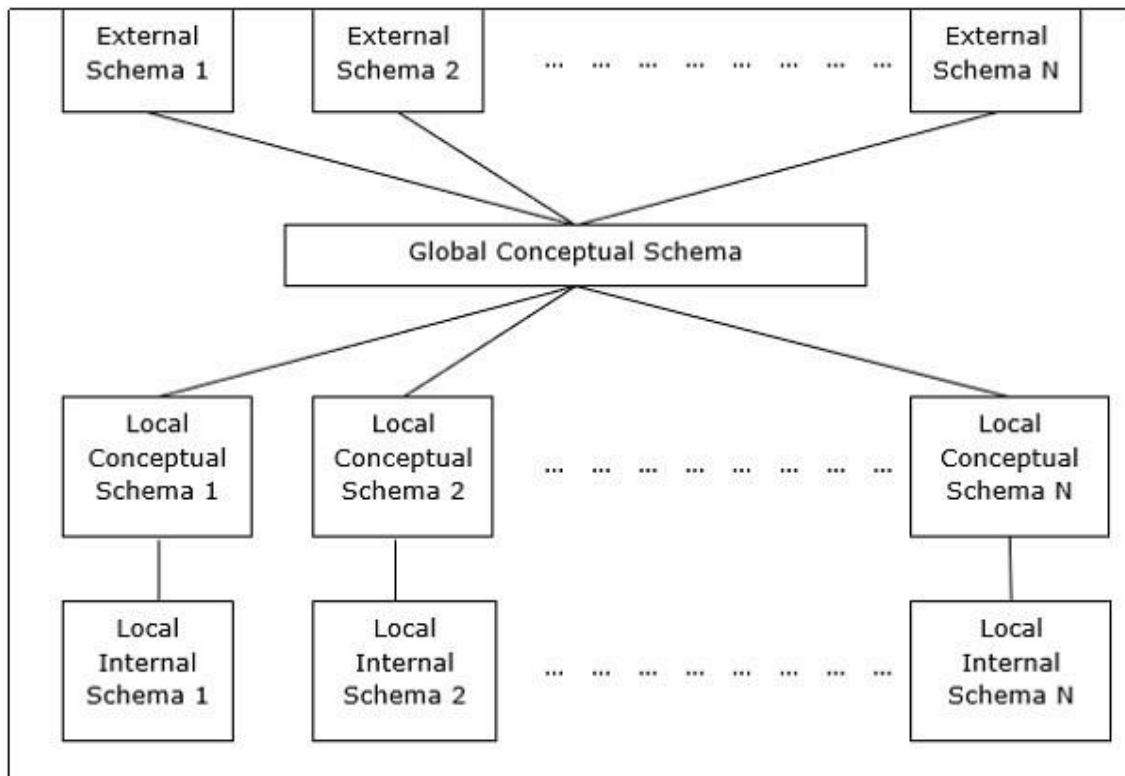


Figure 10 Peer- to-Peer Architecture for DDBMS

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 –

1. Model with multi-database conceptual level

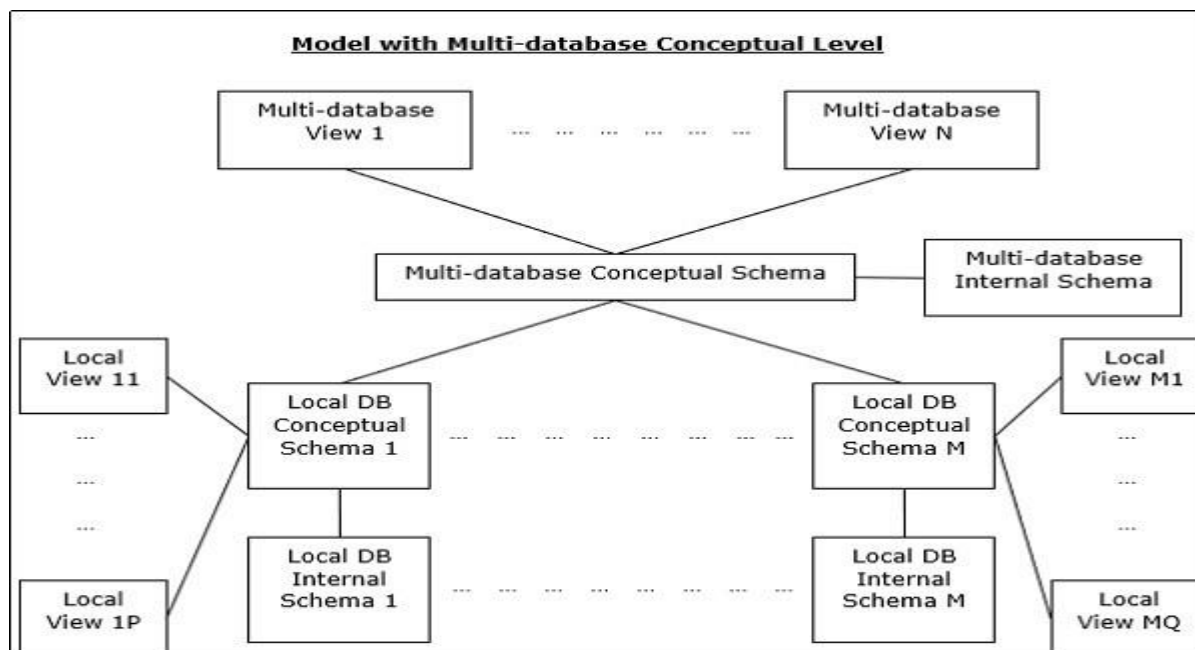


Figure 11 Model with multi-database conceptual level

Models Using a Global Conceptual Schema

- GCS is defined by integrating either the external schemas of local autonomous databases or parts of their local conceptual schema
- Users of a local DBMS define their own views on the local database.
- If heterogeneity exists in the system, then two implementation alternatives exist: unilingual and multilingual
- Unilingual requires the users to utilize possibly different data models and languages
- Basic philosophy of multilingual architecture, is to permit each user to access the global database.

GCS in multi-DBMS

- Mapping is from local conceptual schema to a global schema
- Bottom-up design

Model without multi-database conceptual level.

- Consists of two layers, local system layer and multi database layer.
- Local system layer , present to the multi-database layer the part of their local database they are willing share with users of other database.
- System views are constructed above this layer
- Responsibility of providing access to multiple database is delegated to the mapping between the external schemas and the local conceptual schemas.
- Full-fledged DBMs, exists each of which manages a different database.

GCS in Logically integrated distributed DBMS

- Mapping is from global schema to local conceptual schema
- Top-down procedure

Global Directory Issues

Global Directory is an extension of the normal directory, including information about the location of the fragments as well as the makeup of the fragments, for cases of distributed DBMS or a multi-DBMS, that uses a global conceptual schema,

- Relevant for distributed DBMS or a multi-DBMS that uses a global conceptual schema
- Includes information about the location of the fragments as well as the makeup of fragments.
- Directory is itself a database that contains meta-data about the actual data stored in database.

Three issues

- A directory may either be global to the entire database or local to each site.
- Directory may be maintained centrally at one site, or in a distributed fashion by distributing it over a number of sites.
 - If system is distributed, directory is always distributed
- Replication may be single copy or multiple copies.
 - Multiple copies would provide more reliability

Organization of Distributed systems

Three orthogonal dimensions

- Level of sharing
 - No sharing, each application and data execute at one site
 - Data sharing, all the programs are replicated at other sites but not the data.
 - Data-plus-program sharing, both data and program can be shared
- Behavior of access patterns
 - Static
 - Does not change over time
 - Very easy to manage
 - Dynamic
 - Most of the real life applications are dynamic
- Level of knowledge on access pattern behavior.
 - No information
 - Complete information
 - Access patterns can be reasonably predicted
 - No deviations from predictions
 - Partial information
 - Deviations from predictions

Top Down Design

- Suitable for applications where database needs to be build from scratch
- Activity begins with requirement analysis
- Requirement document is input to two parallel activities:
 - view design activity, deals with defining the interfaces for end users
 - conceptual design, process by which enterprise is examined
 - Can be further divided into 2 related activity groups
 - Entity analyses, concerned with determining the entities, attributes and the relationship between them
 - Functional analyses, concerned with determining the fun
 - Distributed design activity consists of two steps
 - Fragmentation
 - Allocation

Bottom-Up Approach

- Suitable for applications where database already exists
- Starting point is individual conceptual schemas
- Exists primarily in the context of heterogeneous database.

Design Alternatives

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.

Design Strategies

In the last chapter, we had introduced different design alternatives. In this chapter, we will study the strategies that aid in adopting the designs. The strategies can be broadly divided into replication and fragmentation. However, in most cases, a combination of the two is used.

Data Replication

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

Advantages of Data Replication

- Reliability – In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- Reduction in Network Load – Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- Quicker Response – Availability of local copies of data ensures quick query processing and consequently quick response time.

- Simpler Transactions – Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication

- Increased Storage Requirements – Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
- Increased Cost and Complexity of Data Updating – Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
- Undesirable Application – Database coupling – If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application – database coupling.

Some commonly used replication techniques are

Snapshot replication

Near-real-time replication

Pull replication

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

1. Permits a number of transactions to be executed concurrently
2. Results in parallel execution of a single query
3. Increases level of concurrency, also referred to as, intra query concurrency
4. Increased System throughput.
5. Since data is stored close to the site of usage, efficiency of the database system is increased.
6. Local query optimization techniques are sufficient for most queries since data is locally available.
7. Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages

1. Applications whose views are defined on more than one fragment may suffer performance degradation, if applications have conflicting requirements.
2. Simple tasks like checking for dependencies, would result in chasing after data in a number of sites
3. When data from different fragments are required, the access speeds may be very high.
4. In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
5. 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.

Grouping

- Starts by assigning each attribute to one fragment
 - At each step, joins some of the fragments until some criteria is satisfied.
- Results in overlapping fragments

Splitting

- Starts with a relation and decides on beneficial partitioning based on the access behavior of applications to the attributes
- Fits more naturally within the top-down design
- Generates non-overlapping fragments

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

```
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 conform to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

- Primary horizontal fragmentation is defined by a selection operation on the owner relation of a database schema.
- Given relation R_i , its horizontal fragments are given by

$$R_i = \sigma_{F_i}(R), \quad 1 \leq i \leq w$$

F_i selection formula used to obtain fragment R_i

The example mentioned in slide 20, can be represented by using the above formula as

$$Emp_1 = \sigma_{Sal \leq 20K}(Emp)$$

$$Emp_2 = \sigma_{Sal > 20K}(Emp)$$

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";
```

Derived Horizontal Fragmentation

- Defined on a member relation of a link according to a selection operation specified on its owner.
- Link between the owner and the member relations is defined as equi-join
- An equi-join can be implemented by means of semijoins.
- Given a link L where owner (L) = S and member (L) = R , the derived horizontal fragments of R are defined as

$$R_i = R \bowtie S_i, \quad 1 \leq i \leq w$$

Where,

$$S_i = \sigma_{F_i}(S)$$

w is the max number of fragments that will be defined on

F_i is the formula using which the primary horizontal fragment S_i is defined

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.

Transparency

Transparency in DBMS stands for the separation of high level semantics of the system from the low-level implementation issue. High-level semantics stands for the endpoint user, and low level implementation concerns with complicated hardware implementation of data or how the data has been stored in the database. Using data independence in various layers of the database, transparency can be implemented in DBMS.

Distribution transparency is the property of distributed databases by the virtue of which the internal details of the distribution are hidden from the users. The DDBMS designer may choose to fragment tables, replicate the fragments and store them at different sites. However, since users are oblivious of these details, they find the distributed database easy to use like any centralized database.

Unlike normal DBMS, DDBMS deals with communication network, replicas and fragments of data. Thus, transparency also involves these three factors.

Following are three types of transparency:

1. Location transparency
2. Fragmentation transparency
3. Replication transparency

Location Transparency

Location transparency ensures that the user can query on any table(s) or fragment(s) of a table as if they were stored locally in the user's site. The fact that the table or its fragments are stored at remote site in the distributed database system, should be completely oblivious to the end user. The address of the remote site(s) and the access mechanisms are completely hidden. In order to incorporate location transparency, DDBMS should have access to updated and accurate data dictionary and DDBMS directory which contains the details of locations of data.

Fragmentation Transparency

Fragmentation transparency enables users to query upon any table as if it were unfragmented. Thus, it hides the fact that the table the user is querying on is actually a fragment or union of some fragments. It also conceals the fact that the fragments are located at diverse sites. This is somewhat similar to users of SQL views, where the user may not know that they are using a view of a table instead of the table itself.

Replication Transparency

Replication transparency ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists. Replication transparency is associated with concurrency transparency and failure transparency. Whenever a user updates a data item, the update is reflected in all the copies of the table. However, this operation should not be known to the user. This is concurrency transparency. Also, in case of failure of a site, the user can still proceed with his queries using replicated copies without any knowledge of failure. This is failure transparency.

Combination of Transparencies

In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent. The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user. However, complete distribution transparency is a tough task and requires considerable design efforts.

Database Control

Database control refers to the task of enforcing regulations so as to provide correct data to authentic users and applications of a database. In order that correct data is available to users, all data should conform to the integrity constraints defined in the database. Besides, data should be screened away from unauthorized users so as to maintain security and privacy of the database. Database control is one of the primary tasks of the database administrator (DBA).

The three dimensions of database control are –

- Authentication
- Access Control
- Integrity Constraints

Authentication

In a distributed database system, authentication is the process through which only legitimate users can gain access to the data resources.

Authentication can be enforced in two levels –

Controlling Access to Client Computer – At this level, user access is restricted while login to the client computer that provides user-interface to the database server. The most common method is a username/password combination. However, more sophisticated methods like biometric authentication may be used for high security data.

Controlling Access to the Database Software – At this level, the database software/administrator assigns some credentials to the user. The user gains access to the database using these credentials. One of the methods is to create a login account within the database server.

Access Rights

A user's access rights refers to the privileges that the user is given regarding DBMS operations such as the rights to create a table, drop a table, add/delete/update tuples in a table or query upon the table.

In distributed environments, since there are large number of tables and yet larger number of users, it is not feasible to assign individual access rights to users. So, DDBMS defines certain roles. A role is a construct with certain privileges within a database system. Once the different roles are defined, the individual users are assigned one of these roles. Often a hierarchy of roles are defined according to the organization's hierarchy of authority and responsibility.

For example, the following SQL statements create a role "Accountant" and then assigns this role to user "ABC".

```
CREATE ROLE ACCOUNTANT;  
  
GRANT SELECT, INSERT, UPDATE ON EMP_SAL TO ACCOUNTANT; GRANT INSERT, UPDATE,  
DELETE ON TENDER TO ACCOUNTANT; GRANT INSERT, SELECT ON EXPENSE TO  
ACCOUNTANT;  
  
COMMIT;
```

Semantic Integrity Control

Semantic integrity control defines and enforces the integrity constraints of the database system.

The integrity constraints are as follows –

Data type integrity constraint

Entity integrity constraint

Referential integrity constraint

Data Type Integrity Constraint

A data type constraint restricts the range of values and the type of operations that can be applied to the field with the specified data type.

For example, let us consider that a table "HOSTEL" has three fields - the hostel number, hostel name and capacity. The hostel number should start with capital letter "H" and cannot be NULL, and the capacity should not be more than 150. The following SQL command can be used for data definition –

```
CREATE TABLE HOSTEL (  
H_NO VARCHAR2(5) NOT NULL, H_NAME VARCHAR2(15), CAPACITY INTEGER,  
CHECK ( H_NO LIKE 'H%'), CHECK ( CAPACITY <= 150)  
);
```

Entity Integrity Control

Entity integrity control enforces the rules so that each tuple can be uniquely identified from other tuples. For this a primary key is defined. A primary key is a set of minimal fields that can uniquely identify a tuple. Entity integrity constraint states that no two tuples in a table can have identical values for primary keys and that no field which is a part of the primary key can have NULL value.

For example, in the above hostel table, the hostel number can be assigned as the primary key through the following SQL statement (ignoring the checks) –

```
CREATE TABLE HOSTEL (  
H_NO VARCHAR2(5) PRIMARY KEY, H_NAME VARCHAR2(15),  
CAPACITY INTEGER);
```

Referential Integrity Constraint

Referential integrity constraint lays down the rules of foreign keys. A foreign key is a field in a data table that is the primary key of a related table. The referential integrity constraint lays down the rule that the value of the foreign key field should either be among the values of the primary key of the referenced table or be entirely NULL.

For example, let us consider a student table where a student may opt to live in a hostel. To include this, the primary key of hostel table should be included as a foreign key in the student table. The following SQL statement incorporates this –

```
CREATE TABLE STUDENT (  
S_ROLL INTEGER PRIMARY KEY, S_NAME VARCHAR2(25) NOT NULL, S_COURSE VARCHAR2(10),  
S_HOSTEL VARCHAR2(5) REFERENCES HOSTEL);
```