📘 Distributed Computing – Unit 1.0 (Detailed Notes)

Distributed Database System (DDBS)

---

1. What is Distributed Database System (DDBS)?

A **Distributed Database System (DDBS)** is a collection of logically interrelated databases that are **distributed across multiple locations** but connected through a **computer network**.

- Data is stored at different sites (servers, nodes, data centers).
- Users can access data as if it is stored at a **single location**.
- The system manages **transparency, coordination, and data consistency**.

Definition

"A Distributed Database System is a database in which storage devices are not all attached to a common processor. Data is distributed across different physical locations and managed by a distributed DBMS."

---

2. Features of Distributed Database Systems

1. **Distribution Transparency**
   Users do not need to know where the data actually resides.
2. **Replication**
   Copies of data may be stored at different sites to improve reliability and performance.
3. **Scalability**
   Easy to add new sites without affecting the overall system.
4. **Fault Tolerance**
   If one site fails, the system continues to work with replicated data.
5. **Improved Performance**
   Local queries run faster since data is available nearby.
6. **Heterogeneity Support**
   Systems can work with different hardware, software, or DBMS.
7. **Concurrency Control**
   Ensures correct results even when many users access data simultaneously.
8. **Security and Authorization**
   Access control is maintained across multiple sites.

3. Promises of DDBS (Advantages)

1. **Reliability & Availability**
   Failure of one site does not stop the entire system.
2. **Improved Performance**
   Local queries reduce response time.
3. **Modular Growth**
   New sites can be added easily without downtime.
4. **Lower Communication Cost**
   Data is stored closer to where it is used.
5. **Better Sharing of Data**
   Departments can share distributed data easily.
6. **Continuation of Service**
   Even during partial network failure, system keeps functioning.

---

4. Design Issues in Distributed Database System

Designing a DDBS is complex. Major issues include:

a) Fragmentation

Splitting a database into smaller logical units (fragments).

Types:

- Horizontal Fragmentation
- Vertical Fragmentation
- Hybrid Fragmentation

b) Replication

Maintaining multiple copies of data at different sites.

Replication Methods:

- Full Replication
- Partial Replication
- No Replication

c) Allocation

Deciding where to place each fragment or replica.

d) Query Processing

Optimizing distributed queries for efficiency.

e) Concurrency Control

Ensuring consistency when multiple users update distributed data.

Methods:

- Distributed Two-Phase Locking (2PL)
- Timestamp Ordering

f) Reliability & Fault Tolerance

Handling failures at sites, links, or network.

Techniques:

- Checkpointing
- Recovery Protocols

g) Data Consistency

Ensuring all replicas hold the same values after updates.

h) Security

Maintaining access control across all sites.

---

5. Distributed DBMS Architecture

There are **three main architectures**:

---

1. Client/Server Architecture

Description

Database is stored on one or more **central servers**, and clients request data.

Working

- **Client** → sends request
- **Server** → processes query and sends result

Advantages

- Simple to implement
- Centralized control
- Better performance for small systems

Disadvantages

- Server overload
- Single point of failure

---

2. Peer-to-Peer (P2P) Architecture

Description

Every site (node) performs both **client and server roles**.

Features

- Equal responsibility
- No central controller
- Better reliability

Advantages

- No single point of failure
- Scalable system
- Supports distributed processing

Disadvantages

- Complex concurrency control
- Difficult to maintain global data consistency

---

3. Multi-Database System (MDBS)

Description

Combines multiple autonomous databases into a single system.

Each site:

- Has its own DBMS
- Manages its own data
- May follow different models (SQL, NoSQL, Oracle, etc.)

Features

- No central control over data
- Sites cooperate to answer global queries

Advantages

- Supports integration of different independent databases
- Flexible and highly scalable

Disadvantages

- Very complex
- Hard to maintain global schema
- Data conflicts may occur

Common Formula (Copy Format)

1. Data Allocation Cost Formula

```
Total Cost = Storage Cost + Transfer Cost + Processing Cost
```

2. Query Response Time

```
Response Time = Local Processing Time + Communication Time
```

3. Replication Overhead

```
Replication Cost = (Number of Replicas - 1) × Update Cost
```

4. Reliability of Distributed System

```
Total Reliability = 1 - (Product of individual failure probabilities)
```

📘 Distributed Computing – Unit 2.0 (Detailed Notes)

Distributed Database Design (DDD)

---

1. Distributed Database Design Concept

Distributed Database Design is the process of **structuring, fragmenting, replicating, and allocating** data across multiple sites in a distributed environment.

The main steps of distributed database design are:

1. **Database Fragmentation**
2. **Data Allocation (Placement of fragments)**
3. **Replication (If required)**
4. **Ensuring Transparency**

Aim of Distributed Design

To design a database that is:

- Fast
- Reliable

- Always available
- Easy to expand
- Cost effective

Distributed database design plays a crucial role in determining how data will be stored, accessed, and maintained across multiple locations.

---

2. Objective of Data Distribution

The main objectives of distributing data are:

1. Improved Performance

- Data stored close to users reduces access time.
- Local processing is faster.

2. Reliability and Availability

- Failure of one site does not affect others.
- Replication increases fault tolerance.

3. Reduced Communication Cost

- Queries can be processed locally.
- Less data travels across the network.

4. Scalability / Growth

- New sites can be added easily.

5. Data Sharing

- Different departments can access relevant fragments.

6. Autonomy

- Each site can manage its own data and workload.

---

3. Data Fragmentation

Fragmentation is the process of **breaking a database into smaller logical pieces** (fragments). Each fragment can be stored at different sites.

Goals of Fragmentation

- Improve performance
- Increase availability
- Reduce communication cost
- Support parallelism

Types of Fragmentation

A. Horizontal Fragmentation

- Divides a table **row-wise**.
- Each fragment contains **some rows** of the table.
- Based on a **selection condition**.

Example

A "Customer" table may be split by region:

- Fragment 1 → Customers in North region
- Fragment 2 → Customers in South region

Formula

```
R = R1 ∪ R2 ∪ R3
(Each fragment contains different rows)
```

B. Vertical Fragmentation

- Divides a table **column-wise**.
- Each fragment contains **some columns** plus the **primary key** (to maintain reconstruction).

Example

Employee table split as:

- F1: EmpID, Name, Address
- F2: EmpID, Salary, Department

Formula

```
R = R1 ⋈ R2
(Join using primary key)
```

C. Mixed / Hybrid Fragmentation

Combination of **horizontal** + **vertical** fragmentation.

Example

First split customers by region (horizontal),
then split each fragment by columns (vertical).

Correctness Rules for Fragmentation

A fragmentation is correct if:

1. **Completeness**
   All data must appear in some fragment.
2. **Reconstruction**
   We must be able to rebuild the original table using fragments.
3. **Disjointness**
   For horizontal: no overlapping rows
   For vertical: only primary key overlaps.

---

4. Allocation of Fragments

Allocation means **deciding which site will store which fragment**.

Fragment allocation strategies include:

1. Centralized Allocation

All fragments stored at a **single central site**.

- Simple
- Not fault tolerant

2. Replicated Allocation

Fragments are stored at **multiple sites**.

Advantages

- High availability
- Faster local queries

Disadvantages

- Update cost increases

3. Partitioned Allocation

Fragments are distributed across many sites,
but **without replication**.

Advantages

- Low update cost
- Balanced load

Disadvantages

- If one site fails, fragment becomes unavailable

4. Mixed Allocation

Combination of:

- Some fragments replicated
- Some stored at a single site

This is the **most commonly used** method in real systems.

Fragment Allocation Formula (Copy Format)

Minimization of Total Allocation Cost

```
Total Cost = Storage Cost + Query Cost + Update Cost + Communication Cost
```

Query Processing Cost

```
Query Cost = Local Processing Cost + Remote Access Cost
```

---

5. Transparencies in Distributed Database Design

Transparency means **hiding the complexity of distributed data** from the users.

Main types:

---

1. Fragmentation Transparency

User does not know data is broken into fragments.

Example:
When user queries CUSTOMER table → system automatically collects data from all fragments.

---

2. Replication Transparency

User does not know multiple copies of data exist.

System decides:

- Which replica to read
- Which replica to update

---

3. Location Transparency

Users do not need to know **where** data is stored.

They use a global schema:

```
SELECT * FROM EMPLOYEE;
```

System fetches data from the correct site.

---

4. Local Mapping Transparency

Local DBMS may be different (Oracle, MySQL, SQL Server),
but user sees a unified global view.

---

5. Distributed Query Transparency

User writes normal SQL, system handles:

- Fragment location
- Communication
- Optimization

---

6. Transaction Transparency

Distributed transactions behave like:

- A single, unified transaction
- Even if they run across multiple sites

Uses:

- 2-phase commit protocol

- Locking protocols

---

7. Performance Transparency

System automatically optimizes queries for:

- Minimum response time
- Minimum network cost

📘 Distributed Computing – Unit 3.0 (Detailed Notes)

Distributed Transaction and Concurrency Control

---

1. Basic Concepts of Transaction Management

A **transaction** is a sequence of operations performed as a single logical unit of work.

Example:

```
BEGIN
   Transfer ₹1000 from Account A
   Add ₹1000 to Account B
END
```

Properties of Transaction (ACID)

1. **Atomicity** – All operations succeed or none do.
2. **Consistency** – Database remains in a valid state.
3. **Isolation** – Parallel transactions do not interfere.
4. **Durability** – Results of committed transactions are permanent.

Why Transaction Management?

- To maintain correctness
- To avoid data loss
- To handle multiple users accessing same data
- To resolve conflicts and failures

---

2. Objective of Distributed Transaction Management

Distributed transactions are executed across **multiple sites**, **multiple databases**, and **multiple servers**.

Main objectives:

1. Ensure Global ACID Properties

All participating sites must follow ACID.

2. Coordination

Coordinate execution across sites using protocols.

3. Failure Recovery

Handle:

- Site failure
- Network failure
- Communication failure

4. Atomic Commitment

Ensure all sites either **commit** or **rollback**.

5. Global Consistency

Every site must end in a consistent state.

6. Minimize Communication Cost

Reduce the network messages during transaction execution.

---

3. Model for Transaction Management

A distributed transaction management model consists of:

---

1. Local Transaction Manager (LTM)

- Present at each site
- Manages local transactions
- Maintains local logs
- Handles local concurrency control

---

2. Global Transaction Manager (GTM)

- Coordinates all participating sites
- Controls global scheduling
- Ensures global ACID properties

---

3. Transaction Components

a. Coordinator

Initiates the transaction, communicates with all participants.

b. Participants / Cohorts

Execute transaction operations as instructed.

---

4. Commit Protocols

i. Two-Phase Commit Protocol (2PC)

Used to ensure atomicity.

Phase 1: Voting Phase

- Coordinator → "PREPARE to commit?"
- Participants → "YES" or "NO"

Phase 2: Decision Phase

- If all say YES → Coordinator sends COMMIT
- If any say NO → Coordinator sends ABORT

ii. Three-Phase Commit (3PC)

Improves over 2PC by eliminating blocking during failures.

---

4. Distributed Concurrency Control

Concurrency control ensures that **multiple transactions can run simultaneously without conflicts**.

Objectives:

1. Maintain Serializability

Execution should be equivalent to a serial order.

2. Avoid Conflicts

Avoid overlapping read/write operations that cause inconsistency.

3. Ensure Isolation

Each transaction works as if it is running alone.

4. Handle Distributed Environment Challenges

- Network delay
- Site failures
- Synchronization across sites

---

5. Concurrency Control Anomalies

If concurrency control is not used, the following problems occur:

---

1. Lost Update Problem

Two transactions update the same data, and one update is overwritten.

Example:
T1 reads X → T2 reads X → T2 updates X → T1 updates X
T2's update is lost.

---

2. Dirty Read Problem

A transaction reads uncommitted data from another transaction.

Example:
T1 updates X but does not commit → T2 reads that updated X → T1 rolls back → T2 has read wrong data.

---

3. Unrepeatable Read

A transaction reads the same data twice but gets different values.

---

4. Phantom Read

Query returns different sets of rows when executed twice due to updates/inserts.

---

6. Distributed Serializability

Distributed serializability ensures that transactions across multiple sites behave as if executed in **one global serial order**.

Types of Serializability

1. **Conflict Serializability**
   Transactions do not conflict in order.
2. **View Serializability**
   Based on read/write views of transactions.

Global Serialization Graph (SG)

Each site has a local serialization graph.
Global SG = Union of all local SGs.

If global SG is **acyclic**, then schedule is **serializable**.

---

7. Locking Based Algorithms

Locking ensures transactions access data items in an exclusive or shared way.

Types of Locks

1. Shared Lock (S-lock)

Allows a transaction to read data.
Multiple shared locks allowed.

2. Exclusive Lock (X-lock)

Allows a transaction to read & write data.
Only one X-lock allowed.

Two-Phase Locking (2PL) Protocol

Ensures conflict-serializability.

Phase 1: Growing Phase

Transaction obtains locks, but **cannot release** any lock.

Phase 2: Shrinking Phase

Transaction releases locks, but **cannot acquire** new ones.

Guarantee:

Schedules will be conflict-serializable but may cause **deadlocks**.

Distributed Locking Algorithms

1. Centralized Lock Manager

One site manages all locks.

Advantage: Simple
Disadvantage: Single point of failure

---

2. Primary Copy Locking

Each data item has a primary site for lock control.

---

3. Distributed (Fully Decentralized) Locking

Each site manages locks for its own data.

---

4. Majority Consensus Locking

Transaction needs locks from **majority of sites** before accessing data.

Deadlock Handling

Two Methods

1. **Deadlock Prevention**
   - Timestamp-based rules
   - Wound-Wait
   - Wait-Die
2. **Deadlock Detection & Recovery**
   - Wait-for graph
   - Victim selection
   - Transaction rollback

Formula (Copy Format)

1. Serializability Condition

```
A schedule is serializable if Global Serialization Graph (SG) is acyclic.
```

2. Locking Rule (2PL)

```
Growing Phase: Acquire Locks
Shrinking Phase: Release Locks
(No overlap allowed)
```

3. Commit Protocol Rule

```
If all participants vote YES → Commit
If any participant votes NO → Abort
```

📘 Distributed Computing – Unit 4.0 (Detailed Notes)

Distributed Deadlock and Recovery

---

1. Introduction to Deadlock

A **deadlock** is a situation where two or more transactions wait indefinitely for each other to release resources, and none of them can proceed.

Example (Simple Deadlock)

```
T1 holds Lock on X     → waits for Lock on Y
T2 holds Lock on Y     → waits for Lock on X
```

Both transactions wait forever → **DEADLOCK**.

Conditions for Deadlock (All must hold)

1. **Mutual Exclusion** – Resource held by only one transaction.
2. **Hold and Wait** – A transaction holds one resource and waits for another.
3. **No Preemption** – Resources cannot be forcibly taken away.
4. **Circular Wait** – Transactions form a waiting cycle.

---

2. Distributed Deadlock

In distributed systems:

- Resources are stored at **multiple sites**
- Locks are handled by **local lock managers**
- Global waits occur across sites
- Deadlock detection becomes more complex

Example:

```
T1 at Site A waits for lock at Site B
T2 at Site B waits for lock at Site A
```

---

3. Distributed Deadlock Handling Methods

Deadlock handling in distributed systems uses **three major techniques**:

A. Deadlock Prevention

Deadlock is prevented by eliminating one of the four conditions.

Methods:

---

1. Wound–Wait Scheme

- Based on **timestamps**
- If older transaction requests a lock held by a younger one → younger transaction is **wounded (rolled back)**
- If younger transaction requests lock from older one → younger **waits**

Advantage: No deadlock occurs

Disadvantage: Many rollbacks may occur

---

2. Wait–Die Scheme

- Also timestamp-based
- If older transaction requests lock held by younger → older **waits**
- If younger requests lock held by older → younger **dies (rolled back)**

Both ensure **no circular wait** → deadlock cannot occur.

B. Deadlock Avoidance

System checks for safe state **before granting a lock**.

Concept:

- Transaction provides lock requirements in advance
- System checks if granting a lock leads to a **potential deadlock**
- If unsafe → lock is not granted

Major Techniques:

1. **Resource Allocation Graph (RAG) Analysis**
2. **Safe State Check (similar to Banker's algorithm)**

Limitation:

- Requires advance knowledge of resource needs
- Difficult in distributed environment

C. Deadlock Detection

Deadlock detection is widely used in distributed systems.

System periodically checks for deadlocks using:

---

1. Wait-for Graph (WFG)

- Each transaction = node
- Edge T1 → T2 means T1 waiting for T2
- Cycle in WFG → **Deadlock**

Distributed WFG

Each site keeps a local WFG.
Global WFG = Combination of all local WFGs.

Deadlock Detection Algorithms

1. Path-Pushing Algorithm

Sites send dependency paths to central site → combines to detect cycles.

---

2. Edge-Chasing Algorithm

Uses special messages called **probes**:

```
(PROBE, Ti, Tj, Tk)
```

If probe returns to origin → deadlock exists.

---

3. Global State Detection

Snapshot of global lock state is evaluated to see if deadlock exists.

---

4. Deadlock Recovery

When a deadlock is detected, it must be resolved.

Recovery Techniques

1. Transaction Rollback (Victim Selection)

Pick a transaction to abort.

Criteria:

- Youngest transaction
- Minimum cost rollback
- Least progress made

---

2. Partial Rollback

Rollback only the last few operations instead of full rollback.

---

3. Resource Preemption

Take resource away from a transaction (rare in DBMS).

---

5. Two-Phase Commit Protocol (2PC)

2PC ensures **atomic commit** across all sites.

Phase 1: Prepare / Voting Phase

Coordinator → "PREPARE to commit?"
Participants → "YES" (vote-commit) or "NO" (vote-abort)

Phase 2: Commit / Abort Phase

If ALL vote YES → Coordinator sends COMMIT
If ANY votes NO → Coordinator sends ABORT

Properties

- Guarantees atomic commitment
- Can block if coordinator fails

---

6. Three-Phase Commit Protocol (3PC)

3PC improves 2PC by making the protocol **non-blocking**.

Phase 1: Can Commit?

Coordinator → asks participants if they can commit.

Phase 2: Pre-Commit

If all say YES → coordinator sends **PRE-COMMIT** message.
Participants prepare to commit but do not commit yet.

Phase 3: Do Commit

Coordinator sends **COMMIT** message.
Participants commit.

Advantages over 2PC

- Non-blocking
- Handles coordinator failure better
- Reduced waiting time

Comparison of 2PC vs 3PC

Feature	2PC	3PC
Phases	2	3
Blocking	Yes	No
Failure Handling	Weak	Strong

Feature	2PC	3PC
Performance	Faster	Slow but safe
Complexity	Simple	More complex

FORMULAS (Copy Format)

1. Deadlock Condition

```
Deadlock exists if Wait-For Graph contains a cycle.
```

2. Commit Protocol Rule

```
IF all participants vote YES → COMMIT
ELSE → ABORT
```

3. Circular Wait Condition

```
T1 → T2 → T3 → ... → Tn → T1   (Deadlock)
```

📘 Distributed Computing – Unit 5.0 (Detailed Notes)

Distributed Query Processing and Optimization

---

1. Concepts of Distributed Query Processing (DQP)

Distributed Query Processing refers to the techniques and algorithms used to **execute database queries in a distributed database system**, where:

- Data is stored at **multiple sites**
- Network communication is required
- Local DBMS systems may differ

Goal of DQP

To process a query efficiently and return correct results **with minimum cost** (communication + computation).

Major Activities

1. **Decompose the global query**
2. **Optimize the query plan**
3. **Locate data fragments**
4. **Execute operations at appropriate sites**
5. **Assemble final result**

---

2. Objectives of Distributed Query Processing

1. Minimize Communication Cost

Communication is expensive in distributed systems.
Reduce:

- Data transfer
- Message passing
- Network traffic

2. Improve Performance

Perform operations closer to data by using:

- Local processing
- Parallelism

3. Reduce Response Time

Execute operations in parallel at multiple sites.

4. Ensure Correctness

Distributed processing should produce the **same result** as centralized execution.

5. Handle Heterogeneous Systems

Different sites may use:

- Different DBMS
- Different data models
- Different hardware

---

3. Phases of Distributed Query Processing

Distributed query processing occurs in **four major phases**:

Phase 1: Query Decomposition

The global query is converted into algebraic expressions.

Steps:

- Parse SQL query
- Apply **query rewriting rules**
- Generate algebraic representation

Example:

```
SELECT * FROM EMPLOYEE WHERE CITY = 'PATNA';
```

→ Selection + Projection + Scan operations

Phase 2: Data Localization (Fragment Processing)

Convert the global query into **fragment queries** based on how data is distributed.

Tasks:

- Identify **relevant fragments**
- Apply **fragmentation rules**
- Apply **localization rules**

Example:

If EMPLOYEE table is horizontally fragmented:

- EMP1 (North Region)
- EMP2 (South Region)

Then query becomes:

```
SELECT * FROM EMP1 WHERE CITY='PATNA'
UNION
SELECT * FROM EMP2 WHERE CITY='PATNA'
```

Phase 3: Global Optimization

Create the **best possible execution plan**.

Optimization techniques:

- Minimize communication
- Select optimal join order
- Reduce intermediate results
- Use semi-join if beneficial

Phase 4: Local Optimization and Execution

Each local site optimizes its own sub-query.

Local optimization involves:

- Choosing best access path
- Using indexes
- Using sorting algorithms
- Using efficient join strategies

Results are sent back to the coordinator for final assembly.

---

4. Join Strategies in Fragment Relations

When relations are fragmented across sites, joins become challenging.

Below are the common distributed join strategies:

A. Semi-Join Strategy

Reduces the amount of data transferred.

Steps:

1. Send projection of join attribute from Site A to Site B
2. Site B filters its relation
3. Send only matching tuples back to Site A
4. Final join at Site A

Benefit:

Minimum communication cost.

B. Full Fragment Transfer

Transfer one full fragment to the site of the other fragment.

Example:

Send R1 to site of S1 and perform join.

Benefit:

Simple and fast for small fragments.

C. Hybrid Join Strategy

Combination of semi-join + full transfer.

Uses:**

- When fragments are too large
- When selective filtering is possible

D. Bloom Join

Uses **Bloom filters** to reduce communication cost.

Steps:

1. Site A sends Bloom filter of join attributes to Site B
2. Site B filters tuples
3. Sends only possible matching tuples back

Benefit:

Less data transferred than semi-join.

E. Local Fragment Join

If required fragments are available locally at one site, perform join locally without communication.

F. Repartition Join

Re-partition both relations by join key and send partitions to appropriate sites.

Used when:

Data is evenly distributed.

G. Broadcast Join

Broadcast smaller relation to all sites containing the larger relation.

Used when:

One relation is very small.

Table: Comparison of Join Strategies

Join Strategy	Communication Cost	Best Use Case
Semi-Join	Low	Selective joins, large tables

Join Strategy	Communication Cost	Best Use Case
Full Transfer	High	Small tables
Bloom Join	Medium-Low	Better than semi-join for large data
Local Join	Zero	All fragments at one site
Repartition Join	Medium	Distributed parallel join
Broadcast Join	Low/Medium	One relation very small

---

5. Global Query Optimization

Global optimization finds the **most efficient execution plan** for the entire distributed query.

Goals:

- Reduce communication cost
- Reduce total response time
- Reduce resource usage

Major Techniques Used

---

1. Rule-Based Optimization (RBO)

Uses transformation rules such as:

- Selection pushdown
- Projection pushdown
- Join reordering
- Join associativity
- Fragment elimination

Example rule:

```
σ(condition)(R ⋈ S) = (σ(condition)(R)) ⋈ S
```

2. Cost-Based Optimization (CBO)

Evaluates cost formulas and selects the lowest-cost plan.

Cost Formula

```
Total Query Cost = Local Processing Cost + Communication Cost + Disk Access
Cost
```

---

3. Heuristic Optimization

Uses general guidelines like:

- Perform selection early
- Perform projection early
- Use semi-joins
- Minimize data movement

---

4. Dynamic Optimization

Optimization decisions change at run-time based on:

- System load
- Network performance
- Available resources

Formulas

1. Cost of Distributed Query

```
Total Cost = Local Processing Cost + Communication Cost + Remote Access Cost
```

2. Semi-Join Formula

```
R ⋈ S = R ⋈ (π JoinAttribute(S))
```

3. Serialization of Distributed Query

```
Global Optimization = Σ (Local Optimization + Communication Minimization)
```

Unit-6.0: Heterogeneous Database (8 Hrs)

1. Architecture of Heterogeneous Database

A **Heterogeneous Database System (HDBS)** is a distributed database system where different sites use **different DBMSs, data models, query languages, schemas, and hardware platforms**.

Key Characteristics

- Contains **multiple autonomous databases**.
- Databases may differ in:
  - Data Model (Relational, Object-oriented, Hierarchical etc.)
  - Query Language (SQL, OQL, Native languages)
  - Hardware/OS
  - Transaction Protocol
- Communication and coordination are done through a **middleware or mediator**.

Basic Architecture

A heterogeneous system has **three main layers**:

(A) Local Layer

- Individual databases operate **independently**.
- Each DBMS maintains its own **local schema** and performs local queries.

(B) Global Conceptual Layer / Federated Layer

- Integrates all local schemas into a **Global Conceptual Schema (GCS)**.
- Provides a **uniform view** of all distributed databases.
- Acts as a communication bridge.

(C) External/Application Layer

- Represents how applications and users see the global database.
- User queries go to the GCS → translated → executed at local sites.

Types of Heterogeneous Database Architectures

1. **Federated / Multidatabase System (MDBS)**
   - Local systems remain autonomous.

    - Only partial sharing of data happens.
2. **Client/Server Architecture**
    - Client issues global queries.
    - Server (mediator) decomposes queries and communicates with local DBMS.
3. **Mediator–Wrapper Architecture**
    - Wrapper: Converts global queries into local DBMS-compatible format.
    - Mediator: Integrates results from all wrappers.

---

2. Database Integration

Database integration means **combining multiple heterogeneous databases** into one unified system.

Integration is needed because:

- Organizations use multiple DBMS types.
- Need consistent global access to distributed data.
- Need a unified interface for decision making.

Two Main Tasks of Database Integration

(A) Schema Translation

Converts schemas of different databases into a **common data model (CDM)**.

Steps in Schema Translation

1. Identify the **source DBMS type** (Relational, XML, Object DB etc.).
2. Map source model constructs to **CDM constructs**.
3. Convert:
    - Data types
    - Keys
    - Relationships
    - Constraints
4. Resolve semantic differences (meaning of data).

Example

- Object-oriented class → Relational Table
- XML structure → Relational or Object model

(B) Schema Integration

Combines multiple translated schemas into a **single global schema**.

Tasks in Schema Integration

1. **Schema Matching**
   - Identify correspondences between elements of different schemas.
   - Example:
     - *Customer_ID ↔ CustID*
2. **Schema Merging**
   - Combine matched elements into a single definition.
3. **Conflict Resolution**
   Three types of conflicts must be resolved:
   - **Naming Conflicts**
     - Same name, different meaning (Homonyms)
     - Different name, same meaning (Synonyms)
   - **Structural Conflicts**
     - Different representations of the same concept.
       Example:
       - One DB stores "Address" as a single field; another uses multiple fields.
   - **Data Type Conflicts**
     - Example: INT in one DB and VARCHAR in another.
4. Create **Global Conceptual Schema (GCS)**
   - Final unified schema seen by the user.

---

3. Query Processing Issues in Heterogeneous Databases

Query processing in HDBS is **complex** because of differences in:

- Query languages
- Data models
- Local execution strategies
- Communication protocols

Major Issues

(A) Query Decomposition

- A global query must be **broken into sub-queries** for each local database.
- Decomposition must consider:
  - Local schema
  - Data model differences
  - Local capabilities

(B) Query Translation

- Each sub-query must be translated to the **local DBMS query language**.
- Example:
  - SQL query converted into OQL or a native API call.

(C) Metadata Conflicts

- Local sites may use different metadata formats.
- Schema mismatches cause problems during query interpretation.

(D) Data Model Differences

- Relational results must be converted to object-oriented structures or vice versa.

(E) Execution Optimization

- Global optimizer must decide:
  - Which site to query first
  - Where joins should occur
  - How to minimize data transfer cost
- Must consider:
  - Network latency
  - Site load
  - Local query optimizers' capabilities

(F) Heterogeneous Transaction Management

- Distributed transactions involve multiple DBMSs.
- Two-Phase Commit (2PC) may not work uniformly.
- Recovery mechanisms differ across systems.

(G) Data Integrity and Consistency

- Different DBMSs may handle constraints differently.
- Ensuring global consistency is a major challenge.

(H) Security Issues

- Each database may enforce different security policies.
- The global system needs **uniform access rules**.

Summary (Copy Format)

```
Unit-6.0: Heterogeneous Database (8 Hrs)

1. Architecture of Heterogeneous Database
- Involves multiple autonomous databases using different DBMSs, data models,
and languages.
- Layers: Local Layer, Global Conceptual Layer, External Layer.
- Architectures: Federated System, Client/Server, Mediator-Wrapper.

2. Database Integration
A. Schema Translation:
- Converts local schemas to a common data model.
- Handles data type, structure, and semantic conversions.

B. Schema Integration:
- Combines multiple schemas into a Global Conceptual Schema.
- Resolves naming, structural, and data type conflicts.

3. Query Processing Issues in Heterogeneous Database
- Query decomposition and translation.
- Metadata and data model differences.
- Global optimization and execution issues.
- Transaction management challenges.
- Ensuring consistency and security.
```