Introduction to Database Management System

Database System Environment

A Database System Environment refers to the complete setup of hardware, software, data, people, and procedures that work together to create, manage, and use a database effectively.
It provides the platform for storing, processing, and retrieving data in a controlled and efficient manner.

Components:

1. Hardware – Servers, storage devices, client machines, networking devices.
2. Software – DBMS (MySQL, Oracle, PostgreSQL), operating system, application software.
3. Data – The stored facts (structured, semi-structured, unstructured).
4. People – DBAs, developers, end users.
5. Procedures – Rules and instructions for designing, using, and maintaining the database.
6. Networking Components – Required for multi-user and distributed environments.

Functions of a Database System Environment

- Provides an interface between users and data.
- Ensures efficient storage, retrieval, and manipulation of data.
- Maintains data integrity, security, and availability.
- Supports multi-user concurrent access without conflicts.

File-Oriented Approach

The file-oriented approach is the traditional method of storing and managing data before Database Management Systems (DBMS) became common.
In this approach, data is kept in separate files, and application programs are written to handle the data in each file directly

Characteristics

1. Separate files for each application – e.g., one file for student attendance, another for exam results.
2. Application-dependent – Each program knows how to read/write its own file format.
3. Manual linking – Relationships between data in different files must be managed by the programmer.
4. No centralized control – Each file is independent.

Example-A school storing data in three separate files:

- attendance.txt → Attendance records
- marks.txt → Examination scores
- fees.txt → Fee payment details

If a student changes their name, it must be updated manually in all files.

Disadvantages

1. Data Redundancy – Same data repeated in multiple files.
2. Data Inconsistency – Different versions of the same data may exist in different files.
3. Difficulty in Access – Requires separate programs to fetch data.
4. Poor Security – No centralized authentication or permission control.
5. No Concurrent Access – Multiple users cannot safely update files at the same time.
6. No Data Independence – Any change in file structure requires rewriting programs.

Database Approach

The Database Approach is a method of storing and managing data where all data is kept in a centralized database managed by a Database Management System (DBMS), instead of scattered in separate files. Multiple applications and users share the same data through the DBMS, which controls access, ensures security, and maintains consistency.

Key Features

1. Centralized Data Storage – All data is stored in one database rather than separate files.
2. Data Integration – Data from different departments/applications is combined in a single repository.
3. Data Independence – Changes to the database structure do not require changes in application programs.
4. Reduced Redundancy – Duplicate copies of the same data are minimized.
5. Multi-user Access – Multiple users can access and update data at the same time.
6. Security and Access Control – Permissions are centrally managed.

Advantages

- Consistency – Single source of truth, fewer data conflicts.
- Easier Maintenance – One place to update and back up.
- Improved Data Sharing – Multiple applications can use the same dataset.
- Enhanced Security – Controlled by DBMS authentication and authorization.
- Better Decision-Making – Reliable and up-to-date information.

Disadvantages

- Higher Cost – DBMS software, hardware, and trained staff are expensive.
- Complexity – Requires skilled administrators and proper planning.
- Central Point of Failure – If the central database fails, all dependent applications may stop working.

Example

In a university:

- Attendance, examination, and fee details are all stored in one database.
- The administration, faculty, and accounts department all access this same database via the DBMS.

Users of DBMS

Users of a Database Management System (DBMS) are individuals or groups who interact with the database to perform tasks such as creating, updating, retrieving, or managing data.
They can be classified based on their interaction level and role in the database environment.

Types of DBMS Users

1. Database Administrators (DBAs)

- Responsible for managing the entire database system.
- Roles & Responsibilities:
    - Install and configure DBMS software.
    - Manage user accounts and permissions.
    - Ensure data security, backup, and recovery.
    - Monitor and tune database performance.
- Example: The IT team member maintaining a hospital's patient database.

2. Database Designers

- Responsible for designing the database schema and structure.
- Roles & Responsibilities:
    - Identify what data needs to be stored.
    - Create relationships between data entities.
    - Ensure the database meets the needs of the organization.
- Example: Designing a relational database for an online shopping platform.

3. Application Programmers / Developers

- Write software programs that interact with the database.
- Roles & Responsibilities:
    - Develop and maintain applications using database APIs and SQL.
    - Implement business logic.
- Example: Building a mobile app that retrieves and stores product details in a database.

4. End Users

These are the people who actually use the database for their work.
They are divided into:

1. **Naïve Users**
    - Use pre-designed applications without knowing database details.
    - Example: ATM users withdrawing cash.
2. **Casual Users**
    - Use the database occasionally and may write ad-hoc queries.
    - Example: A manager checking sales reports.
3. **Sophisticated Users**
    - Use advanced query tools, analytics, or programming interfaces.
    - Example: Data analysts running complex SQL queries.

Intended Use of DBMS:

A DBMS is used for:
- Efficient data storage, retrieval, and manipulation.
- Supporting multi-user access.
- Providing security and privacy.
- Ensuring backup and recovery.
- Helping in application development via query languages and APIs.

Benefits of Database Approach

1. Data Redundancy Control

- Eliminates duplicate copies of the same data.
- Example: A student's information stored once, not separately in library and exam records.

2. Data Consistency

- Since data is stored centrally, any update is reflected everywhere.
- Example: Updating an address in the student database updates it for all departments.

3. Improved Data Sharing

- Multiple users and applications can access the same database simultaneously.
- Example: Sales, inventory, and finance departments accessing the same stock database.

4. Better Data Security

- Access rights and permissions can be assigned to specific users.
- Example: Only managers can view salary details.

5. Enhanced Data Integrity

- Enforces rules (constraints) to ensure accuracy and reliability.
- Example: Preventing negative stock quantities in inventory.

6. Data Independence

- Applications are independent of how data is stored or structured physically.
- Example: Changing the database storage format without modifying programs.

7. Backup and Recovery

- Automatic tools for data backup and restoration after failures.
- Example: Restoring bank transaction data after a power outage.

8. Support for Concurrent Access

- Many users can work with the database at the same time without conflicts.
- Example: Multiple ATMs updating the same account balance in real time.

9. Better Decision-Making

- Centralized and accurate data supports analytics and business intelligence.
- Example: Generating sales trend reports to decide marketing strategies

Concepts of Client-Server Architecture and Distributed System

A. Client-Server Architecture:

Definition – A client-server architecture is a network model in which the client (front-end) sends requests for services, and the server (back-end) provides the requested services.
In DBMS, clients send SQL queries to the server, and the server processes these queries and returns results.

Components:

1. **Client**:
   - Runs the user interface.
   - Sends queries to the server.
   - Examples: Web browser, desktop application.
2. **Server**:
   - Hosts the DBMS.
   - Processes requests, performs operations, and sends responses.
3. **Network**:
   - Medium that connects clients and servers.

Advantages:

- Centralized control of data and security.
- Easier maintenance – updates only on the server.
- Better data integrity – single source of truth.

Example:
Banking system – branch computers (clients) connect to a central database server.

B. Distributed System:

Definition – A distributed database system is one where data is stored across multiple physical locations, but appears to users as a single logical database.

Advantages:

- Faster local access – Users access data from the nearest location.
- Higher reliability – Failure at one site does not stop the whole system.
- Load balancing – Workload is shared among sites.

Challenges:

- Data synchronization – Keeping data updated across sites.
- Consistency maintenance – Avoiding conflicts between copies.

- Higher complexity – More complicated than a centralized DBMS.

Example:
An e-commerce company stores customer data in Asia, orders in Europe, and payments in the US, but users access it as one unified system.

Types:
1) **Homogeneous Distributed Database System** - is a type of distributed database where all sites use the same DBMS software, data models, and operating systems.
This uniformity makes communication, query processing, and data management easier across all sites.

Key Features

1. Same DBMS software at all locations.
2. Same data model (e.g., relational).
3. Same operating system and hardware type (often, but not always required).
4. Easier integration and management compared to heterogeneous systems.
5. Sites appear as if they are part of one central database.

Advantages

- Simpler query execution – Since all sites understand the same query language (e.g., SQL).
- Easier data replication – Same structure and format at all sites.
- Uniform security policies – Same DBMS features at each site.
- Lower maintenance complexity – Single type of software to update and manage.

Disadvantages

- Less flexible if different locations have unique requirements.
- Dependent on a single DBMS vendor and technology stack.

Example

A multinational company with offices in different countries, all running Oracle DBMS for their branch databases.
Even though data is stored at different sites, the same DBMS software and schema are used everywhere.

2) **Heterogeneous Distributed Database System**- is a type of distributed database where different sites may use different DBMS software, data models, operating systems, or hardware platforms.
Despite the differences, these systems are integrated so that users experience them as a single unified database.

Key Features

1. Different DBMS software at various sites (e.g., Oracle at one site, MySQL at another).
2. Different data models possible (e.g., relational at one site, object-oriented at another).
3. Different operating systems (Windows, Linux, etc.).
4. Requires middleware or gateway software to translate queries and data between systems.
5. More flexible but also more complex than homogeneous systems.

Advantages

- Flexibility – Each site can choose the DBMS that best suits its local needs.
- Integration of legacy systems – Older databases can work with newer ones.
- Vendor independence – No dependency on a single DBMS vendor.

Disadvantages

- Complex query processing – Queries must be translated across different DBMS.
- Data conversion overhead – Differences in data formats and models require transformations.
- Higher maintenance complexity – Updates and security must be managed for multiple systems.
- Possible performance issues – Due to translation and compatibility layers.

Example

A retail company has:

- Oracle DB for finance (Linux server)
- MySQL DB for sales (Windows server)
- MongoDB for customer analytics (Cloud)

All these databases are connected so managers can run combined reports.

Database system concept and Application

Data Models

- **Definition**:
  A **data model** is a set of concepts, rules, and structures used to describe and represent the data, relationships, and constraints in a database.
- Provides a **conceptual framework** for database design and implementation.
- Acts as a **bridge** between the real world and the database system.

Types of Data Models

1. **High-Level (Conceptual) Data Models**
   - Provide concepts close to the way users perceive data.
   - Example: **Entity-Relationship (ER) Model** → Entities, Attributes, Relationships.
2. **Representational (Implementation) Data Models**
   - Provide concepts that may be understood by users but are closer to computer representation.
   - Example: **Relational Model** → Tables (relations), Rows (tuples), Columns (attributes).
3. **Low-Level (Physical) Data Models**
   - Describe how data is stored in the computer.
   - Example: Record formats, Indexes, Pointers, File structures

Schemas

- **Definition**:
  A **schema** is the overall description of the database structure, written in a specific data model.
- It acts like a **blueprint** or **design** of the database.
- **Characteristics**:
  - Relatively **stable** (changes rarely).
  - Defines entities, attributes, relationships, and constraints.

Types of Schemas

1. **Physical Schema** – How data is physically stored (files, indexes, partitions).
2. **Logical Schema** – The logical structure of the database (tables, relationships, constraints).
3. **View Schema (External Schema)** – User-specific views of the data (subsets of the database).

Instances

- **Definition**:
  A **database instance** is the **snapshot of data** in the database at a given point in time.
- It is the **actual data** stored in the database, not the structure.
- **Characteristics**:
  - Changes frequently as data is inserted, deleted, or updated.
  - Multiple instances exist during the lifetime of a database, but schema usually remains fixed.

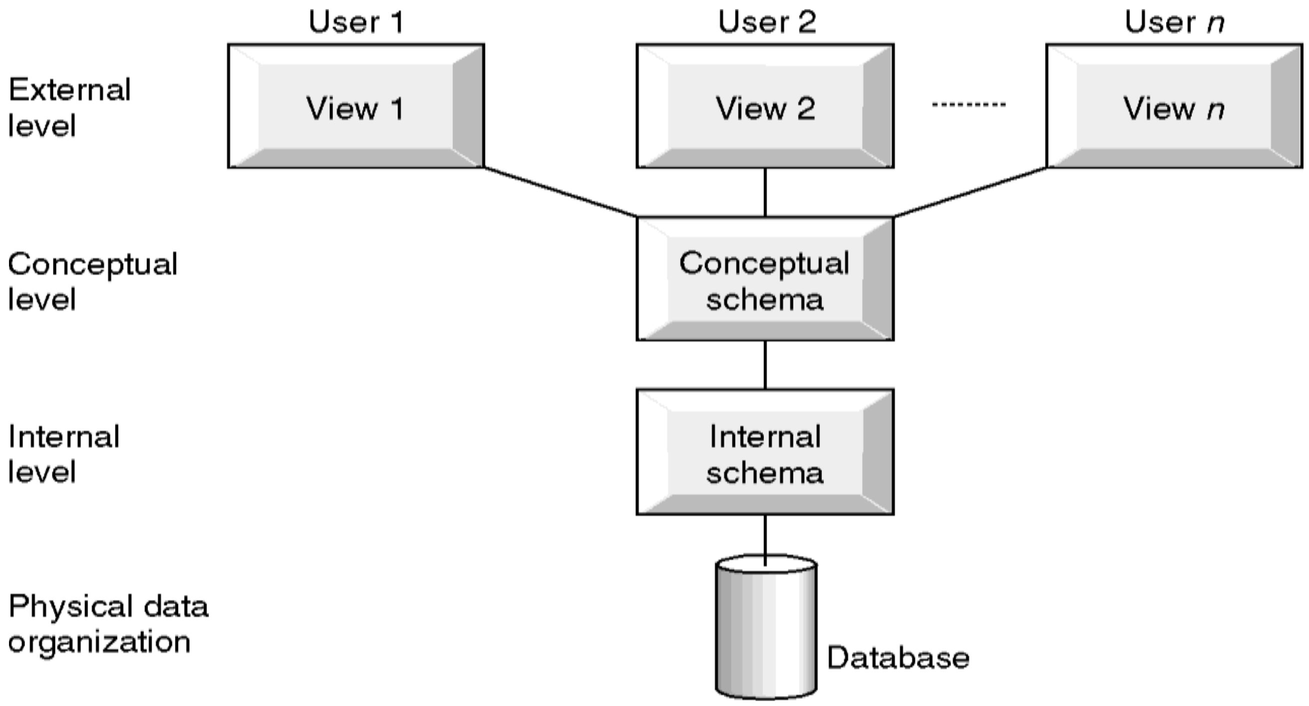
DBMS Architecture

DBMS architecture refers to the **structure and organization** of a Database Management System. It defines how users interact with the database and how data is stored, accessed, and managed.

Levels of DBMS Architecture (Three-Level Architecture)

Proposed by **ANSI/SPARC (American National Standards Institute / Standards Planning and Requirements Committee)**.

1. **External Level (View Level)**
   - Closest to the **end users**.
   - Defines how individual users see the data (subsets of the database).
   - Users interact with the database via queries.
   - Example: A student sees only their marks, but the faculty sees marks of all students.
2. **Conceptual Level (Logical Level)**
   - Describes the **structure of the entire database**.
   - Defines entities, relationships, constraints, and data types.
   - Independent of physical storage details.
   - Example: Schema → STUDENT(Roll_No, Name, Dept, Marks).
3. **Internal Level (Physical Level)**
   - Lowest level, closest to the **physical storage**.
   - Describes **how data is stored** (files, indexes, record formats).
   - Example: Student table stored as a B-tree index on disk.



DBMS Components

- **DDL Compiler** → Translates Data Definition Language into schema definitions.
- **DML Compiler** → Translates user queries into low-level instructions.
- **Query Optimizer** → Chooses the most efficient query execution plan.
- **Transaction Manager** → Ensures ACID properties.
- **Storage Manager** → Manages data storage on disk.

Data Independence

Data independence is the ability to **change the schema at one level** of the database **without affecting the schema at the next higher level**.

Types of Data Independence

1. **Logical Data Independence**
    - Ability to change the **conceptual schema** (tables, attributes, relationships) without affecting **external schemas** (user views).
    - Example: Adding a new attribute "Email" to STUDENT schema should not affect the faculty's view of students' marks.
2. **Physical Data Independence**
    - Ability to change the **internal schema** (how data is physically stored) without affecting the **conceptual schema**.
    - Example: Moving data from HDD to SSD, or changing file organization, without changing the logical schema STUDENT(Roll_No, Name, Dept, Marks).

Importance of Data Independence

- Provides **flexibility** in database design.
- Makes the system **easier to maintain and upgrade**.
- Ensures **user programs are insulated** from storage details.
- Enhances **data security and abstraction**.

Database Languages

Database languages are special-purpose languages used to **define, manipulate, and control** data in a database.

1.Data Definition Language (DDL)

- Used to **define and modify database schema** (structure of database objects).
- Commands:
    - `CREATE` – Create database objects (tables, views).
    - `ALTER` – Modify existing objects.
    - `DROP` – Delete objects.

Example

```
CREATE TABLE STUDENT (
      Roll_No INT PRIMARY KEY,
      Name VARCHAR(50),
      Dept VARCHAR(20),
      Marks INT
);
```

2.Data Manipulation Language (DML)

- Used to **retrieve and modify data**.
- Types:
    - **Procedural DML** – User specifies *what* data is needed and *how* to get it.
    - **Non-Procedural DML (Declarative)** – User specifies *what* data is needed, DBMS figures out *how*. (e.g., SQL).

- Commands:
  - `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
- Example:

```
SELECT Name, Marks FROM STUDENT WHERE Dept = 'CSE';
```

3.Data Control Language (DCL)

- Used to **control access and permissions**.
- Commands:
  - `GRANT` – Give access rights.
  - `REVOKE` – Remove access rights.
- Example:

```
GRANT SELECT ON STUDENT TO Faculty;
```

4.Transaction Control Language (TCL)

- Used to manage **transactions** in a database (ensuring ACID properties).
- Commands:
  - `COMMIT` – Save changes permanently.
  - `ROLLBACK` – Undo changes.
  - `SAVEPOINT` – Mark a point within a transaction.
- Example:

```
BEGIN;
UPDATE STUDENT SET Marks = Marks + 5 WHERE Dept = 'CSE';
COMMIT;
```

5.Query Language

- Subset of DML, focused on retrieving information.
- SQL (`SELECT`) is the most widely used **query language**.
- Example:

```
SELECT Dept, AVG(Marks) FROM STUDENT GROUP BY Dept;
```

Database Interfaces

Database interfaces are the **methods through which users or applications interact with the DBMS**.

1. Menu-Based Interfaces

- Users interact through menus and options.
- Simple for non-technical users.
- Example: ATM screens, hotel booking systems.

2.Form-Based Interfaces

- Users enter data into **forms** with fields (textboxes, dropdowns).
- Example: Online admission form, shopping site checkout form.

3. Graphical User Interfaces (GUIs)

- Provides graphical elements like windows, icons, and buttons for interaction.
- Example: MS Access GUI, Oracle SQL Developer.

4. Natural Language Interfaces

- Users type queries in natural (English-like) language.
- DBMS interprets them and retrieves results.
- Example: "Show me all students in CSE department."

5. Command-Line Interfaces (CLI)

- Users type **SQL commands** directly.
- Powerful but requires technical knowledge.
- Example: MySQL CLI, PostgreSQL `psql`.

6. Application Program Interfaces (APIs)

- Allow applications to interact with the DBMS programmatically.
- Examples:
  - **ODBC (Open Database Connectivity)**.
  - **JDBC (Java Database Connectivity)**.
  - Python connectors (`mysql-connector`, `sqlite3`).

RELATIONAL DATA MODEL

The **Relational Data Model** was proposed by **Dr. E. F. Codd**.
It represents a database in the form of **tables (relations)** consisting of rows and columns. It is the **most widely used data model** in database systems.

Domains, Attributes, Tuples, and Relations

1. Domain

A **Domain** is a **set of all possible valid values** that an attribute can take.

✅It defines the **type of data** for an attribute.

2. Attribute

An **Attribute** represents a **column of a table**.
It describes a **property or characteristic of an entity**.

3. Tuple

A **Tuple** represents a **single row (record) in a table**.
It is a collection of attribute values.

4. Relation

A **Relation** is a **table consisting of rows (tuples) and columns (attributes)**.

Degree and Cardinality of a Relation

- **Degree** → Number of attributes (columns)
- **Cardinality** → Number of tuples (rows)

Key Constraint

A **key** is an attribute or set of attributes that **uniquely identifies a tuple** in a relation.

Types of Keys:

- Super Key
- Candidate Key
- Primary Key
- Alternate Key
- Foreign Key

Entity Integrity Constraint

This constraint states that:

Primary key value of a relation cannot be NULL.

It ensures every tuple can be uniquely identified.

Referential Integrity Constraint

This constraint ensures **consistency between related tables**.

Rule:

A **foreign key value must either be NULL or match a primary key value** in another table.

Relational Database Schema

A **Relational Database Schema** is the **logical structure of the database**.
It defines:

- Relation names
- Attributes
- Data types
- Constraints

It is a **blueprint of the database** before actual data is stored.

E–R DIAGRAM (ENTITY–RELATIONSHIP MODEL)

The **E–R Model** is a **conceptual data model** used to represent the structure of a database in a **graphical form**. It was proposed by **Peter Chen**.
An **E–R Diagram (ERD)** shows:

- Entities
- Attributes
- Relationships
  between real-world objects.

It is mainly used in the **database design phase**.

Defining Relations and Entity Set

1. Entity

An **entity** is a **real-world object** that has an independent existence and can be uniquely identified.

Examples:

- Student
- Teacher
- Employee
- Book
- Department

2. Entity Set

An **Entity Set** is a **collection of similar entities** of the same type.

Examples:

- All students in a college → STUDENT entity set
- All employees in an office → EMPLOYEE entity set

3. Attributes

An **attribute** is a **property or characteristic of an entity**.

Examples:

- Student_ID, Name, Age → Attributes of STUDENT
- Emp_ID, Salary → Attributes of EMPLOYEE

Types of Attributes:

1. **Simple Attribute** – Cannot be divided
   Example: Age
2. **Composite Attribute** – Can be divided into sub-parts
   Example: Address (House No, City, PIN)
3. **Single-valued Attribute** – Only one value
   Example: Roll Number

4. **Multi-valued Attribute** – More than one value
   Example: Phone numbers
5. **Derived Attribute** – Can be calculated
   Example: Age derived from Date of Birth
6. **Key Attribute** – Uniquely identifies an entity
   Example: Student_ID

4. Relationship

A **relationship** is an **association between two or more entities**.

Examples:

- A student *enrolls in* a course
- A teacher *teaches* a subject

5. Relationship Set

A **Relationship Set** is a **collection of similar relationships**.

Example:

- TEACHES → relationship set between TEACHER and SUBJECT

6. Degree of a Relationship

The **degree** indicates the number of entities involved:

- **Unary relationship** – One entity
- **Binary relationship** – Two entities (most common)
- **Ternary relationship** – Three entities

7. Cardinality Ratio

It specifies **how many entities are related to one another**.

Types:

- **1 : 1** (One-to-One)
- **1 : N** (One-to-Many)
- **M : N** (Many-to-Many)

8. Participation Constraint

It shows whether participation is:

- **Total Participation** – Every entity must participate
- **Partial Participation** – Some entities may not participate

9. Relation (in DBMS sense)

A **relation** is a **table representation of an entity or relationship** in the relational model.

Example:
STUDENT (Student_ID, Name, Age, Address)

E–R Model Concept with Examples

Definition of E–R Model

The **Entity–Relationship Model** is a **high-level data model** used to describe data and its relationships using:

- Entities
- Attributes
- Relationships
- Constraints

It is mainly used for **conceptual database design**.

Advantages of E–R Model

- Easy to understand
- Simple graphical representation
- Makes database design systematic
- Reduces design errors
- Acts as a blueprint for database creation

SQL (Structured Query Language)

SQL (Structured Query Language) is a standard language used to:

- Create and manage databases
- Store and retrieve data
- Control access to data

SQL is used with relational database systems like:

- MySQL
- Oracle
- SQL Server
- PostgreSQL

Data Definition in SQL (DDL)

Data Definition Language (DDL) is used to **define the structure of database objects** like tables, views, and indexes.

Common DDL Commands:

1. CREATE

Used to create database object

Create Database:

```
CREATE DATABASE College;
```

Create Table:

```
CREATE TABLE Student (
   Student_ID INT PRIMARY KEY,
   Name VARCHAR(50),
   Age INT,
   Address VARCHAR(100)
);
```

ALTER

Used to modify the structure of an existing table.

Add a new column:

```
ALTER TABLE Student ADD Email VARCHAR(50);
```

Modify a column:

```
ALTER TABLE Student MODIFY Age INT;
```

DROP

Used to delete a table or database completely.

```
DROP TABLE Student;
```

Data is permanently deleted.

TRUNCATE

Deletes all records from a table but keeps the structure.

```
TRUNCATE TABLE Student;
```

Views in SQL

What is a View?

A **View** is a **virtual table** based on the result of an SQL query.
It does **not store data physically**.

Advantages of Views

- Improves **security**

- Simplifies **complex queries**
- Provides **data abstraction**
- Hides sensitive columns
- Improves user access control

Types of Views

1. **Simple Views** – Based on a single table
2. **Complex Views** – Based on multiple tables

NORMALIZATION (DBMS)

Normalization is the process of **organizing data in a database** to:

- Reduce **data redundancy**
- Avoid **data anomalies**
- Improve **data integrity**
- Ensure **logical data storage**

It divides large tables into **smaller, well-structured tables** using normal forms.

Functional Dependencies (FD)

Definition:

A **Functional Dependency (FD)** is a relationship between two sets of attributes in a relation.

It is written as:

X → Y

This means:

Attribute Y is functionally dependent on attribute X
(X determines Y)

Example:

STUDENT (Roll_No, Name, Dept)

- Roll_No → Name
- Roll_No → Dept

Here, **Roll_No is the determinant**.

Types of Functional Dependencies:

1. **Trivial FD**
   If Y is a subset of X
   Example:
   (A, B) → A

2. **Non-Trivial FD**
   Y is not a subset of X
   Example:
   Roll_No → Name
3. **Fully Functional Dependency**
   Y depends on **entire X**, not on part of it
   Example:
   (Roll_No, Subject) → Marks
4. **Partial Dependency**
   Y depends on **part of a composite key**
   Example:
   (Roll_No, Subject) → Student_Name
   (Here Student_Name depends only on Roll_No)
5. **Transitive Dependency**
   A → B and B → C then A → C
   Example:
   Roll_No → Dept_ID
   Dept_ID → Dept_Name
   So Roll_No → Dept_Name (transitive)

Normal Forms Based on Primary Keys

1. First Normal Form (1NF)

A table is in **1NF** if:

- All attributes contain **atomic (single) values**
- No **multi-valued attributes**
- Each record can be **uniquely identified**

Example (Not in 1NF):

Roll	Name	Phone
1	Ravi	9876, 9123

✅After converting to 1NF:

Roll	Name	Phone
1	Ravi	9876
1	Ravi	9123

2. Second Normal Form (2NF) – Definition:

A relation is in **2NF** if:

- It is in **1NF**
- It has **no partial dependencies**

Used mainly when **composite primary keys** exist.

3.Third Normal Form (3NF) – Definition:

A relation is in **3NF** if:

- It is in **2NF**
- It has **no transitive dependencies**
- Non-prime attributes depend **only on the primary key**

Example (Not in 3NF):

EMPLOYEE (Emp_ID, Emp_Name, Dept_ID, Dept_Name)

FD:

- Emp_ID → Emp_Name
- Emp_ID → Dept_ID
- Dept_ID → Dept_Name ❌Transitive dependency

✅Convert to 3NF:

EMPLOYEE (Emp_ID, Emp_Name, Dept_ID)
DEPARTMENT (Dept_ID, Dept_Name)

4.Boyce–Codd Normal Form (BCNF)

Definition:

A relation is in **BCNF** if:

- It is in **3NF**
- For every functional dependency **X → Y**,
  X must be a super key

BCNF is a **stronger version of 3NF**.

Example (3NF but not in BCNF):

STUDENT_SUBJECT (Student, Subject, Teacher)

FD:

- (Student, Subject) → Teacher
- Teacher → Subject ❌Violation (Teacher is not a super key)

✅Decomposition:

TEACHER (Teacher, Subject)
STUDENT_TEACHER (Student, Teacher)

Now table is in **BCNF**.

Advantages of Normalization

- Eliminates **data redundancy**
- Prevents **update anomalies**
- Improves **data consistency**
- Saves **storage space**
- Makes database **easy to maintain**

Disadvantages of Normalization

- Increases **number of tables**
- More **joins required**
- May reduce **performance** in large systems

TRANSACTION PROCESSING AND CONCURRENCY CONTROL (DBMS)

Introduction to Transaction Processing

Transaction – Definition

A **transaction** is a **sequence of database operations** (read, write, update, delete) that **must be executed as a single logical unit of work**.

A transaction ends with either:

- **COMMIT** → Changes are permanently saved
- **ROLLBACK** → Changes are undone

Example of a Transaction (Bank Transfer)

1. Read balance of Account A
2. Subtract ₹1000 from Account A
3. Add ₹1000 to Account B
4. Save changes (COMMIT)

If any step fails → **ROLLBACK** is done.

States of a Transaction

1. **Active** – Transaction is executing
2. **Partially Committed** – Last statement executed
3. **Committed** – Changes permanently saved
4. **Failed** – Error occurs
5. **Aborted** – Rolled back due to failure

Types of Transactions

- **Read-only transaction**
- **Read-write transaction**

Desirable Properties of Transactions (ACID Properties)

For reliable transaction processing, a transaction must satisfy **ACID properties**:

1. Atomicity

- A transaction must be treated as **all-or-nothing**
- Either **all operations succeed** or **none are applied**
- If failure occurs → **rollback**

Example:
Money must be both deducted and deposited, not only one

2. Consistency

- A transaction must take the database from **one valid state to another valid state**
- All **integrity constraints** must be satisfied

Example:
Total balance in a bank remains correct after transfer.

3. Isolation

- Each transaction must **execute independently**
- Intermediate results must **not be visible to other transactions**

4. Durability

- Once a transaction is **committed**, its changes are **permanent**
- Even if power failure occurs, data remains saved

Basic Concepts of Concurrency Control

Concurrency – Definition

Concurrency means **multiple transactions executing at the same time**.

Why Concurrency is Needed?

- Better **CPU utilization**
- Faster **response time**
- Higher **system throughput**

Concepts of Locks

A **lock** is a mechanism used by DBMS to **control access to data items** by transactions.

Types of Locks

1. Shared Lock (S-Lock)

- Used for **reading**

- Multiple transactions can read at the same time
- No updates allowed

2. Exclusive Lock (X-Lock)

- Used for **writing**
- Only one transaction can access the data
- Others are blocked

Dead Lock

Definition:

A **deadlock** occurs when **two or more transactions wait indefinitely for each other's resources**.

Example:

- T1 locks A and waits for B
- T2 locks B and waits for A
  → Both wait forever → **Deadlock**

Conditions for Deadlock:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

Deadlock Handling

- Deadlock prevention
- Deadlock detection
- Deadlock recovery

Live Lock

Definition:

A **livelock** occurs when:

- A transaction **keeps changing its state repeatedly**
- But **never makes progress**
- It keeps restarting due to conflicts

SECURITY AND INTEGRITY

Database security and integrity are essential to:

- Protect data from **unauthorized access**
- Prevent **data loss or corruption**
- Maintain **correctness and reliability** of data

Security and Integrity Violation

1. Database Security

Database Security refers to the **protection of the database against unauthorized access, misuse, and damage**.

It protects:

- Data
- DBMS software
- Hardware resources

2. Integrity

Data Integrity ensures that the **data stored in the database is accurate, consistent, and reliable** at all times.

3. Security Violations

A **security violation** occurs when an **unauthorized user gains access to database resources**.

Examples:

- Unauthorized data reading
- Unauthorized data modification
- Data deletion
- Identity theft
- Data leakage

4. Integrity Violations

An **integrity violation** occurs when **incorrect, inconsistent, or invalid data** enters the database.

Examples:

- Inserting a student without roll number
- Entering duplicate primary key values
- Foreign key mismatch

5. Types of Integrity Constraints (for Prevention)

1. **Domain Integrity**
   - Values must be within a valid range
     Example: Age > 0
2. **Entity Integrity**
   - Primary key must be **unique and not NULL**
3. **Referential Integrity**
   - Foreign key must match a primary key in another table
4. **User-defined Integrity**
   - Business rules defined by users

Authorization

Definition:

Authorization is the process of **granting or denying privileges** to users to access database objects.

It ensures:

- Only **authorized users** can perform specific operations
- Different users have **different access rights**

Types of Privileges

1. **System Privileges**
   - Create user
   - Create database
   - Backup database
2. **Object Privileges**
   - SELECT
   - INSERT
   - UPDATE
   - DELETE
   - REFERENCES

Advantages of Authorization

- Prevents unauthorized data access
- Improves data security
- Controls user responsibilities
- Protects sensitive information

Authorization and Views

What is a View?

A **view** is a **virtual table** created from one or more tables using a query.
It **does not store data physically**.

Why Views Are Used for Security?

Views are used to:

- **Hide sensitive columns**
- Provide **restricted access**
- Allow users to see **only required data**

Advantages of Authorization Using Views

- Column-level security
- Data abstraction
- Simplifies user access
- Prevents accidental modification

- Enhances privacy

DISTRIBUTED DATABASES (DBMS)

A **Distributed Database** is a collection of **logically related databases distributed over different locations** but connected through a network.
The entire system acts as **a single database to users**.

Principles of Distributed Database

Definition

A **Distributed Database System (DDBMS)** is a system in which:

- Data is stored at **multiple physical locations**
- Locations are connected using a **network**
- Users access data as if it were stored in **one single data**

Advantages of Distributed Databases

- High availability
- Improved performance
- Better reliability
- Easy scalability
- Local control of data

Disadvantages of Distributed Databases

- Complex design
- Higher cost
- Network dependency
- Difficulty in maintaining consistency
- Security management is challenging