

Database Management System (CSC265)

B.Sc. CSIT, IV Semester



Prepared By:

Ms. Sarita Neupane

Senior Lecturer

Kathford International College of Engineering and Management

Email: sarita.neupane@kathford.edu.np

Unit 2: Database System – Concepts and Architecture

Data Models:

A **data model** is a conceptual framework used to **describe how data is structured, stored, and manipulated** in a database. It provides a set of rules, concepts, and diagrams for defining the **logical organization** of data and the **relationships** between different data elements.

Why Data Model?

- To represent real-world entities and their relationships.
- To define how data is organized and how users interact with it.
- To support data consistency, clarity, and integrity in the database design process.

Types of Data Models

1. Hierarchical Data Model
2. Network Data Model
3. Entity-Relationship Data Model
4. Relational Data Model
5. Object – Oriented Data Model
6. NoSQL Data Model

1. Hierarchical Data Model

A **hierarchical data model** is a data model in which the data is organized into a tree-like structure. The data are stored as **records** which is a collection of one or more **fields**.

Characteristics:

- ✓ Each parent can have **multiple child records**.

- ✓ But each child **has only one parent** (1:N relationship).
- ✓ The **topmost record** is called the **root**.
- ✓ Data is accessed using **navigational paths** from parent to child.

Hierarchical database model

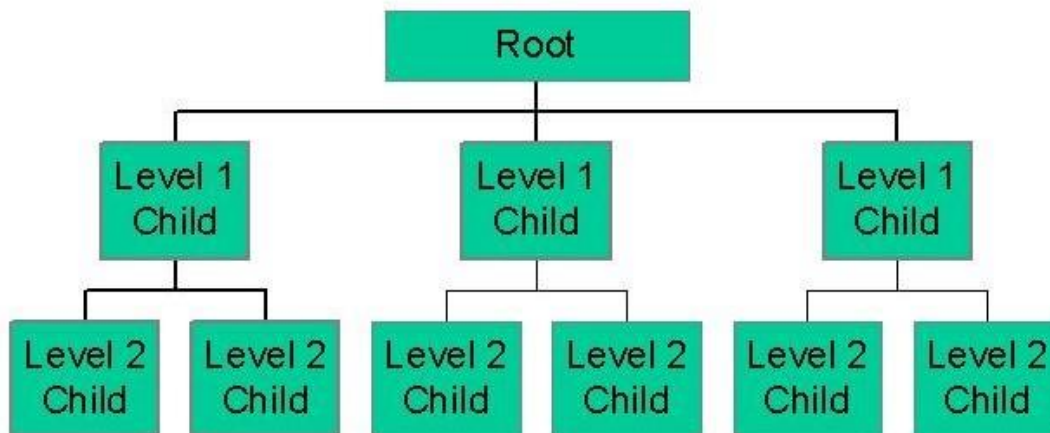


Fig: Hierarchical Data Model

Disadvantages:

- ✓ **Lack of flexibility:** Cannot handle many-to-many relationships easily.
- ✓ **Redundancy:** Data duplication if children need to be linked to multiple parents.
- ✓ **Complex navigation:** Must follow the hierarchy path even to access lower-level data.
- ✓ **Does not support Many-Many Relationships**

2. Network Data Model:

The **Network Data Model** is an early database model that extends the **hierarchical model** by allowing **many-to-many (M:N) relationships** between records. It organizes data as a **graph** where records (nodes) are connected via **pointers** (edges).

Key Characteristics

1. Graph Structure:

- ✓ **Records** = Nodes.
- ✓ **Relationships** = Directed edges (pointers).

2. Flexible Relationships:

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

3. Owner-Member Relationships:

- ✓ Each relationship has an **owner record** and **member records**.

4. Navigational Access:

- ✓ Data is retrieved by traversing pointers (no SQL-like queries).

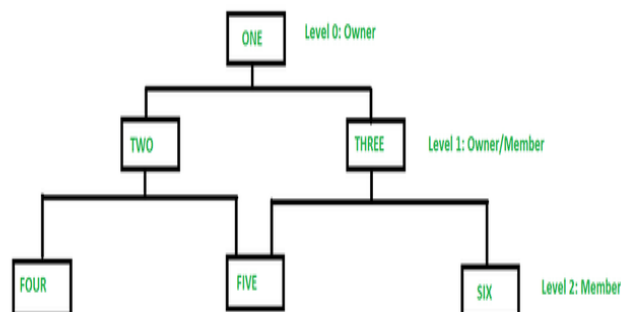


Figure: Network Data Model

Advantages of Network Data Model:

- ✓ This model is very simple and easy to design like the hierarchical data model.
- ✓ This model is capable of handling multiple types of relationships which can help in modeling real-life applications, for example, 1: 1, 1: M, M: N relationships.
- ✓ In this model, we can access the data easily, and also there is a chance that the application can access the owner's and the member's records within a set.
- ✓ This network does not allow a member to exist without an owner which leads to the concept of Data integrity.
- ✓ Like a hierarchical model, this model also does not have any database standard,
- ✓ This model allows to represent multi parent relationships.

Disadvantages of Network Data Model:

- ✓ The schema or the structure of this database is very complex in nature as all the records are maintained by the use of pointers.
- ✓ There's an existence of operational anomalies as there is a use of pointers for navigation which further leads to complex implementation.
- ✓ The design or the structure of this model is not user-friendly.
- ✓ This model does not have any scope of automated query optimization.
- ✓ This model fails in achieving structural independence even though the network database model is capable of achieving data independence.

3. E-R Data Model:

The **Entity-Relationship (E-R) Model** is a **conceptual framework** used to design databases by visually representing data as **entities**, **attributes**, and **relationships**. It acts as a **blueprint** before implementing a database in relational, NoSQL, or other systems.

- ✓ **Entity:** An objects that is stored as data such as *Student*, *Course* or *Company*.

- ✓ **Attribute:** Properties that describes an entity such as *StudentID*, *CourseName*, or *EmployeeEmail*.
- ✓ **Relationship:** A connection between entities such as "a *Student* enrolls in a *Course*".

The graphical representation of this model is called an Entity-Relation Diagram (ERD).

Components of E-R Diagram:

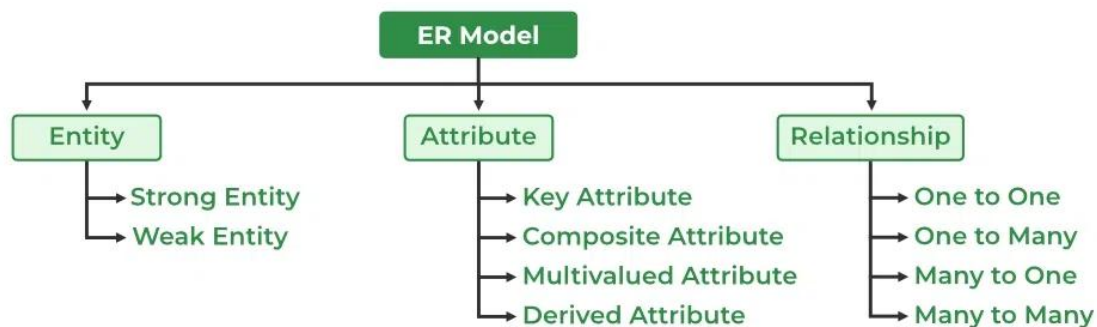


Figure: Components of E-R Diagram

ER Model in Database Design Process

We typically follow the below steps for designing a database for an application.

- ✓ Gather the requirements (functional and data) by asking questions to the database users.
- ✓ Create a logical or conceptual design of the database. This is where ER model plays a role. It is the most used graphical representation of the conceptual design of a database.
- ✓ After this, focus on Physical Database Design (like indexing) and external design (like views)

Why Use ER Diagrams In DBMS?

- ✓ ER diagrams represent the E-R model in a database, making them easy to convert into relations (tables).
- ✓ These diagrams serve the purpose of real-world modeling of objects which makes them intently useful.
- ✓ Unlike technical schemas, ER diagrams require no technical knowledge of the underlying DBMS used.

- They visually model data and its relationships, making complex systems easier to understand.






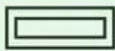
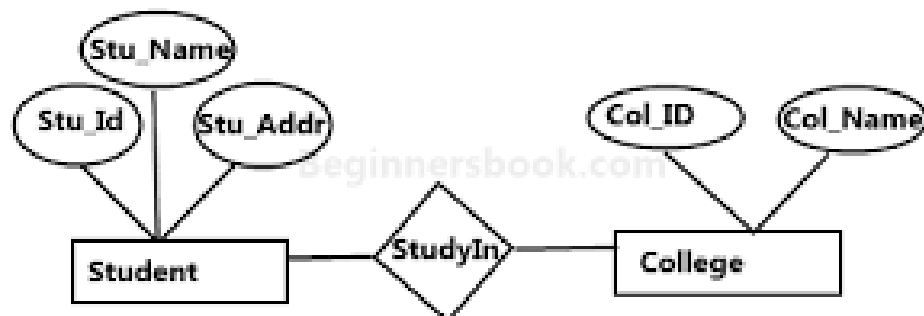
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Figure: Symbols used in ER Data Model



Sample E-R Diagram

4. Relational Data Model

The **Relational Data Model** represents data and their relationships through a collection of tables. Each table also known as a **relation** consists of rows and columns. Every

column has a unique name and corresponds to a specific attribute, while each row contains a set of related data values representing a real-world entity or relationship. This model is part of the record-based models which structure data in fixed-format records each belonging to a particular type with a defined set of attributes.

E.F. Codd introduced the Relational Model to organize data as relations or tables. After creating the conceptual design of a database using an ER diagram, this design must be transformed into a relational model which can then be implemented using relational database systems like Oracle SQL or MySQL.

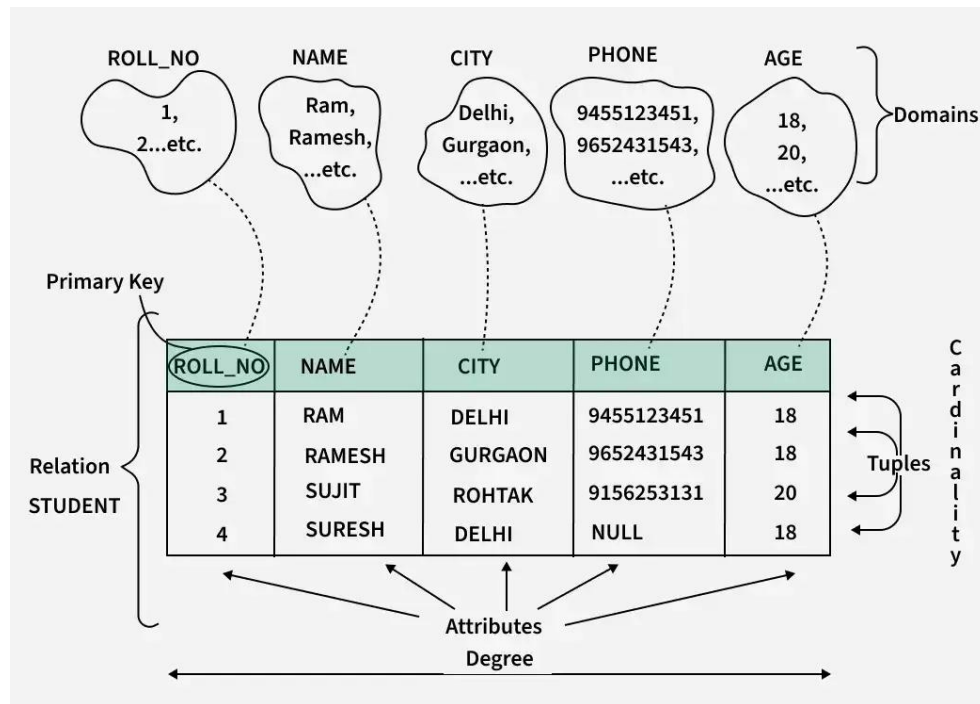


Figure: Example of Relational Data Model

Characteristics of the Relational Model

- 1. Data Representation:** Data is organized in tables (relations), with rows (tuples) representing records and columns (attributes) representing data fields.
- 2. Atomic Values:** Each attribute in a table contains atomic values, meaning no multi-valued or nested data is allowed in a single cell.

- 3. Unique Keys:** Every table has a primary key to uniquely identify each record, ensuring no duplicate rows.
- 4. Attribute Domain:** Each attribute has a defined domain, specifying the valid data types and constraints for the values it can hold.
- 5. Tuples as Rows:** Rows in a table, called tuples, represent individual records or instances of real-world entities or relationships.
- 6. Relation Schema:** A table's structure is defined by its schema, which specifies the table name, attributes, and their domains.
- 7. Data Independence:** The model ensures logical and physical data independence, allowing changes in the database schema without affecting the application layer.
- 8. Integrity Constraints:** The model enforces rules like:
- 9. Domain constraints:** Attribute values must match the specified domain.
- 10. Entity integrity:** No primary key can have NULL values.
- 11. Referential integrity:** Foreign keys must match primary keys in the referenced table or be NULL.
- 12. Relational Operations:** Supports operations like **selection, projection, join, union, and intersection**, enabling powerful data retrieval manipulation.
- 13. Data Consistency:** Ensures data consistency through constraints, reducing redundancy and anomalies.

Key Terms in the Relational Model

- 1. Attribute:** Attributes are the properties that define an entity.

Example: ROLL_NO, NAME, ADDRESS etc.

- 2. Relation Schema:** A relation schema defines the structure of the relation and represents the name of the relation with its attributes.

Example: STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, and AGE) is the relation schema for STUDENT. If a schema has more than 1 relation it is called Relational Schema.

- 3. Tuple:** A **Tuple** represents a row in a relation. Each tuple contains a set of attribute values that describe a particular entity.

Example: (1, RAM, DELHI, 9455123451, 18) is a tuple in the STUDENT table.

- 4. Relation Instance:** The set of tuples of a relation at a particular instance of time is called a **relation instance**. It can change whenever there is an insertion, deletion or update in the database.

5. Degree: The number of attributes in the relation is known as the degree of the relation.

Example: The STUDENT relation has a degree of 5, as it has 5 attributes.

6. Cardinality: The number of tuples in a relation is known as cardinality.

Example: The STUDENT relation defined above has cardinality 4.

7. Column: The column represents the set of values for a particular attribute.

Example: The column ROLL_NO is extracted from the relation STUDENT.

8. NULL Values: The value which is not known or unavailable is called a NULL value. It is represented by NULL.

Example: PHONE of STUDENT having ROLL_NO 4 is NULL.

5. Object – Oriented Data Model:

The Object-Oriented Model in DBMS or OODM is the data model where data is stored in the form of objects. This model is used to represent real-world entities. The data and data relationship are stored together in a single entity known as an object in the Object-Oriented Model. The Object-Oriented Database Management System is built on top of Object-Oriented Model.

We can use the Object-Oriented Model in DBMS to store real-world entities. Here, we can store pictures, audio, video, and other types of data, which was previously impossible to store with the relational approach (Even though we can store video and audio in the relational database, it is generally not recommended).

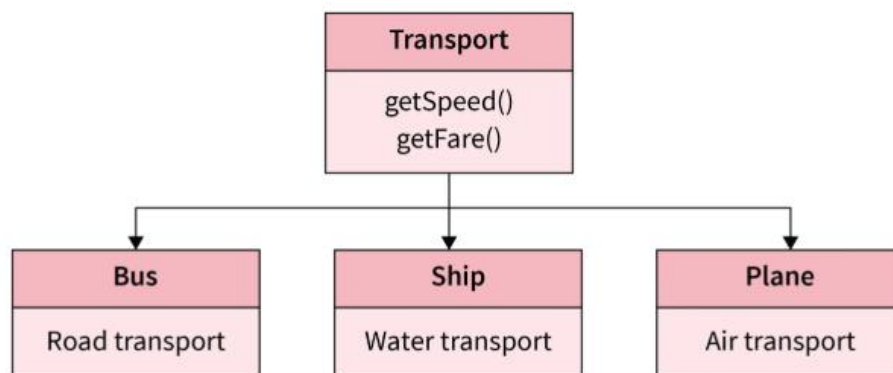


Figure: Example of Object-Oriented Data Model

- ✓ Here Transport, Bus, Ship, and Plane are objects.
- ✓ Bus has Road Transport as the attribute.
- ✓ Ship has Water Transport as the attribute.
- ✓ Plane has Air Transport as the attribute.
- ✓ The Transport object is the base object and the Bus, Ship, and Plane objects derive from it.

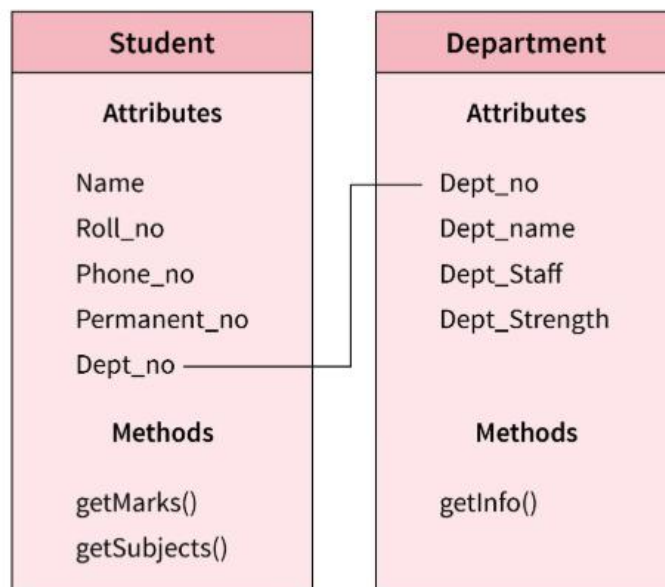
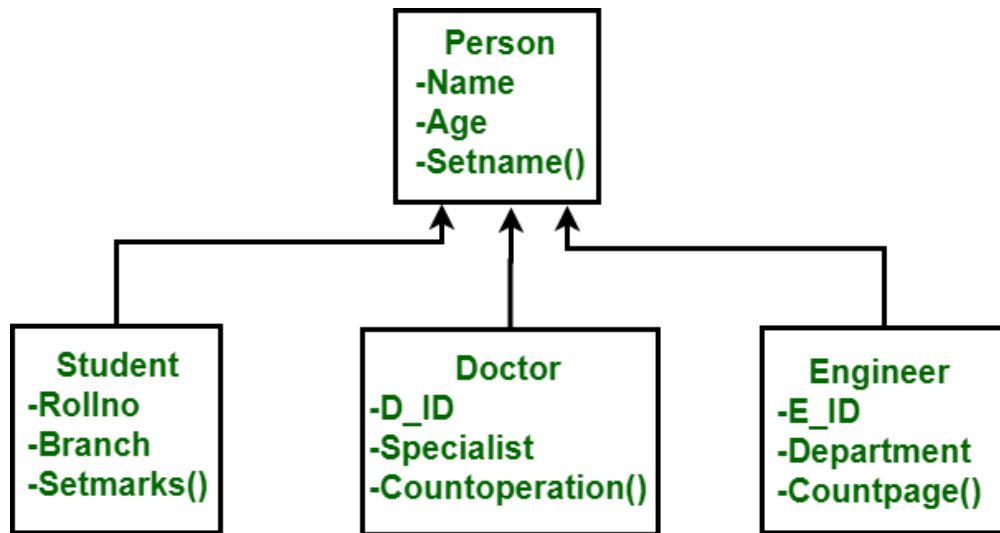


Figure: Another example of Object-Oriented Data Model

Components of Object-Oriented Data Model

Components of the Object-Oriented Data Model namely objects, classes, object attributes, class hierarchy, etc., are explained as follows- Object- It is a physical or a real-world entity.

Object attribute- The objects have certain characteristics. These are known as the attributes of the object.

Object method- The object's behavior is shown using object methods.

Class- It is a collection of similar kinds of objects. It is an entity that has attributes and methods together.

Inheritance- It is the ability of the object within the class hierarchy to inherit the attributes and methods of the classes above it. A new class can be derived from an existing class, the new class has the attributes and methods described in the existing class and also has its attributes and methods. This helps in code reusability.

6. No SQL Model:

NoSQL (Not Only SQL) is a **non-relational** database model designed for **flexible, scalable, and high-performance** data management. Unlike SQL databases, NoSQL does not enforce rigid schemas and is optimized for **unstructured, semi-structured, or distributed data**.

Types of NoSQL Databases

1. Document-Oriented Model

- ✓ **Structure:** Stores data as **JSON-like documents**.
- ✓ **Use Case:** Content management, catalogs, user profiles.

Examples:

- ✓ **MongoDB** (Flexible schema for e-commerce products).
- ✓ **CouchDB** (Offline-first apps).

2. Key-Value Model

- ✓ **Structure:** Simple key → value pairs (like a dictionary).
- ✓ **Use Case:** Caching, session management, real-time analytics.

Examples:

- ✓ **Redis** (In-memory caching for high-speed access).
- ✓ **DynamoDB** (AWS's scalable key-value store).

3. Column-Family Stores

- ✓ **Structure:** Stores data in **columns instead of rows** (optimized for analytics).
- ✓ **Use Case:** Time-series data, big data analytics.

Examples:

- ✓ **Cassandra** (Used by Netflix for streaming data).
- ✓ **HBase** (Hadoop integration).

4. Graph Databases

1. **Structure:** Nodes (entities) + Edges (relationships).
2. **Use Case:** Social networks, fraud detection, recommendation engines.

Examples:

- ✓ **Neo4j** (Friend-of-friend queries).
- ✓ **Amazon Neptune** (Knowledge graphs).

Database Schema:

A database schema is the blueprint or structure of a database. It defines how data is organized and how relationships between data are maintained. It includes table definitions, columns, data types, constraints, and relationships.

1. It is static — meaning it does not change frequently.

- ✓ Defined using Data Definition Language (DDL) statements like CREATE TABLE.

A simple schema for a student database:

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR (50),  
    Email VARCHAR (100),  
    DOB DATE  
);
```

Database Instances:

A database instance is the actual data stored in the database at a particular moment in time. It is the current state of the database.

- ✓ It is **dynamic** — changes whenever data is inserted, updated, or deleted.

Given the Students table schema, a database instance might look like:

StudentID	Name	Email	DOB
101	Aayushma	aayu@gmail.com	2001-05-12
102	Sofia	sofia@example.com	2002-07-08

Differences between Schema and Instance

Schema	Instance
The structure or design of the database (tables, fields, relationships, constraints).	A snapshot of the data stored in the database at a specific moment in time.
Static; defines the organization of data.	Dynamic; represents the actual data stored.
Changes infrequently (when the database structure is modified).	Changes frequently (as data is inserted, updated, or deleted).
Defines the rules, constraints, and layout for data storage.	Reflects the current state of the data in the database.
Similar to declaring variables in a program (defining their structure).	Similar to the values assigned to variables at a particular moment.
Remains largely unchanged unless structural changes are made.	Changes continuously as data is modified.
A table structure with columns like "ID", "Name", "Age".	The actual records of data in those columns (e.g., "1, John, 25").

Three-Level Schema Architecture

The Three- Level Schema Architecture in DBMS is a conceptual framework used to describe how data is organized in a database system. It divides the database system into three different levels of abstraction, promoting data independence and making database systems more flexible and user-friendly.

It divides the database into three different levels:

1. Internal Schema (Physical Level)
2. Conceptual Schema (Logical Level)
3. External Schema (View Level)

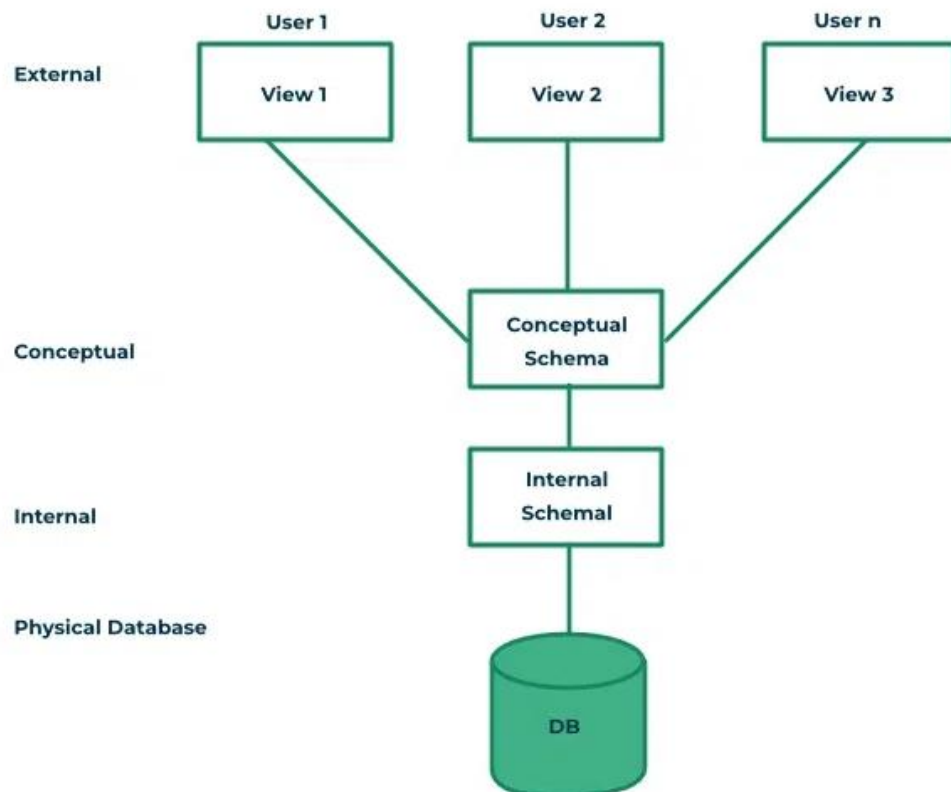


Figure: Three-Level Schema Architecture

1. Internal Schema (Physical Level)

- ✓ Describes **how the data is physically stored** in the database.
- ✓ Includes details such as **file formats, indexing, compression**, and storage blocks.
- ✓ Handled by the **DBMS and system administrators**.

Example: Data is stored using B-trees or hash indexes in disk blocks.

2. Conceptual Schema (Logical Level)

- ✓ Describes the **structure of the entire database** for the community of users.
- ✓ Includes definitions of **tables, data types, relationships, constraints**, etc.
- ✓ Hides the physical details and focuses on what data is stored and how it's related.

Example: A university database schema defines entities like `Student`, `Course`, and `Enrollment` and their relationships.

3. External Schema (View Level)

1. Defines **how different users view the data**.
2. Allows creation of **customized views** for different user roles (e.g., admin, student).
3. Supports **data security** by restricting access to only what a user need.

Example:

- A **student** sees only their grades.
- A **professor** sees all students in their course.

Data Independence

Data Independence is a key concept in database management systems (DBMS) that refers to the ability to change a database's schema (structure) at one level without affecting the schema at higher levels. It provides abstraction between different levels of data storage, making the system more flexible, easier to maintain, and less prone to disruptions when changes occur.

There are two types of data independence.

- ✓ logical data independence
- ✓ Physical data independence

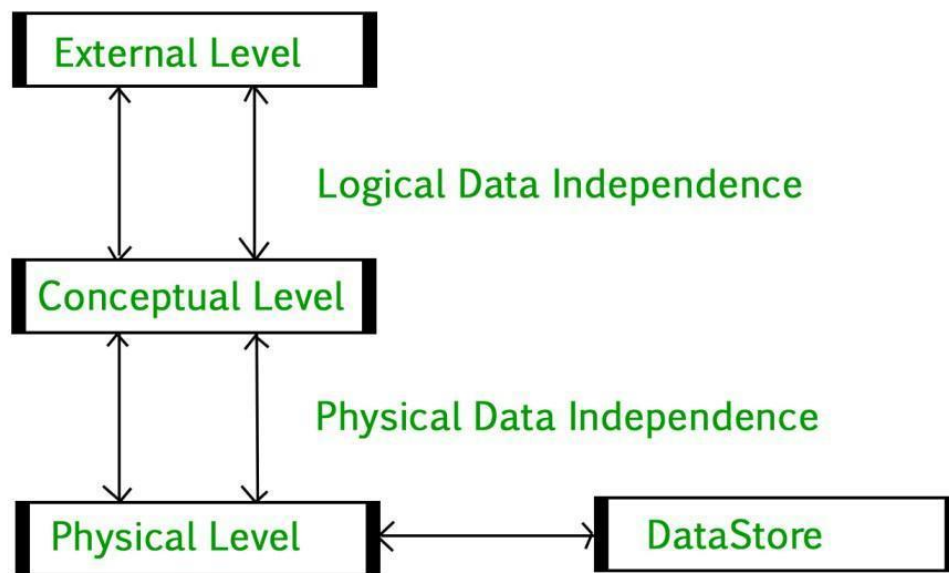


Figure: Data Independence

Logical Data Independence

- ✓ Changing the logical schema (conceptual level) without changing the external schema (view level) is called logical data independence.
- ✓ It is used to keep the external schema separate from the logical schema.
- ✓ If we make any changes at the conceptual level of data, it does not affect the view level.
- ✓ This happens at the user interface level.
- ✓ For example, it is possible to add or delete new entities, attributes to the conceptual schema without making any changes to the external schema.

Physical Data Independence

- ✓ Making changes to the physical schema without changing the logical schema is called physical data independence.
- ✓ If we change the storage size of the database system server, it will not affect the conceptual structure of the database.
- ✓ It is used to keep the conceptual level separate from the internal level.
- ✓ This happens at the logical interface level.
- ✓ Example – Changing the location of the database from C drive to D drive.

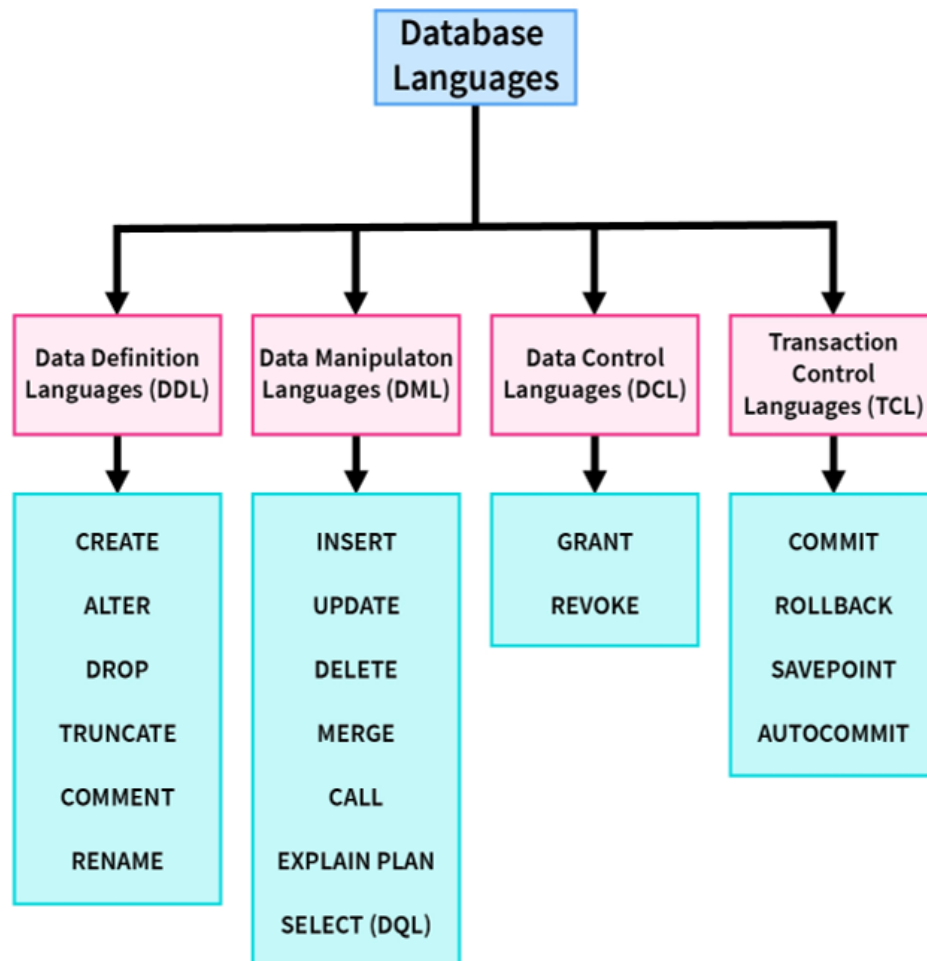
Difference Between Physical and Logical Data Independence

Physical Data Independence	Logical Data Independence
It mainly concerns how the data is stored in the system.	It mainly concerns about changes to the structure or data definition.
To make changes at the physical level we generally do not require changes at the application program level.	To make changes at the logical level, we need to make changes at the application level.
It tells about the internal schema.	It tells about the conceptual schema.
There may or may not be a need for changes to be made at the internal level to improve the structure.	Whenever the logical structure of the database has to be changed, the changes made at the logical level are important.
Example- change in compression technology, hashing algorithm, storage device etc.	Example – adding/modifying or deleting a new attribute.

Database Languages and Interfaces:

Database languages are specialized languages used to interact with a database. They allow users to perform different tasks such as defining, controlling, and manipulating the data. There are several types of database languages in DBMS, categorized into the following four main types:

1. DDL (Data Definition Language)
2. DML (Data Manipulation Language)
3. DCL (Data Control Language)
4. TCL (Transaction Control Language)



DDL (Data Definition Language)

Data definition language (DDL) creates the framework of the database by specifying the database schema, which is the structure that represents the organization of data. Its common uses include the creation and alteration of tables, files, indexes and columns within the database.

Key Commands:

- CREATE: Creates a new database or object, such as a table, index or column
- ALTER: Changes the structure of the database or object
- DROP: Deletes the database or existing objects
- RENAME: Renames the database or existing objects

DML (Data Manipulation Language)

Data manipulation language (DML) provides operations that handle user requests, offering a way to access and manipulate the data that users store within a database. Its common functions include inserting, updating and retrieving data from the database.

Key Commands:

- INSERT: Adds new data to the existing database table
- UPDATE: Changes or updates values in the table
- DELETE: Removes records or rows from the table
- SELECT: Retrieves data from the table or multiple tables

DCL (Data Control Language)

Data control language (DCL) controls access to the data that users store within a database. Essentially, this language controls the rights and permissions of the database system. It allows users to grant or revoke privileges to the database.

Key Commands:

GRANT: Gives a user access to the database

REVOKE: Removes a user's access to the database

TCL (Transaction Control Language)

Transaction control language (TCL) manages the transactions within a database. Transactions group a set of related tasks into a single, executable task. All the tasks must succeed in order for the transaction to work.

Key Commands:

- COMMIT: Carries out a transaction
- ROLLBACK: Restores a transaction if any tasks fail to execute
- SAVEPOINT: Sets a point in a transaction to save

Database Interfaces:

A database interface is a method or tool that allows users or applications to interact with the Database Management System (DBMS).

1. Command-Line Interface (CLI)

- ✓ Users type commands directly into a terminal or shell.
- ✓ Best suited for **technical users** like DBAs and developers.
- ✓ Offers **full control** over the database.

Examples:

- MySQL Shell
- Oracle SQL*Plus

2. Graphical User Interface (GUI)

- ✓ **User-friendly interface** using forms, buttons, and menus.
- ✓ Suitable for **non-technical users** or beginners.
- ✓ Makes database tasks easier via visual tools.

Examples:

- phpMyAdmin (for MySQL)
- SQL Server Management Studio (SSMS)
- Oracle SQL Developer

3. Application Program Interface (API)

- ✓ Allows software applications to communicate with the database using programming languages.
- ✓ Enables dynamic data processing in apps.

Examples:

- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

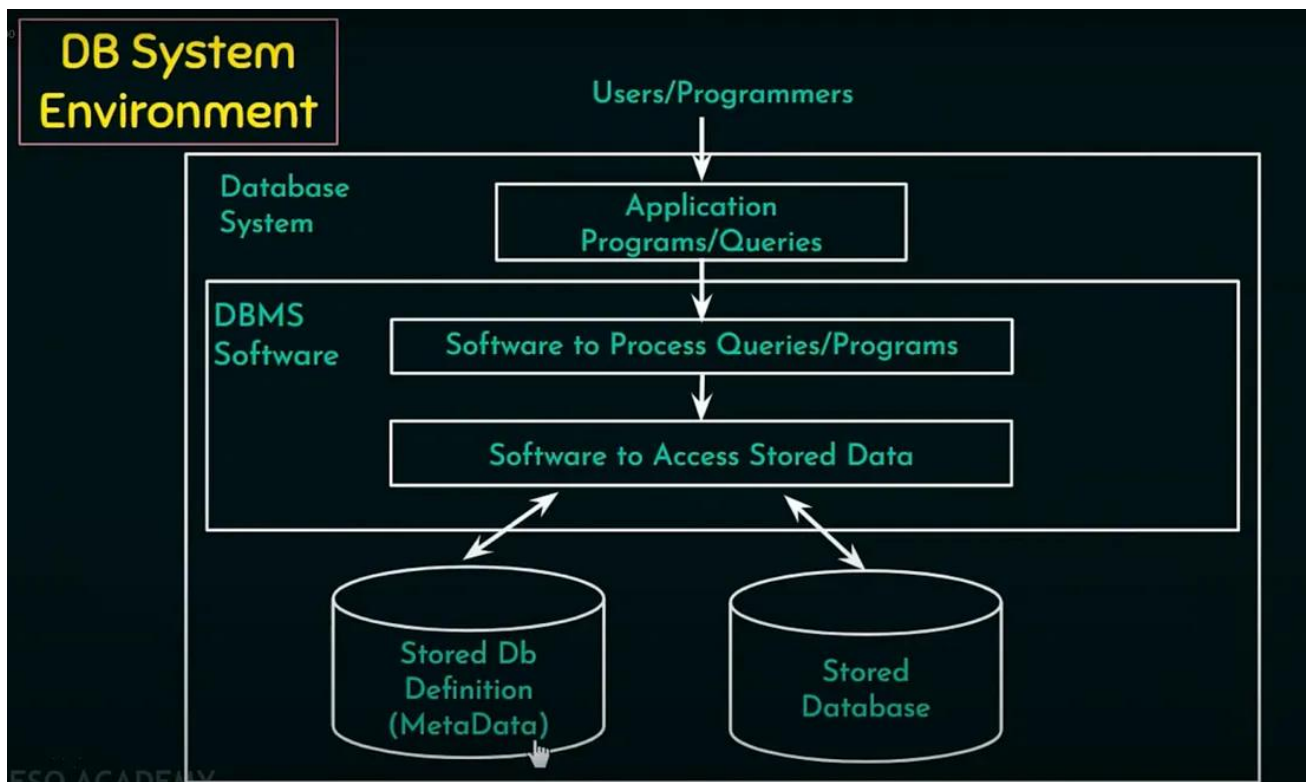
4. Web Interface

- ✓ Browser-based interface to interact with the database via web forms or dashboards.
- ✓ Used in online applications like portals and CMS.

Examples:

- A student portal in College Management System
- Online booking systems (hotel, airline, etc.)

Database System Environment



The database system environment refers to the entire ecosystem that supports database operations, including hardware, software, data, people, and procedures. It ensures efficient storage, retrieval, security, and management of data in a structured manner.

1. Key Components of the Database System Environment

A. Hardware

- ✓ **Database Servers:** Host the DBMS (e.g., Oracle, MySQL, MongoDB).
- ✓ **Storage Devices:** HDDs, SSDs, or cloud storage for data persistence.
- ✓ **Memory (RAM):** Used for caching and fast query execution.
- ✓ **Networking:** Enables communication between clients and servers.

B. Software

- ✓ **Database Management System (DBMS):** Core software (e.g., PostgreSQL, SQL Server).
- ✓ **Operating System:** Runs the DBMS (Linux, Windows Server).
- ✓ **Application Software:** Programs that interact with the database (e.g., ERP, CRM).
- ✓ **Utilities:** Backup tools, import/export tools, and monitoring software.

C. Data

- ✓ **User Data:** Actual business data (e.g., customer records, transactions).
- ✓ **Metadata:** Data about data (e.g., table schemas, indexes).
- ✓ **Application Data:** Forms, reports, and query definitions.

D. People

- ✓ **Database Administrators (DBAs):** Manage, secure, and optimize databases.
- ✓ **Database Designers:** Create schemas and define relationships.
- ✓ **Application Developers:** Write code to interact with the database.
- ✓ **End Users:** Access data via applications (e.g., sales teams, analysts).

E. Procedures

- ✓ **Database Design:** Steps to create and normalize schemas.
- ✓ **Backup & Recovery:** Strategies to prevent data loss.
- ✓ **Security Policies:** Authentication, authorization, and encryption rules.
- ✓ **Performance Tuning:** Query optimization and indexing strategies.

Centralized and Client/Server Architecture

Centralized Architecture:

A **centralized architecture** for a **Database Management System (DBMS)** is a system where all database resources—**data, DBMS software, and control mechanisms**—are located on a **single central server**. Users access this central system through **terminals or client machines** via a network.

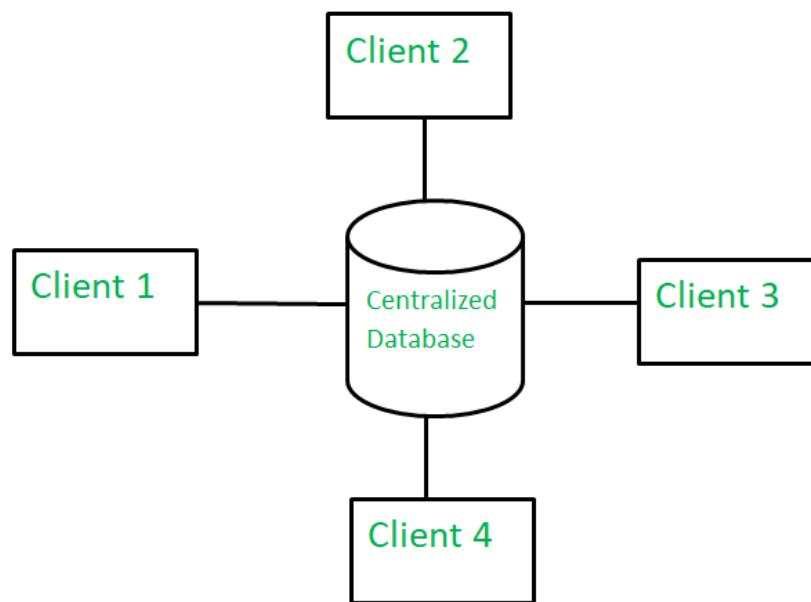


Figure: Centralized Database

Advantages

- ✓ Since all data is stored at a single location only thus it is easier to access and coordinate data.

- ✓ The centralized database has very minimal data redundancy since all data is stored in a single place.
- ✓ It is cheaper in comparison to all other databases available.

Disadvantages

- ✓ The data traffic in the case of a centralized database is more.
- ✓ If any kind of system failure occurs in the centralized system then the entire data will be destroyed.

When to Use Centralized Architecture

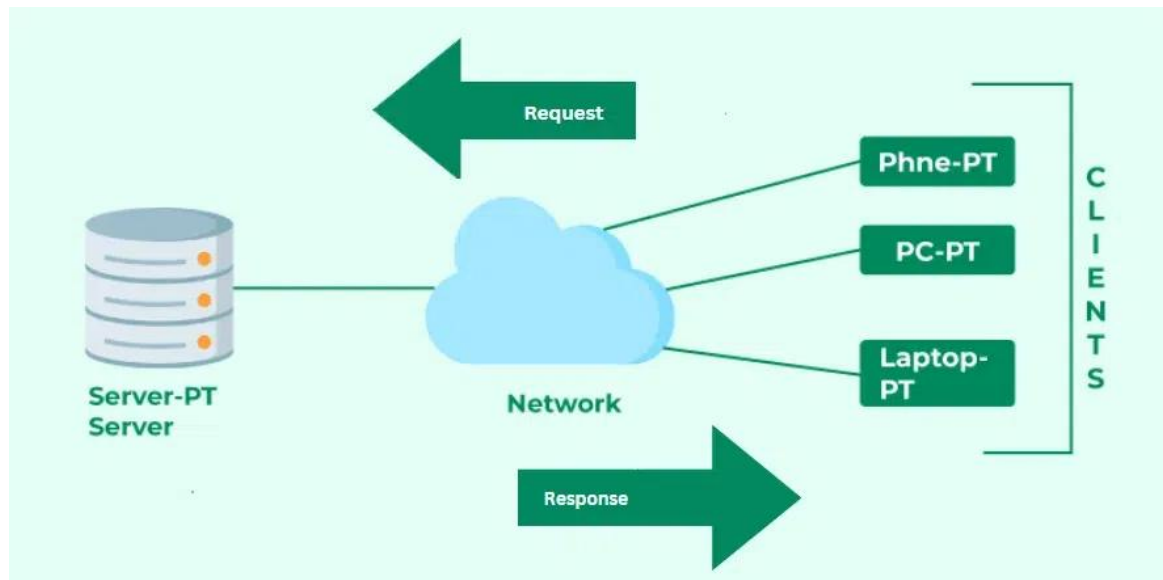
- ✓ Small to medium-sized organizations
- ✓ Applications with limited number of concurrent users
- ✓ Systems where data consistency is critical
- ✓ Budget-constrained projects
- ✓ Applications that don't require high availability

Client/Server Architecture in DBMS

The **Client/Server (C/S) architecture** is a widely used database model that separates the DBMS into two main components:

- ✓ **Clients** (front-end) that request services
- ✓ **Servers** (back-end) that process requests and return results

This architecture improves scalability, performance, and flexibility compared to a purely centralized system.



Key Components of Client/Server DBMS Architecture

1. Client (Front-End)

- ✓ Responsible for the **user interface** and application logic
- ✓ Sends requests (queries) to the server
- ✓ Receives and displays results
- ✓ Examples: Web browsers, mobile apps, desktop applications

2. Server (Back-End)

- ✓ Handles **data storage, processing, and management**
- ✓ Executes queries and transactions
- ✓ Ensures data integrity, security, and concurrency control
- ✓ Returns results to clients
- ✓ Examples: MySQL Server, Oracle Database, Microsoft SQL Server

Types of Client/Server DBMS Architectures

1. Two-Tier Architecture

- ✓ **Clients** directly interact with the **database server**
- ✓ Business logic can be on the client or server side

- ✓ Simple but less scalable

Example:

A desktop application (client) connects to a MySQL database (server)

2. Three-Tier Architecture

- ✓ Introduces a **middle layer (application server)** between client and database
- ✓ Improves scalability, security, and flexibility

Layers:

1. **Presentation Tier (Client)** – User interface
2. **Application Tier (Middleware)** – Business logic (e.g., APIs, web servers)
3. **Database Tier (Server)** – Data storage & retrieval

Example:

Web browser (client) → Node.js/Java server (middleware) → PostgreSQL (database)

3. N-Tier Architecture

- ✓ Extends three-tier with additional layers (e.g., caching, microservices)
- ✓ Used in large-scale distributed systems