



# UNIT 1 – Database Fundamentals

## 1.1 Introduction to Data and Databases

### Subtopics:

What is Data | Information vs Data | What is Database | Role of Database

### Learning Outcome:

By the end of this lesson, you will be able to –

- ✓ Define data, information and database in clear terms
- ✓ Explain how raw data transforms into information
- ✓ Recognize the role of databases in modern systems

## Concept Explanation

Picture this, baby – you have a notebook filled with random numbers and names.

That's **data** — raw, unstructured, and meaningless until you organize it.

Now, you rearrange it like this:

Name	Age	Phone
Dinesh	25	9876543210

You just turned data into **information** ✨.

That tiny act is what every database does — it gives structure to chaos.

## What is Data?

**Data** is a collection of raw facts or observations that by themselves don't carry meaning.

They can be numbers, names, symbols or measurements.

**Examples:** 25, Chennai, A+, True, [90, 85, 92, 78]



### Real-life analogy:

Your phone stores photos, texts, and calls — that's *data*.


When the gallery groups them by date or person, that's *information*.

## Information vs Data

Aspect	Data	Information
Definition	Raw, unprocessed facts	Processed, meaningful data




Example	"John", "95", "Physics"	"John scored 95 in Physics."
Meaning	No context	Has context and purpose
Usefulness	Limited	Useful for decision-making

 **Simplified:** Data is what we collect; Information is what we understand.

## The Journey of Data → Information → Database → Decision

Data → Organize → Information → Store → Database → Analyze → Decision

 Example:

Raw sales numbers → stored in tables → summarized as reports → used for business decisions.

This flow is the *heartbeat of every data-driven system*.

## What is Metadata?

**Metadata** is *data about data*.

It describes how data is organized and what it represents.

**Example:**

In a file named students.csv:

- The marks = data
- File name, size, date created = metadata

In databases, metadata is what defines tables, columns, and relationships.

It's like the **DNA** of your database structure.

## What is a Database?

A **database** is an organized collection of related data stored and managed for easy access, updating, and retrieval.

**Formal Definition:**

A database is a structured collection of data stored in a computer system and managed by a software system called a DBMS.

**Examples:**

- Your WhatsApp chat history
- Instagram profiles and followers
- College student management system

Each stores information systematically for quick use.

## Role of Database in Everyday Life

Domain	Purpose	Example
Banking	Account info, transactions	SBI Core Banking
Education	Students, marks, attendance	College ERP
E-Commerce	Products, orders, customers	Amazon, Flipkart
Healthcare	Patient records, prescriptions	Apollo Hospitals
Social Media	Posts, messages, likes	Instagram, Facebook

Every click online reads or writes to a database.

## CRUD Operations – The Core of Database Interaction

CRUD stands for:

- **C** – Create (new data)
- **R** – Read (existing data)
- **U** – Update (data changes)
- **D** – Delete (remove data)

In MySQL:

```
CREATE DATABASE my_contacts;
```

```
USE my_contacts;
```

```
CREATE TABLE friends (id INT, name VARCHAR(30), city VARCHAR(30));
```

These simple commands represent the “C” and “R” of CRUD — your first step into real database work.

## Story Time: From Chaos to Clarity

A college used to store student data in Excel sheets.

Finding one student’s marks took minutes.

After moving to MySQL, they could query it instantly and avoid errors.

That’s the real power of a database — speed, accuracy, and organization.

## Pro Connect Box – Linking to Tech Stacks

- In **MERN**, MongoDB is your database.
- In **Java Full Stack**, MySQL handles your backend data.



- In **Python Flask**, SQLAlchemy or MySQL stores your app info.  
So DBMS is the **foundation of every developer's stack**.

## Pro Tips

- Always end SQL commands with ;
- Use lowercase for database names (learning\_db)
- Back up your database regularly — treat it like your memories 📁

## Common Mistakes

- 1 Confusing data with information.
- 2 Assuming databases are only for big companies.
- 3 Ignoring metadata when designing tables.

## Summary

- **Data** = raw facts.
- **Information** = processed data with context.
- **Database** = structured storage of data.
- **Metadata** = data about data.
- **CRUD** = Create, Read, Update, Delete.

## Checkpoint Review

- 1 Define data and information in your own words.
- 2 List three real-life databases you interact with daily.
- 3 Write SQL to create a database named student\_db.
- 4 Explain metadata with an example.
- 5 What does CRUD stand for and why is it important?

## Hands-On Practice

In MySQL Workbench 🖱️

```
CREATE DATABASE college_data;
```



```
USE college_data;
CREATE TABLE student (
    roll_no INT,
    name VARCHAR(30),
    mark INT
);
SHOW TABLES;
```

🔮 Output should confirm your table creation — your first working database!

## Reflection Prompt

If you were to design a database for your daily life (e.g., expenses, tasks, friends), what would be your tables and columns?

## Real-World Connect

Netflix collects data on what you watch (data), analyzes it to find patterns (information), stores it in databases for retrieval, and uses it to recommend shows (decision).

That's DBMS in action every time you hit *"Next Episode."* 🎬

## 1.2 Database Management System (DBMS)

### Subtopics:

- Definition of DBMS
- Components of DBMS
- Advantages of DBMS over File System
- Examples of Popular DBMS Software
- DBMS vs RDBMS

### Learning Outcome:

By the end of this topic, you will be able to:

- Define DBMS and explain its main components
- Differentiate between traditional file systems and DBMS
- Understand why DBMS is essential for managing modern data systems

## Concept Explanation

Every organization collects data — about students, employees, products, or customers. But simply storing data is not enough; it must be managed efficiently, securely, and consistently.



A **Database Management System (DBMS)** is software that allows users to **create, manage, and operate databases** effectively. It acts as an interface between users and the database, ensuring that data is consistently organized and easily accessible.

## Definition

A **Database Management System (DBMS)** is a **software system** designed to store, retrieve, define, and manage data in a structured and controlled way.

### Formal Definition:

A DBMS is a software that enables users to define, create, maintain, and control access to the database.

## DBMS in Simple Terms

Think of DBMS as a **library management system**:

- The **bookshelves** represent the database.
- The **librarian** is the DBMS.
- The **students** are the users.

When a student requests a book, the librarian finds it efficiently without the student searching the shelves directly. Similarly, a DBMS retrieves data for users through queries.

## Main Functions of DBMS

1. **Data Definition** – Allows creation and modification of database structure.
2. **Data Storage & Retrieval** – Efficiently stores and retrieves data on request.
3. **Data Manipulation** – Enables insertion, deletion, and updating of records.
4. **Data Security** – Controls access and ensures only authorized users can interact with data.
5. **Backup & Recovery** – Restores data in case of hardware or software failure.
6. **Concurrency Control** – Manages simultaneous data access by multiple users.
7. **Integrity Management** – Ensures data accuracy and consistency.

## DBMS Architecture Overview

DBMS typically works on **three levels of abstraction**:

Level	Description	Example
<b>Physical Level</b>	How data is stored physically on disk.	File organization and indexing.
<b>Logical Level</b>	What data is stored and relationships among data.	Table structures, fields, keys.

<b>View Level</b>	How users interact with data.	SQL queries, forms, reports.
-------------------	-------------------------------	------------------------------

This architecture helps separate the physical storage details from user interaction, ensuring flexibility and security.

## Components of DBMS

Component	Purpose
<b>DBMS Engine</b>	The core service that communicates between the data and the application.
<b>Database Schema</b>	Defines structure — tables, fields, data types, relationships.
<b>Query Processor</b>	Interprets and executes SQL queries.
<b>Transaction Manager</b>	Ensures ACID properties (Atomicity, Consistency, Isolation, Durability).
<b>Storage Manager</b>	Manages how data is stored and retrieved on disk.
<b>Metadata Catalog</b>	Stores information about data — data about data.
<b>User Interface</b>	Provides interaction mechanisms like MySQL Workbench or command-line tools.

## Advantages of DBMS Over File System

Aspect	Traditional File System	DBMS
<b>Data Redundancy</b>	High (same data may be stored in multiple files).	Reduced by normalization.
<b>Data Integrity</b>	Difficult to maintain.	Enforced using constraints and keys.
<b>Security</b>	Limited control.	Advanced user-level access control.
<b>Backup &amp; Recovery</b>	Manual or inconsistent.	Automatic and consistent.
<b>Concurrent Access</b>	File locking issues.	Handled by transaction control.
<b>Scalability</b>	Poor for large datasets.	Highly scalable and efficient.

### In summary:

A DBMS brings order, consistency, and reliability to how data is managed — something that a file system cannot do effectively for growing organizations.

## Examples of Popular DBMS Software



Type	Examples
Relational DBMS (RDBMS)	MySQL, Oracle, PostgreSQL, SQL Server
Object-Oriented DBMS	db4o, ObjectDB
NoSQL Databases	MongoDB, Cassandra, Redis
In-Memory DBMS	SAP HANA

## DBMS vs RDBMS

Feature	DBMS	RDBMS
Data Storage	In files or hierarchical form	In tables (rows and columns)
Relationship Management	Not maintained	Enforced via foreign keys
Normalization	Not supported	Supported to avoid redundancy
Query Language	May not use SQL	Uses SQL for queries
Data Integrity	Managed manually	Ensured automatically

### Conclusion:

Every RDBMS is a DBMS, but not every DBMS is relational. MySQL, for instance, is an RDBMS.

## Hands-On Practice (MySQL)

### Create a Database

```
CREATE DATABASE company_data;  
USE company_data;
```

### Create a Sample Table

```
CREATE TABLE employees (  
    emp_id INT PRIMARY KEY,  
    emp_name VARCHAR(50),  
    dept VARCHAR(30)  
);
```

### Insert and Retrieve Data

```
INSERT INTO employees VALUES (1, 'Anita', 'HR');  
SELECT * FROM employees;
```

This exercise demonstrates the **Create** and **Read** operations within DBMS.





## Checkpoint Review

1. Define a Database Management System.
2. List any three main components of DBMS.
3. State two advantages of DBMS over file systems.
4. How does concurrency control improve data consistency?
5. What are the differences between DBMS and RDBMS?

## Reflection Prompt

If your college still uses Excel files for attendance, what problems might occur?  
How would implementing a DBMS improve this process?

## Summary

- A DBMS is software used to create, manage, and maintain databases.
- It provides data security, integrity, and concurrency control.
- It overcomes limitations of traditional file systems.
- RDBMS adds relational modeling and SQL for advanced data management.
- MySQL is one of the most popular and beginner-friendly RDBMS tools.

## 1.3 DBMS Architecture

### Subtopics

- 1-Tier, 2-Tier, and 3-Tier Architecture
- Client-Server Model
- Major Components of DBMS

### Learning Outcome

By the end of this topic, you will be able to:

- Explain the different layers of DBMS architecture.
- Describe how users, applications, and databases interact.
- Identify the key internal components of a DBMS.

## Concept Explanation

A database system involves many activities: entering data, storing it, ensuring security, processing queries, and producing results.



The **DBMS architecture** defines *how these tasks are divided among different layers* so that each layer performs its role efficiently and independently.

Just as an organization has hierarchy (employees → managers → directors), a DBMS uses layers to separate user interaction, application logic, and data storage.

## 1-Tier Architecture

### Description:

- All the database functions—application, DBMS, and database—reside on a single system.
- The user interacts directly with the database through the DBMS interface or command line.
- Suitable for standalone applications or local development.

### Example:

Using **MySQL Workbench** or the MySQL CLI on your personal computer where both client and database server are on the same machine.

### Characteristics:

- Simple and fast for small tasks.
- No network communication.
- Limited scalability and multi-user capability.

## 2-Tier Architecture (Client–Server Model)

### Description:

- The application (client) and database (server) are on different systems.
- The client sends queries to the server, which processes them and sends results back.
- This architecture forms the basis of most web or enterprise database systems.

### Example:

- A Java application connecting to a MySQL server using JDBC.
- The developer machine acts as the client; MySQL server runs on a separate host.

### Flow:

Client Application → DBMS Server → Database Storage

### Advantages:

- Better performance for multiple users.
- Centralized data management.
- Easier to maintain data integrity and security.

## 3-Tier Architecture

### Description:



- Adds a middle layer (application server) between the client and the database server.
- Separates business logic from data storage and presentation.
- Commonly used in large-scale enterprise and web applications.

**Layers:**

1. **Presentation Layer (Client)** – User interface (browser, app, form).
2. **Application Layer (Server)** – Processes business logic and communicates with the DBMS.
3. **Data Layer (Database Server)** – Manages data storage, queries, and transactions.

**Example:**

- A web browser (client) sends a request → Node.js/Java Spring server (application layer) → MySQL database (data layer).

**Advantages:**

- High scalability.
- Better security and fault isolation.
- Easier maintenance and deployment of updates.

## Client–Server Model in Practice

Component	Responsibility
Client	Sends SQL queries and displays results.
Server	Processes the query, retrieves data, and returns output.

**Example Interaction:**

Client: `SELECT * FROM employees WHERE dept='HR';`

Server: Executes query → Sends matching records back.

This separation allows multiple clients to interact with one database server simultaneously.

## Components of DBMS

Component	Function
DBMS Engine	Core service that communicates between data and application.
Query Processor	Translates user queries into low-level instructions.
Storage Manager	Handles data storage, retrieval, and indexing.
Transaction Manager	Maintains ACID properties and concurrency control.
Metadata Catalog	Stores schema details—data about data.



<b>Authorization Manager</b>	Controls user access and permissions.
<b>Utilities</b>	Tools for backup, recovery, and performance tuning.

### Simplified Flow:

User → Query Processor → DBMS Engine → Storage Manager → Database Files

## Advantages of Layered Architecture

- **Modularity:** Each layer handles a specific responsibility.
- **Security:** Sensitive operations remain at lower layers.
- **Maintenance:** Changes in one layer rarely affect others.
- **Scalability:** Supports increasing user load or data volume easily.

## Hands-On Practice (MySQL)

### 1. Check Server Information

```
SELECT VERSION();
```

Displays MySQL server details, verifying the **server layer**.

### 2. Create a New User and Grant Access

```
CREATE USER 'training_user'@'localhost' IDENTIFIED BY 'train@123';  
GRANT ALL PRIVILEGES ON *.* TO 'training_user'@'localhost';  
FLUSH PRIVILEGES;
```

### 3. Demonstrates how **client and server interaction** is controlled in MySQL.

## Checkpoint Review

1. Differentiate between 1-Tier, 2-Tier, and 3-Tier architecture.
2. What are the main responsibilities of the application server in 3-Tier architecture?
3. Name any three core components of DBMS and their roles.
4. Why is client-server architecture preferred in enterprise systems?
5. Write one real-world example where 3-Tier architecture is used.

## Reflection Prompt

Consider an online examination portal. Which architecture (1-Tier, 2-Tier, 3-Tier) would best suit it and why?

## Summary



- **DBMS Architecture** organizes interaction between users, applications, and storage.
- **1-Tier:** All layers on one machine (standalone).
- **2-Tier:** Client–server separation (most enterprise systems).
- **3-Tier:** Adds application logic between client and database for scalability and security.
- Key DBMS components include query processor, storage manager, and transaction manager.
- Proper layering improves performance, maintainability, and reliability.

## 1.4 Data Models

### Subtopics:

- Introduction to Data Models
- Types of Data Models
- Relational Model
- Concepts: Tables, Tuples, Attributes, Domains
- Schema vs Instance

### Learning Outcome

By the end of this topic, you will be able to:

- Explain what a data model is and why it is essential.
- Identify different types of data models.
- Understand the structure of the relational data model.
- Differentiate between schema and instance.

## Concept Explanation

In any database, data must be represented in a structured form so both humans and computers can understand it.

A **data model** provides that structure — it defines **how data is stored, organized, and related**.

Think of a data model as the **blueprint of a building** — before construction starts, you design how each room connects. Similarly, before creating a database, you design how data elements relate to one another.

## Definition

A **data model** is a **conceptual framework** that describes how data is represented, stored, and manipulated within a database system.

In simpler terms, a data model defines:



- What type of data can be stored,
- How data elements are organized,
- How data relates to other data.

## Need for Data Models

1. To provide a clear structure for storing data.
2. To ensure relationships between data are properly defined.
3. To make communication between database designers and developers easier.
4. To act as a bridge between real-world entities and the digital database structure.

## Types of Data Models

Data models have evolved over time as technology and requirements changed.

Model Type	Description	Example / Use Case
<b>Hierarchical Model</b>	Data is organized in a tree-like structure. Each parent can have multiple children, but each child has only one parent.	File systems, XML data.
<b>Network Model</b>	Extends hierarchical model; allows many-to-many relationships using pointers.	Telecom or complex manufacturing systems.
<b>Relational Model</b>	Data is stored in tables (relations) consisting of rows and columns. Relationships are maintained using keys.	MySQL, Oracle, SQL Server.
<b>Entity-Relationship (E-R) Model</b>	Conceptual representation of real-world entities and their relationships.	Database design stage.
<b>Object-Oriented Model</b>	Stores data as objects similar to object-oriented programming.	ObjectDB, db4o.
<b>Document/NoSQL Model</b>	Stores unstructured or semi-structured data as documents.	MongoDB, CouchDB.



Among all these, the **Relational Model** is the most widely used and forms the foundation for systems like **MySQL**.

## The Relational Model

Proposed by **E.F. Codd** in 1970, the relational model organizes data into **relations (tables)**, where each relation consists of rows and columns.

Term	Description	Example
<b>Relation</b>	A table in a database.	student
<b>Tuple</b>	A row in a table (record).	(101, 'Kavya', 'CSE')
<b>Attribute</b>	A column in a table (field).	student_name, department
<b>Domain</b>	The set of allowed values for an attribute.	For gender → {'Male', 'Female'}

### Example Table: Student

Roll_No	Name	Department	Mark
101	Kavya	CSE	88
102	Arjun	ECE	92
103	Divya	IT	79

- Each **row** = one **tuple**.
- Each **column** = one **attribute**.
- The table itself = one **relation**.

## Schema and Instance

Concept	Definition	Example
<b>Schema</b>	The overall design or structure of the database (like a blueprint).	Table name, attributes, data types.
<b>Instance</b>	The actual data present in the database at a given time.	The current rows in the table.

### Analogy:

Schema is like the design of a building, while the instance is the people currently inside it.

## Hands-On Practice (MySQL)

## Create a Database and Table

```
CREATE DATABASE university;
USE university;

CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(30),
    mark INT
);
```

### Insert Data

```
INSERT INTO student VALUES
(101, 'Kavya', 'CSE', 88),
(102, 'Arjun', 'ECE', 92),
(103, 'Divya', 'IT', 79);
```

### View the Table (Instance)

```
SELECT * FROM student;
```

This demonstrates both **schema (structure)** and **instance (data)** in the relational model.

## Advantages of Relational Model

- Simple and easy to understand.
- Data is stored in structured tables.
- Supports SQL for querying and manipulation.
- Ensures data integrity using constraints and keys.
- Provides flexibility in data access using joins and relationships.

## Checkpoint Review

1. Define a data model in your own words.
2. List four types of data models with examples.
3. What is the difference between a schema and an instance?
4. In the relational model, what do tuples and attributes represent?
5. Why is the relational model the most widely used?





## Reflection Prompt

If you were asked to design a small database for a library, how would you model the relationships between books, authors, and borrowers?

## Summary

- A **data model** defines the logical structure of a database.
- **Hierarchical, Network, Relational, and Object-oriented** are common data models.
- The **Relational Model** stores data in tables made up of rows and columns.
- **Schema** defines structure; **Instance** is the data at a particular time.
- MySQL and most modern DBMS systems are based on the **relational model**.

## 1.5 Keys in DBMS

### Subtopics

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

### Learning Outcome

By the end of this topic, you will be able to:

- Explain what keys are and why they are essential in relational databases.
- Identify different types of keys and their purposes.
- Design tables using proper keys to maintain data integrity.

## Concept Explanation

In a database, *keys* are attributes or combinations of attributes that help uniquely identify records and establish relationships between tables.

They form the backbone of relational design by ensuring that each row in a table can be accessed, related, or referenced accurately.

Without keys, duplication, inconsistency, and loss of data integrity would become inevitable.

## 1. Super Key

**Definition:**

A **super key** is any combination of attributes that can uniquely identify a record in a table.

**Example:**

Consider a table STUDENT(roll\_no, name, email, phone)

roll_no	name	email	phone
101	Kavya	<a href="mailto:kavya@gmail.com">kavya@gmail.com</a>	9991112222
102	Arjun	<a href="mailto:arjun@gmail.com">arjun@gmail.com</a>	8883334444

Possible super keys:

- {roll\_no}
- {email}
- {phone}
- {roll\_no, email}

All these uniquely identify a student.

**Note:** A super key may contain extra attributes that are not necessary for uniqueness.

## 2. Candidate Key

**Definition:**

A **candidate key** is the minimal super key—i.e., a super key with no unnecessary attributes.

Each candidate key is a potential choice for the primary key.

**Example:**

From the previous table:

- {roll\_no}, {email}, and {phone} are all candidate keys.
- Each one uniquely identifies a record and cannot be reduced further.

## 3. Primary Key

**Definition:**

A **primary key** is the candidate key chosen by the database designer to uniquely identify each record in a table.

A table can have only one primary key.

**Characteristics:**

- Must contain unique values.
- Cannot have NULL values.
- Automatically indexed by most RDBMS.

**Example:**



```
CREATE TABLE student (
    roll_no INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    phone VARCHAR(15)
);
```

Here, roll\_no is the primary key.

## 4. Alternate Key

### Definition:

All remaining candidate keys that are not chosen as the primary key are called **alternate keys**.

### Example:

If roll\_no is the primary key, then {email} and {phone} become alternate keys.

### Purpose:

They can serve as backup unique identifiers if needed.

## 5. Foreign Key

### Definition:

A **foreign key** is an attribute in one table that refers to the primary key of another table.

It establishes a link between related tables and maintains *referential integrity*.

### Example:

DEPARTMENT table

dept_id	dept_name
D01	HR
D02	IT

EMPLOYEE table

emp_id	emp_name	dept_id
E01	Meera	D01
E02	Raj	D02

Here, dept\_id in **EMPLOYEE** is a *foreign key* referencing dept\_id in **DEPARTMENT**.

### SQL Example:



```
CREATE TABLE department (  
    dept_id CHAR(3) PRIMARY KEY,  
    dept_name VARCHAR(30)  
);  
  
CREATE TABLE employee (  
    emp_id CHAR(3) PRIMARY KEY,  
    emp_name VARCHAR(50),  
    dept_id CHAR(3),  
    FOREIGN KEY (dept_id) REFERENCES department(dept_id)  
);
```

## 6. Composite Key

### Definition:

A **composite key** uses more than one attribute together to uniquely identify a record.

### Example:

In a table COURSE\_REGISTRATION(student\_id, course\_id, semester), neither student\_id nor course\_id alone is unique, but the combination (student\_id, course\_id) is.

### SQL Example:

```
CREATE TABLE course_registration (  
    student_id INT,  
    course_id INT,  
    semester VARCHAR(10),  
    PRIMARY KEY (student_id, course_id)  
);
```

## Importance of Keys

1. Enforce **uniqueness** and prevent duplicate records.
2. Maintain **relationships** between tables.
3. Ensure **data integrity** and **consistency**.
4. Improve **query performance** through indexing.

## Hands-On Practice (MySQL)

### Create Two Tables with Keys

```
CREATE DATABASE key_demo;  
USE key_demo;  
  
CREATE TABLE department (  
    dept_id CHAR(3) PRIMARY KEY,  
    dept_name VARCHAR(30)  
);
```



```
    dept_id CHAR(3) PRIMARY KEY,  
    dept_name VARCHAR(30)  
);  
  
CREATE TABLE employee (  
    emp_id CHAR(3) PRIMARY KEY,  
    emp_name VARCHAR(50),  
    dept_id CHAR(3),  
    FOREIGN KEY (dept_id) REFERENCES department(dept_id)  
);
```

**Insert Data**

```
INSERT INTO department VALUES ('D01','HR'), ('D02','IT');  
INSERT INTO employee VALUES ('E01','Meera','D01'), ('E02','Raj','D02');
```

**View Records**

```
SELECT e.emp_name, d.dept_name  
FROM employee e  
JOIN department d ON e.dept_id = d.dept_id;
```

## Checkpoint Review

1. Define a super key and candidate key.
2. How does a primary key differ from an alternate key?
3. What is the purpose of a foreign key?
4. Write SQL to create a composite key in a table.
5. Why are keys important for referential integrity?

## Reflection Prompt

If a college stores marks of students across multiple courses, how would you design keys to prevent duplication and maintain relationships?

## Summary

- Keys uniquely identify records and link related tables.
- **Super Key:** Any set of attributes identifying a record.
- **Candidate Key:** Minimal super key.
- **Primary Key:** Chosen unique identifier.
- **Alternate Key:** Other candidate keys.
- **Foreign Key:** Refers to a primary key in another table.



- **Composite Key:** Combines multiple fields for uniqueness.
- Proper key design ensures data integrity and supports relational consistency.

## 1.6 Entity Relationship (ER) Model

### Subtopics

- Entities and Attributes
- Relationships and Relationship Types
- Attribute Types
- Relationship Degree and Cardinality
- ER Diagram Notations
- Conversion of ER Diagram to Tables

### Learning Outcome

After completing this topic, you should be able to:

- Explain what entities, attributes, and relationships represent in a database design.
- Interpret an ER diagram and its symbols.
- Describe cardinality and participation constraints.
- Convert an ER model into relational tables.

## Concept Explanation

Before a database is created, its structure must be planned conceptually.

The **Entity-Relationship Model (ER Model)** is a graphical and logical design approach that describes how data objects (entities) relate to each other.

It was introduced by **Peter Chen in 1976** and remains the foundation of relational database design.

## 1. Entity

An **entity** represents a real-world object or concept that can be identified and has data stored about it.

### Examples

- *Student, Course, Department, Employee.*

Entities are usually **nouns** and become tables in the database.

### Types of Entities

Type	Description	Example
------	-------------	---------



<b>Strong Entity</b>	Exists independently of any other entity.	STUDENT, DEPARTMENT
<b>Weak Entity</b>	Depends on another entity for its existence; identified by a partial key.	DEPENDENT (depends on EMPLOYEE)

## 2. Attributes

**Attributes** describe properties or characteristics of an entity.

Each attribute becomes a **column** in a database table.

**Example:**

Entity: STUDENT

Attributes: Roll\_No, Name, Department, Email, Phone.

### Types of Attributes

Type	Description	Example
<b>Simple</b>	Cannot be divided further.	Name, Age
<b>Composite</b>	Can be divided into sub-parts.	Full Name → First Name + Last Name
<b>Derived</b>	Calculated from other attributes.	Age from Date of Birth
<b>Multivalued</b>	Can have multiple values.	Phone Numbers
<b>Key Attribute</b>	Uniquely identifies an entity.	Roll_No

## 3. Relationships

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

**Example:**

A *Student* **enrolls in** a *Course*.

Here, *enrolls in* is the relationship.

### Types of Relationships

Type	Description	Example
<b>One-to-One (1:1)</b>	One instance of Entity A is related to one of Entity B.	Each department has one head.

<b>One-to-Many (1:N)</b>	One instance of Entity A can relate to many of Entity B.	One teacher teaches many students.
<b>Many-to-Many (M:N)</b>	Many instances of A relate to many of B.	Students enroll in many courses.

## 4. Degree of Relationship

The **degree** defines how many entities participate in a relationship:

- **Unary** – Same entity type relates to itself (e.g., *Employee manages Employee*).
- **Binary** – Two different entities (most common).
- **Ternary** – Three entities (e.g., *Doctor–Patient–Medicine*).

## 5. Cardinality and Participation

Term	Meaning	Example
<b>Cardinality</b>	Number of entity instances associated with another entity.	A student → enrolls in multiple courses.
<b>Participation</b>	Whether all or only some entities take part in the relationship.	All departments must have faculty = Total Participation.

## 6. ER Diagram Notations

Symbol	Represents
Rectangle	Entity
Ellipse	Attribute
Diamond	Relationship
Underlined Attribute	Primary Key
Double Ellipse	Multivalued Attribute
Dashed Ellipse	Derived Attribute





Double Rectangle	Weak Entity
Line with Arrow / Crow's Foot	Cardinality (1:1, 1:N, M:N)

## Example ER Diagram Description

Scenario: *A university wants to store information about students and courses.*

### Entities

- STUDENT (Roll\_No, Name, Department)
- COURSE (Course\_ID, Course\_Name, Credits)

### Relationship

- ENROLLS IN (between STUDENT and COURSE)

### Cardinality

- A student can enroll in many courses (1:N).
- A course can be taken by many students (M:N).

Hence, it's a **many-to-many** relationship.

## 7. Converting ER Diagram to Tables

ER Component	Relational Table Representation
<b>Entity</b>	Creates a table with attributes as columns.
<b>Strong Entity</b>	Primary key from its key attribute.
<b>Weak Entity</b>	Includes primary key of owner entity + partial key.
<b>1:1 Relationship</b>	Merge tables or use foreign key.
<b>1:N Relationship</b>	Add foreign key to 'many' side.



<b>M:N Relationship</b>	Create a separate relation (table) for the relationship.
-------------------------	--

### Example Conversion

Entities: STUDENT & COURSE

Relationship: ENROLLS IN (M:N)

### Relational Schema:

```
CREATE TABLE STUDENT (  
    roll_no INT PRIMARY KEY,  
    name VARCHAR(50),  
    department VARCHAR(30)  
);  
  
CREATE TABLE COURSE (  
    course_id INT PRIMARY KEY,  
    course_name VARCHAR(50),  
    credits INT  
);  
  
CREATE TABLE ENROLLMENT (  
    roll_no INT,  
    course_id INT,  
    PRIMARY KEY (roll_no, course_id),  
    FOREIGN KEY (roll_no) REFERENCES STUDENT(roll_no),  
    FOREIGN KEY (course_id) REFERENCES COURSE(course_id)  
);
```

## Advantages of ER Model

- Provides a **visual overview** of database design.
- Helps identify entities, attributes, and relationships before implementation.
- Simplifies communication between designers and developers.
- Forms a solid foundation for converting to relational schema.

## Hands-On Practice (MySQL)

### Create Database

```
CREATE DATABASE er_demo;  
USE er_demo;
```

### Implement Example Tables

```
CREATE TABLE student (  
    roll_no INT PRIMARY KEY,
```

```
name VARCHAR(50),
department VARCHAR(30)
);

CREATE TABLE course (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(50)
);

CREATE TABLE enrollment (
    roll_no INT,
    course_id INT,
    PRIMARY KEY (roll_no, course_id),
    FOREIGN KEY (roll_no) REFERENCES student(roll_no),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

### Insert Data and Query

```
INSERT INTO student VALUES (1, 'Arjun', 'CSE'), (2, 'Kavya', 'IT');
INSERT INTO course VALUES (101, 'DBMS'), (102, 'OS');

INSERT INTO enrollment VALUES (1, 101), (1, 102), (2, 101);

SELECT s.name, c.course_name
FROM student s
JOIN enrollment e ON s.roll_no = e.roll_no
JOIN course c ON e.course_id = c.course_id;
```

## Checkpoint Review

1. What is an entity and what are its attributes?
2. Differentiate between 1:1, 1:N, and M:N relationships.
3. What does cardinality represent in ER models?
4. Explain how a many-to-many relationship is converted to tables.
5. Why is the ER model important before database implementation?

## Reflection Prompt

Think of a college event registration system.

List at least three entities and their possible relationships.

## Summary

- The **ER Model** visually represents entities, attributes, and their relationships.



- **Entities** become tables; **attributes** become columns.
- **Relationships** define how data in one entity connects to another.
- **Cardinality** and **participation** set the rules of association.
- ER diagrams are transformed into **relational schemas** for implementation in systems like MySQL.

## 1.7 Normalization

### Subtopics

- Data Redundancy and Anomalies
- Functional Dependency
- 1NF, 2NF, 3NF, BCNF
- Step-by-Step Normalization Examples

### Learning Outcome

After completing this topic, you should be able to:

- Explain what normalization is and why it is necessary.
- Identify data redundancy and anomalies.
- Define functional dependencies.
- Normalize a database up to BCNF through step-by-step examples.

## Concept Explanation

When databases are poorly designed, data is often duplicated or inconsistent, leading to errors and wasted storage.

**Normalization** is the process of organizing data into well-structured tables to **reduce redundancy** and **improve data integrity**.

In simpler terms, normalization ensures that each fact is stored **only once**, and all relationships are properly defined.

## 1. Need for Normalization

### Problems with Unnormalized Data

- **Redundancy:** Data is unnecessarily repeated.
- **Update Anomaly:** Updating one value requires multiple changes.
- **Insertion Anomaly:** Unable to add new data due to missing related data.
- **Deletion Anomaly:** Deleting a record may remove valuable related data.

### Example – Unnormalized Table



Student_ID	Student_Name	Course	Instructor
101	Arjun	DBMS	Ravi
101	Arjun	OS	Meera
102	Kavya	DBMS	Ravi

#### Issues:

- Student details are repeated (redundancy).
- If Instructor Ravi leaves, we lose information about the course (deletion anomaly).

Normalization eliminates these problems.

## 2. Functional Dependency

A **functional dependency (FD)** exists when one attribute uniquely determines another attribute.

**Notation:**  $A \rightarrow B$  means *A functionally determines B*.

#### Example:

In the table STUDENT(roll\_no, name, department)

$\text{roll\_no} \rightarrow \text{name, department}$

Each roll number determines one student's name and department.

Functional dependencies are the foundation for identifying proper table structures and achieving normalization.

## 3. Normal Forms

Normalization is performed through stages known as **normal forms**, each with stricter rules for structuring data.

### First Normal Form (1NF)

#### Rule:

- Each cell should contain a single (atomic) value.
- There should be no repeating groups or arrays.

#### Example (Before 1NF):

Student_ID	Student_Name	Courses
101	Arjun	DBMS, OS

**After 1NF:**

Student_ID	Student_Name	Course
101	Arjun	DBMS
101	Arjun	OS

**Result:**

Each field now holds one value, ensuring atomicity.

**Second Normal Form (2NF)****Rule:**

- Must be in 1NF.
- All non-key attributes should depend **entirely** on the primary key (no partial dependency).

**Example:****Before 2NF:**

COURSE\_REGISTRATION(Student\_ID, Course\_ID, Student\_Name, Course\_Name)

Here, (Student\_ID, Course\_ID) is the composite key.

But Student\_Name depends only on Student\_ID → **Partial dependency**.

**After 2NF:**

Split into two tables:

STUDENT(Student\_ID, Student\_Name)

COURSE\_REGISTRATION(Student\_ID, Course\_ID, Course\_Name)

Now, every non-key attribute fully depends on the primary key of its table.

**Third Normal Form (3NF)****Rule:**

- Must be in 2NF.
- No **transitive dependency** (non-key attribute depending on another non-key attribute).

**Example:****Before 3NF:**

Student_ID	Student_Name	Department_ID	Department_Name
101	Arjun	D01	CSE
102	Kavya	D02	ECE

Here,

Student\_ID  $\rightarrow$  Department\_ID and Department\_ID  $\rightarrow$  Department\_Name

So, Department\_Name depends on Student\_ID *through* Department\_ID (transitive dependency).

**After 3NF:**

STUDENT(Student\_ID, Student\_Name, Department\_ID)

DEPARTMENT(Department\_ID, Department\_Name)

Now all dependencies are direct.

## Boyce-Codd Normal Form (BCNF)

**Rule:**

- Stronger version of 3NF.
- For every functional dependency ( $A \rightarrow B$ ), A must be a super key.

**Example:**

Professor	Subject	Department
Ravi	DBMS	CS
Meera	OS	CS

**Dependency:** Professor  $\rightarrow$  Subject, but not all professors are unique per department (violation of BCNF).

To fix, we divide:

PROFESSOR(Professor, Department)

TEACHES(Professor, Subject)

## 4. Step-by-Step Example

**Initial Table:**

Roll_No	Name	Department	HOD	Subject
101	Arjun	CSE	Dr. Manoj	DBMS
102	Kavya	ECE	Dr. Meera	OS

**Step 1 – 1NF:**

Split multi-valued attributes (if any).

$\rightarrow$  Already atomic.

**Step 2 – 2NF:**

Check for partial dependencies — none since Roll\_No uniquely identifies everything.

**Step 3 – 3NF:**

HOD depends on Department, not directly on Roll\_No.

→ Create separate tables:

STUDENT(Roll\_No, Name, Department)

DEPARTMENT(Department, HOD)

**Result:**

No redundancy, no anomalies, and clear relationships.

## 5. Advantages of Normalization

- Reduces redundant data storage.
- Ensures data consistency and accuracy.
- Simplifies maintenance and updates.
- Makes relationships clear and logical.
- Improves query performance and reliability.

## 6. When Not to Over-Normalize

- In analytical systems where query speed is critical (e.g., data warehouses).
  - When excessive joins make data retrieval slower.
- Designers must balance between **normalization (integrity)** and **performance (speed)**.

## Hands-On Practice (MySQL)

**Unnormalized Table**

```
CREATE DATABASE normalization_demo;
USE normalization_demo;

CREATE TABLE student_raw (
    student_id INT,
    student_name VARCHAR(50),
    courses VARCHAR(100)
);

INSERT INTO student_raw VALUES (101, 'Arjun', 'DBMS, OS');
SELECT * FROM student_raw;
```



**Normalized Table (1NF)**

```
CREATE TABLE student_1nf (  
    student_id INT,  
    student_name VARCHAR(50),  
    course VARCHAR(50)  
);
```

```
INSERT INTO student_1nf VALUES  
(101, 'Arjun', 'DBMS'),  
(101, 'Arjun', 'OS');  
SELECT * FROM student_1nf;
```

**Further Normalization (3NF)**

```
CREATE TABLE student (  
    student_id INT PRIMARY KEY,  
    student_name VARCHAR(50),  
    dept_id CHAR(3)  
);  
  
CREATE TABLE department (  
    dept_id CHAR(3) PRIMARY KEY,  
    dept_name VARCHAR(30),  
    hod VARCHAR(50)  
);
```

## Checkpoint Review

1. Define normalization in your own words.
2. What are data anomalies, and how does normalization address them?
3. Differentiate between partial and transitive dependency.
4. State the rules of 1NF, 2NF, 3NF, and BCNF.
5. Why might over-normalization cause performance issues?

## Reflection Prompt

If you were designing a database for an online shopping site, how would normalization help in managing customer and order data effectively?



## Summary

- **Normalization** structures data to reduce redundancy and maintain consistency.
- **Functional Dependency** identifies attribute relationships.
- **1NF → 3NF → BCNF** progressively eliminate anomalies.
- Each normal form strengthens integrity and reduces dependency issues.
- A well-normalized database is easier to maintain, extend, and query.