What is Database :

A **Database (DB)** is an organized collection of data that can be easily stored, retrieved, managed, and updated. It efficiently stores large amounts of structured information.

Need for a Database

- **Data Organization** – Stores data in structured formats (tables, rows, and columns).
- **Data Security** – Provides access control and authentication.
- **Data Integrity** – Ensures data accuracy and consistency.
- **Concurrency Control** – Allows multiple users to access data simultaneously.
- **Backup & Recovery** – Prevents data loss due to failures.
- **Query Optimization** – Enables fast search and retrieval using SQL.

Example:

- A bank stores customer details, account transactions, and loan details in a database.
- A website stores user profiles, posts, and comments in a database.

Database Architecture

Database architecture defines how databases are designed and structured.

Types of Database Architectures

1. **1-Tier Architecture (Single Layer)**
   - The database and application run on the same machine.
   - **Example:** A local SQLite database inside an application.
2. **2-Tier Architecture (Client-Server)**
   - The client (frontend) communicates directly with the database server.
   - **Example:** A web application using MySQL.
3. **3-Tier Architecture (Presentation-Application-Database)**
   - Divided into three layers:
     - **Presentation Layer (UI)** – Web browser, mobile application.
     - **Application Layer (Logic)** – Middleware (Java, Python, .NET).
     - **Database Layer** – MySQL, PostgreSQL, MongoDB.
   - **Example:** A large-scale e-commerce website.

ACID Properties (Transaction Management)

ACID ensures data reliability in databases.

Property	Description	Example
Atomicity	A transaction must fully complete or roll back.	A fund transfer of $500 should debit from one account and credit another completely or not happen at all.
Consistency	Ensures the database remains valid after a transaction.	A failed transaction should not leave the database in an inconsistent state.
Isolation	Transactions execute independently without affecting each other.	Two users withdrawing money from the same account should not interfere with each other's transactions.
Durability	Once committed, changes remain permanent even after failures.	Even if the server crashes, the transfer should not disappear.

Difference Between SQL and MySQL

Feature	SQL	MySQL
Definition	SQL (Structured Query Language) is a language used to interact with databases.	MySQL is a database management system (DBMS) that uses SQL to manage and manipulate data.
Type	It is a query language.	It is software (a relational database management system - RDBMS).
Purpose	Used to write queries to create, read, update, and delete data in a database.	Used to store, manage, and retrieve data using SQL queries.
Vendor	SQL is a standardized language used by many DBMSs (e.g., MySQL, PostgreSQL, Oracle, SQL Server).	MySQL is developed and maintained by Oracle Corporation.
Usage	Used in various DBMS like MySQL, PostgreSQL, and Oracle.	Specifically used as a database that follows SQL standards.

🔹 **Example:**

```
SELECT * FROM students;
```

- This query can run in **MySQL**, **PostgreSQL**, or **Oracle** because they all use SQL.

Difference Between DBMS and RDBMS

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Definition	A software system that manages databases.	A type of DBMS that stores data in **tables with relationships**.
Data Storage	Stores data as files or tables (no relation between data).	Stores data in **structured tables** with relations (primary key, foreign key).
Normalization	No concept of normalization.	Supports normalization to reduce redundancy.
Data Integrity	No strict constraints.	Enforces constraints like primary key, foreign key, unique, etc.
Example Systems	Microsoft Access, File System, XML Databases	MySQL, PostgreSQL, Oracle, SQL Server

🔹 **Example:**

- **DBMS**: A simple file-based student database where student records are stored without any relation.
- **RDBMS**: A **Students** table with a **StudentID** column as a **primary key**, linked to a **Marks** table with a **foreign key**.

Conclusion

- **SQL vs. MySQL**: SQL is a **language**, MySQL is a **database system** that uses SQL.
- **DBMS vs. RDBMS**: RDBMS is a more advanced **structured** form of DBMS with **relationships**.

Database Models

Model	Description	Example
Hierarchical DB	Data stored in a tree structure (parent-child relationship).	IBM IMS
Network DB	Multiple relationships between records (graph-like).	CODASYL
Relational DB (RDBMS)	Data stored in tables with relationships.	MySQL, PostgreSQL, Oracle

NoSQL DB	Non-tabular format (Key-Value, Document, Column, Graph).	MongoDB, Cassandra

Most modern databases use **RDBMS** or **NoSQL models**.

Software Installation :

- Download MySQL Community installer.
- Install MySQL server and Workbench.

How to Design SQL Tables

Designing SQL tables is a crucial step in structuring data efficiently. Proper table design ensures **data integrity, performance, and maintainability**.

1. Identify Entities and Relationships

- **Entities** → Real-world objects (e.g., `Users`, `Orders`, `Products`).
- **Attributes** → Properties of an entity (e.g., `name`, `email`, `price`).
- **Relationships** → Define how entities interact (One-to-One, One-to-Many, Many-to-Many).

---

2. Define Columns & Data Types

Each column should have an appropriate **data type** based on the data it stores.

Example Table: `Customers`

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE,
    phone_number VARCHAR(15),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

✅ **Best Practices:**

- Use **INT** for IDs with `AUTO_INCREMENT`.
- Use **VARCHAR** for text instead of CHAR (saves space).
- Use **TIMESTAMP** for date tracking.
- Apply **constraints** (`NOT NULL`, `UNIQUE`, `DEFAULT`) for data integrity.

3. Establish Relationships (Foreign Keys)

Foreign keys (FK) create relationships between tables.

Example: Orders Table (Linked to Customers Table)

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE
);
```

✅ **Best Practices:**

- Use **FOREIGN KEY** for referential integrity.
- Use **ON DELETE CASCADE** to remove associated records automatically.

SQL Commands (DML, DDL, DQL, TCL, DCL)

SQL commands are classified into five categories based on their purpose.

1. DDL (Data Definition Language)

DDL commands define and modify the **structure** of a database (tables, schemas, indexes).

Command	Description	Example
CREATE	Creates a new table, database, or index	`CREATE TABLE employees (...)`
ALTER	Modifies an existing table (add/drop columns)	`ALTER TABLE employees ADD salary DECIMAL(10,2);`
DROP	Deletes a table/database permanently	`DROP TABLE employees;`
TRUNCATE	Deletes all records but keeps the table structure	`TRUNCATE TABLE employees;`

2. DML (Data Manipulation Language)

DML commands modify **data** inside tables (insert, update, delete records).

Command	Description	Example
INSERT	Adds new records	`INSERT INTO employees (name, salary) VALUES ('John Doe', 50000);`
UPDATE	Modifies existing records	`UPDATE employees SET salary = 60000 WHERE name = 'John Doe';`
DELETE	Removes records from a table	`DELETE FROM employees WHERE name = 'John Doe';`

---

3. DQL (Data Query Language)

DQL retrieves **data** from tables.

Command	Description	Example
SELECT	Fetches data from a table	`SELECT * FROM employees;`
WHERE	Filters query results	`SELECT * FROM employees WHERE salary > 50000;`
ORDER BY	Sorts query results	`SELECT * FROM employees ORDER BY salary DESC;`

---

4. TCL (Transaction Control Language)

TCL commands manage **transactions** (ensuring consistency and rollback in case of failure).

Command	Description	Example
COMMIT	Saves changes permanently	`COMMIT;`
ROLLBACK	Reverts to the last committed state	`ROLLBACK;`
SAVEPOINT	Creates checkpoints for rollback	`SAVEPOINT save1;`

Example of Transaction Handling:

```
BEGIN;
UPDATE employees SET salary = 70000 WHERE id = 1;
ROLLBACK;  -- Cancels the update
```

---

5. DCL (Data Control Language)

DCL commands control **user permissions** and access to the database.

Command	Description	Example
GRANT	Gives specific privileges to users	`GRANT SELECT, INSERT ON employees TO user1;`
REVOKE	Removes privileges from users	`REVOKE INSERT ON employees FROM user1;`

Summary Table of SQL Commands

Category	Commands	Purpose
DDL	CREATE, ALTER, DROP, TRUNCATE	Defines and modifies table structures
DML	INSERT, UPDATE, DELETE	Modifies data within tables
DQL	SELECT, WHERE, ORDER BY	Retrieves data from tables
TCL	COMMIT, ROLLBACK, SAVEPOINT	Manages transactions
DCL	GRANT, REVOKE	Controls user permissions

ER Diagram & Database Designing

An **Entity-Relationship (ER) Diagram** is a visual representation of data and its relationships.

Key Components of ER Diagrams

- **Entities (Tables)** – Real-world objects (e.g., Student, Course).
- **Attributes (Columns)** – Properties of entities (e.g., Name, Email).
- **Primary Key (PK)** – Uniquely identifies each record.

- **Foreign Key (FK)** – Creates relationships between tables.
- **Relationships** – One-to-One, One-to-Many, Many-to-Many.

Example:

A **Student-Enrollment-Course** relationship:

```
[Student] ----(Enrolls)----> [Course]
     |                            |
     |                            |
[Student_ID] (PK)           [Course_ID] (PK)
```

ER diagrams help in designing an efficient database structure.

Constraints in Databases

Constraints are rules enforced on database columns to ensure data integrity.

Constraint	Description	Example
PRIMARY KEY	Uniquely identifies a record in a table.	`id` column in a `students` table.
FOREIGN KEY	Links two tables (ensures referential integrity).	`course_id` in an `enrollment` table referencing `course` table.
NOT NULL	Prevents a column from having NULL values.	`name` column in a `students` table.
UNIQUE	Ensures no duplicate values in a column.	`email` column in a `users` table.
CHECK	Defines custom conditions for values.	`age >= 18` in a `students` table.
DEFAULT	Sets a default value if none is provided.	Default `country = 'USA'`.

Example SQL Query:

```
CREATE TABLE students (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    age INT CHECK (age >= 18),
    country VARCHAR(50) DEFAULT 'USA'
);
```

Normalization & Rules

Normal Form	Rule	Example
1NF	Remove duplicate columns & create separate tables.	No multiple values in a single column.
2NF	Ensure every non-key attribute depends on the whole primary key.	Splitting a table into separate ones if needed.
3NF	Remove transitive dependencies (no indirect relationships).	Ensure attributes depend only on the primary key.

The goal of normalization is to prevent duplicate data and ensure consistency.

Denormalization

Denormalization reduces complex joins by combining tables to improve read performance.

When to Use Denormalization

- When querying speed is more important than storage efficiency.
- Used in big data and data warehouses for fast aggregation.

Example:

Instead of separate **Orders** and **Customers** tables, storing customer details inside Orders.

ALTER Command with ADD in SQL

The `ALTER` command is used to **modify an existing table**. Using `ADD`, you can add new **columns** or **constraints** to a table.

1. Add a New Column

Syntax:

```
ALTER TABLE table_name ADD column_name data_type;
```

Example:

```
ALTER TABLE students ADD age INT;
```

👉 This adds a new column age of type `INT` to the `students` table.

---

2. Add Multiple Columns

Syntax:

```
ALTER TABLE table_name

ADD column1 data_type,

ADD column2 data_type;
```

Example:

```
ALTER TABLE students

ADD address VARCHAR(255),

ADD phone_number VARCHAR(15);
```

👉 This adds two new columns `address` and `phone_number` to the `students` table.

---

3. Add a Constraint

You can also add constraints like `NOT NULL`, `UNIQUE`, `CHECK`, `DEFAULT`, `PRIMARY KEY`, or `FOREIGN KEY`.

Add NOT NULL Constraint

```
ALTER TABLE students ADD email VARCHAR(100) NOT NULL;
```

👉 Adds an `email` column that **cannot be NULL**.

Add UNIQUE Constraint

```
ALTER TABLE students ADD CONSTRAINT unique_email UNIQUE (email);
```

👉 Ensures that all values in the `email` column are unique.

Add DEFAULT Value

```
ALTER TABLE students ADD status VARCHAR(10) DEFAULT 'Active';
```

👉 The `status` column will have `'Active'` as the default value if no value is provided.

Add a PRIMARY KEY

```
ALTER TABLE students ADD PRIMARY KEY (student_id);
```

👉 Sets `student_id` as the primary key.

Add a FOREIGN KEY

```
ALTER TABLE orders

ADD CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES customers(id);
```

👉 Links `customer_id` in `orders` to `id` in `customers`.

Conclusion

- `ALTER TABLE ... ADD column_name data_type;` → Adds a new column.
- `ALTER TABLE ... ADD CONSTRAINT constraint_name constraint_type (column_name);` → Adds a constraint.
- You can add multiple columns or constraints at once.

JOINS in MySQL

Joins in MySQL are used to **combine data** from multiple tables based on a related column.

🔹 Types of Joins in MySQL

✅ 1. INNER JOIN

✔ Returns **only matching records** from both tables.
✔ **Excludes unmatched rows.**

Example:

```
SELECT employees.id, employees.name, departments.dept_name

FROM employees

INNER JOIN departments ON employees.dept_id = departments.id;
```

✅ **Only employees with a matching `dept_id` in the `departments` table will be shown.**

✅ 2. LEFT JOIN (LEFT OUTER JOIN)

✔ Returns **all records from the left table** and **matching records from the right table**.
✔ If no match is found, NULL values are returned for the right table's columns.

Example:

```
SELECT employees.id, employees.name, departments.dept_name

FROM employees

LEFT JOIN departments ON employees.dept_id = departments.id;
```

✅ **All employees are shown, even if they don't have a department (NULL values for unmatched rows).**

✅ 3. RIGHT JOIN (RIGHT OUTER JOIN)

✔ Returns **all records from the right table** and **matching records from the left table**.
✔ If no match is found, NULL values are returned for the left table's columns.

Example:

```
SELECT employees.id, employees.name, departments.dept_name

FROM employees

RIGHT JOIN departments ON employees.dept_id = departments.id;
```

✅ **All departments are shown, even if they have no employees (NULL values for unmatched rows).**

✅ 4. CROSS JOIN

✔ Returns **the Cartesian product** of both tables.
✔ Every row from the first table is **combined with every row** from the second table.
✔ **No condition is needed.**

Example:

```
SELECT employees.name, departments.dept_name

FROM employees

CROSS JOIN departments;
```

✅ **If `employees` has 5 rows and `departments` has 3 rows, the result will have `5 × 3 = 15` rows.**

✅ 5. SELF JOIN

✔ A table joins **with itself** to compare rows.
✔ Usually used for **hierarchical data** (e.g., employees and managers).

Example:

```
SELECT e1.name AS Employee, e2.name AS Manager

FROM employees e1

LEFT JOIN employees e2 ON e1.manager_id = e2.id;
```

✅ **Shows each employee with their manager's name.**

🔹 Summary Table

Join Type	Returns
INNER JOIN	Only matching rows from both tables
LEFT JOIN	All rows from the left table + matching rows from the right table (NULL for no match)

RIGHT JOIN	All rows from the right table + matching rows from the left table (NULL for no match)
CROSS JOIN	Every row from the left table combined with every row from the right table
SELF JOIN	A table joins with itself to compare rows

Views in MySQL

A **view** in MySQL is a **virtual table** based on the result of an SQL query. It **does not store data physically** but provides a way to access data from one or more tables.

🔹 Why Use Views?

✔ **Security** – Restricts access to specific columns/rows of a table.
✔ **Simplifies Queries** – Stores complex queries for easy reuse.
✔ **Data Abstraction** – Hides unnecessary details from users.
✔ **Consistency** – Ensures a consistent way to query data.

🔹 Types of Views in MySQL

1️⃣Simple View

- Based on a single table.
- Does **not** contain functions, joins, or groupings.
- Can be used to restrict access to specific columns.

Example:

```
CREATE VIEW student_view AS

SELECT id, name FROM students;
```

✔ This **hides** other columns of the `students` table.

2️⃣ Complex View

- Based on **multiple tables** using joins.
- Can include **aggregations (SUM, COUNT, etc.)**.
- Cannot be updated if it includes aggregations.

Example:

```
CREATE VIEW order_summary AS

SELECT customers.name, orders.total_amount

FROM customers

JOIN orders ON customers.id = orders.customer_id;
```

✔ This retrieves customer names along with their total order amount.

3️⃣ Updatable View

- Allows `INSERT`, `UPDATE`, and `DELETE` if:
  ✅ The view is based on a **single table**.
  ✅ It **does not use** `DISTINCT`, `GROUP BY`, `HAVING`, or `JOIN`.

Example:

```
CREATE VIEW updatable_view AS

SELECT id, name, age FROM employees;



UPDATE updatable_view SET age = 30 WHERE id = 1;
```

✔ Changes in the view reflect in the original table.

4️⃣ Read-Only View

- Contains `JOIN`, `GROUP BY`, or `DISTINCT`.

- **Cannot be modified** (no INSERT, UPDATE, DELETE).

Example:

```
CREATE VIEW total_sales AS

SELECT category, SUM(price) AS total

FROM products

GROUP BY category;
```

✔ This **cannot be updated** since it uses SUM and GROUP BY.

5️⃣ Inline View (Subquery View)

- A **subquery inside the FROM clause** that behaves like a temporary view.
- Used in complex queries but **not stored permanently**.

Example:

```
SELECT avg_salary FROM (SELECT AVG(salary) AS avg_salary FROM employees) AS temp;
```

✔ Used for **temporary calculations**.

🔹 How to Manage Views

✅ Modify a View

```
CREATE OR REPLACE VIEW student_view AS

SELECT id, name, age FROM students;
```

✔ **Updates the view** without dropping it.

✅ Delete a View

```
DROP VIEW student_view;
```

✔ **Deletes the view** but **does not delete data** from the original table.

🔹 Summary

Type	Can Modify Data?	Based on
Simple View	✅ Yes	One Table
Complex View	❌ No	Multiple Tables
Updatable View	✅ Yes	Single Table (No Aggregates)
Read-Only View	❌ No	Uses Aggregation or Joins
Inline View	✅ Temporary	Subquery