Day 5: Database Fundamentals & MySQL

1. Relational Database Management System (RDBMS)

Definition
An RDBMS stores data in tables (rows and columns) and manages relationships between tables using keys. It enforces data integrity and lets you query and manipulate data using SQL.

Core concepts

- Table: collection of rows (records).
- Row / Record: one entry in a table.
- Column / Field / Attribute: named property of a record.
- Primary Key (PK): unique identifier for a row (e.g., id).
- Foreign Key (FK): column that references a PK in another table; enforces referential integrity.
- Schema: structure/definition of the database (tables, columns, constraints).
- Index: data structure that speeds up query lookup.
- Constraints: rules that data must follow (NOT NULL, UNIQUE, CHECK, FK).

Why use an RDBMS?

- Structured storage, ACID transactions, predictable querying (SQL), concurrency control, backup/restore, access control.

Example (conceptual):

Users(id PK, name, email UNIQUE)
Wallets(id PK, user_id FK → Users.id, balance)
Transactions(id PK, wallet_id FK → Wallets.id, type, amount, date)

2. Database Normalization (1NF, 2NF, 3NF) — detailed

Goal of normalization: reduce redundancy, avoid update anomalies, and keep data logically organized.

First Normal Form (1NF)

- **Rule**: Each column contains *atomic* values (no repeating groups or arrays). Each row-column intersection contains a single value.

Violation example:

Contacts
id | name | phones
1 | Alice | 111-111,222-222 <-- phones is multi-valued (not 1NF)
Fix: Normalize to rows per phone:
Contacts (id, name)
ContactPhones (contact_id, phone)

Second Normal Form (2NF)

- **Precondition**: table must be in 1NF.
- **Rule**: No partial dependency of any column on part of a composite primary key. Every non-key attribute must depend on the *whole* primary key.
- **When it matters**: composite PKs (e.g., (order_id, product_id)).
- **Violation example**:
  Table OrdersProducts(order_id PK part, product_id PK part, product_name, order_qty)
    - product_name depends only on product_id, not on the whole composite PK → move product_name to Products table.

Third Normal Form (3NF)

- **Precondition**: table must be in 2NF.
- **Rule**: No transitive dependencies — non-key attributes should not depend on other non-key attributes.

Violation example:
Employees(id PK, name, dept_id, dept_name)

- dept_name depends on dept_id (non-key). Move department info to Departments(dept_id, dept_name) and reference it from Employees.

Practical normalization example (unnormalized → 1NF → 2NF → 3NF)

Unnormalized:

OrderFull(order_id, customer_name, customer_address, product_id, product_name, qty)

1NF: ensure atomic fields (already atomic here).
2NF: separate customer & product data into their own tables:

Customers(customer_id, name, address)

Products(product_id, name)

Orders(order_id, customer_id)

OrderItems(order_id, product_id, qty)

3NF: ensure no transitive attributes remain. (E.g., move any repeated lookup columns to separate tables.)

When to stop normalizing: balance normalization with performance. Normalization reduces redundancy but may require more joins; sometimes denormalization for read performance is acceptable with caching/controlled writes.

ACID Properties in Database Transactions

Transaction: A logical unit of work with one or more SQL operations. Either all succeed (COMMIT) or none (ROLLBACK).

ACID Properties

Property	Meaning	Example
Atomicity	All operations in a transaction are completed or none	Debit from wallet A and credit to wallet B happen together
Consistency	Database moves from one valid state to another; constraints are preserved	Foreign key ensures transactions refer to existing wallets
Isolation	Concurrent transactions do not interfere with each other	Two users transferring money at the same time do not affect each other
Durability	Once committed, changes are permanent	Committed balances remain even after a crash

MySQL Commands for Transactions

START TRANSACTION; -- Begin transaction

COMMIT; -- Apply changes

ROLLBACK; -- Undo changes

SELECT @@transaction_isolation; -- Check isolation level

SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ; -- Change isolation level

Example: Wallet Transfer

START TRANSACTION;

UPDATE Wallets SET balance = balance - 200 WHERE id = 1;

UPDATE Wallets SET balance = balance + 200 WHERE id = 2;

INSERT INTO Transactions (wallet_id, type, amount, description) VALUES

(1, 'DEBIT', 200, 'Transfer to wallet 2'),

(2, 'CREDIT', 200, 'Transfer from wallet 1');

COMMIT;

- Atomicity: Both debit and credit happen together
- Consistency: Balances remain valid
- Isolation: Other transactions wait until this finishes
- Durability: Changes remain after commit

MySQL Installation, Configuration, and Basic Operations

1. Installation

- **macOS:** brew install mysql
  Start server: brew services start mysql

2. Configuration

- Default port: 3306 (may vary, e.g., 3307)
- Secure MySQL: mysql_secure_installation (set root password, remove test DB)
- MySQL config file: my.cnf (mac/linux)

3. Start/Stop/Status

- **macOS:** brew services start mysql / brew services stop mysql
  Connect to MySQL
- mysql -u root -p -P 3306 -h localhost

4. Basic Operations

- **Create Database:**
    - CREATE DATABASE digital_wallet;
    - USE digital_wallet;

- **Show Databases:**
    - SHOW DATABASES;

- **Show Tables:**
    - SHOW TABLES;

- **Describe Table:**
    - DESCRIBE Users;

- **Show Create Table:**
    - SHOW CREATE TABLE Wallets;

- **Check version:**
    - SELECT @@version;

- **Check port:**
    - SHOW VARIABLES LIKE 'port';

5. Storage Engines

- **InnoDB:** Supports transactions, foreign keys (recommended)
- **MyISAM:** No transactions, faster for reads (legacy)
- CREATE TABLE t (...) ENGINE=InnoDB;

6. Backup & Restore

- **Backup:**
    - mysqldump -u root -p digital_wallet > backup.sql

- **Restore:**
    - mysql -u root -p digital_wallet < backup.sql

SQL Fundamentals

SQL commands are categorized based on their purpose:

1. DDL – Data Definition Language

- **Purpose:** Define or modify database structure (tables, schema).

Commands:

Command	Description
CREATE	Create database, table, index
ALTER	Modify table structure (add/drop column)
DROP	Delete database or table
TRUNCATE	Remove all rows from a table (keeps structure)

Example:

```
CREATE TABLE Users (
   id INT PRIMARY KEY,
   name VARCHAR(100),
   email VARCHAR(100)
);
```

2. DML – Data Manipulation Language

- **Purpose:** Manage data inside tables.

Commands:

Command	Description
INSERT	Add new rows
UPDATE	Modify existing rows
DELETE	Remove rows

Example:

INSERT INTO Users (name, email) VALUES ('Alice', 'alice@example.com');

UPDATE Users SET email='alice123@example.com' WHERE id=1;

DELETE FROM Users WHERE id=1;

3. DQL – Data Query Language

- **Purpose:** Retrieve data from tables.
- **Command:** SELECT

Example:

SELECT id, name, email FROM Users;

SELECT * FROM Wallets WHERE balance > 1000;

4. DCL – Data Control Language

- **Purpose:** Control access and permissions.

Commands:

Command	Description
GRANT	Give user privileges
REVOKE	Remove user privileges

Example:

GRANT SELECT, INSERT ON digital_wallet.* TO 'user1'@'localhost';

REVOKE DELETE ON digital_wallet.* FROM 'user1'@'localhost';

Database Design Principles & ER Modeling

1. Database Design Principles

1. **Requirement Analysis**
    - Understand the system, entities, and relationships before designing.
2. **Data Normalization**
    - Organize data to **reduce redundancy** and **maintain integrity**.
    - Normal forms: **1NF, 2NF, 3NF**.
3. **Use of Primary Keys (PK)**
    - Each table should have a **unique identifier** (e.g., id).
4. **Use of Foreign Keys (FK)**
    - Maintain **relationships between tables** (referential integrity).
5. **Consistency & Accuracy**

    - Ensure data **follows rules and constraints**.

6. **Scalability & Performance**
    - Index frequently queried columns.
    - Avoid unnecessary duplication but balance for performance.
7. **Security & Access Control**
    - Restrict who can read/write/update tables (DCL).

2. Entity-Relationship (ER) Modeling

- **Purpose:** Graphically represent the **structure and relationships** of a database.
- **Components:**
    1. **Entity:** Real-world object (e.g., User, Wallet, Transaction)
    2. **Attribute:** Properties of an entity (e.g., name, email, balance)
    3. **Primary Key:** Unique identifier for an entity
    4. **Relationship:** How entities are connected (e.g., User **owns** Wallet)
    5. **Cardinality:** Shows relationship type
        - **One-to-One (1:1)**
        - **One-to-Many (1:N)**
        - **Many-to-Many (M:N)**

3. Example for Digital Wallet

Entity	Attributes	PK	FK
Users	id, name, email, created_at	id	-
Wallets	id, user_id, balance, created_at	id	user_id → Users.id
Transactions	id, wallet_id, type, amount, date, description	id	wallet_id → Wallets.id

Relationships:

- User 1 → N Wallets
- Wallet 1 → N Transactions

Diagram (ERD) Idea:

Users (id, name, email)

|

|1

|---< Wallets (id, user_id, balance)

|

|1

|---< Transactions (id, wallet_id, amount, type)