DBMS [Day 1]

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

Question 1:

A University wants to develop a centralized system to manage students, courses, and faculty data. The application must support thousands of concurrent users across different departments and campuses. The client system should be lightweight, and the database should ensure high consistency, integrity, and easy maintainability. Security and data abstraction are also critical requirements.

Tasks:

1. **Architecture Design:**
   - Identify the most suitable database architecture (single-tier, two-tier, or three-tier) for this system. Justify your answer with clear pros and cons in the context of the given scenario.
2. **Database Modelling:**
   - Using the **ER Model**, design an ER diagram covering:
     - Students (ID, Name, Program, Year)
     - Courses (CourseID, Title, Credits)
     - Faculty (FacultyID, Name, Department)
     - Relationships:
       - Students enroll in courses.
       - Faculty teaches courses.
   - Identify cardinality and participation constraints for each relationship.
3. **Schema Translation:**
   - Convert the ER diagram into a **Relational Model** (set of normalized relations).

Specify all **constraints** (Primary Key, Foreign Key, etc.).

Answer:

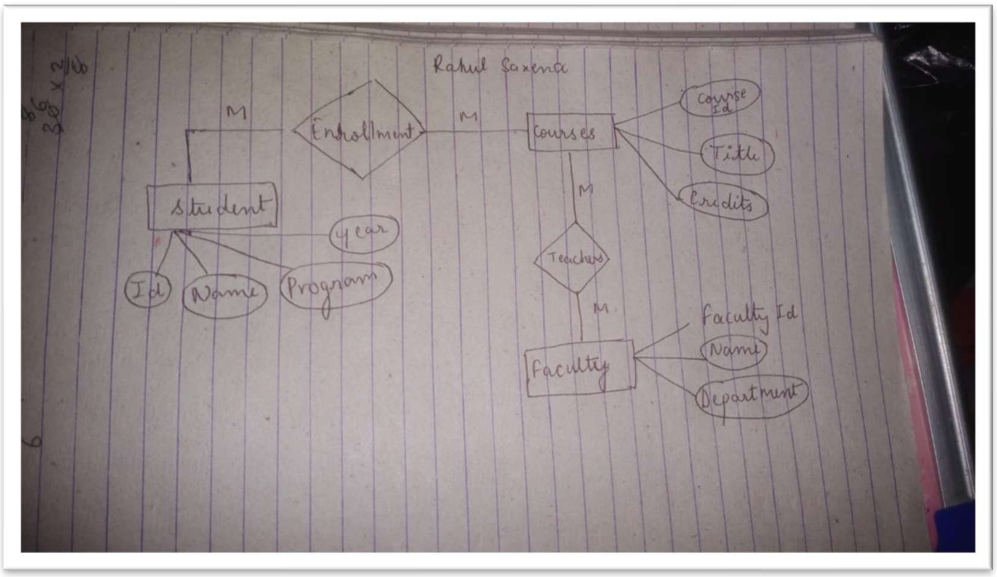
A three-tier architecture is best suited for large-scale, multi-campus university systems requiring scalability, data integrity, and security.

- Pros (Why Three-Tier is Best):
  - Scalability: Can serve thousands of concurrent users across campuses.
  - Security: Sensitive DB access hidden behind the application layer.
  - Maintainability: Business logic can be updated without affecting UI or DB.
  - Data Integrity & Consistency: Centralized control over transactions.

Load Distribution: Each tier can be scaled independently.

- Cons:
  - More complex to implement than single-tier.
  - Requires good infrastructure and network setup.

ER Model:



Relational Model:

```
CREATE TABLE Student (
        ID INT PRIMARY KEY,
        Name VARCHAR(100) NOT NULL,
        Program VARCHAR(50),
        Year INT
    );

CREATE TABLE Course (
        CourseID INT PRIMARY KEY,
        Title VARCHAR(100) NOT NULL,
        Credits INT
    );

CREATE TABLE Faculty (
        FacultyID INT PRIMARY KEY,
        Name VARCHAR(100) NOT NULL,
        Department VARCHAR(50)
    );

CREATE TABLE Enroll (
        StudentID INT,
        CourseID INT,
        PRIMARY KEY (StudentID, CourseID),
        FOREIGN KEY (StudentID) REFERENCES Student(ID),
        FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
    );

CREATE TABLE Teaches (
        FacultyID INT,
        CourseID INT,
        PRIMARY KEY (FacultyID, CourseID),
        FOREIGN KEY (FacultyID) REFERENCES Faculty(FacultyID),
        FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
    );

INSERT INTO Student (ID, Name, Program, Year) VALUES
    (101, 'Rahul Saxena', 'MCA', 3),
    (102, 'Lakshya Singh', 'MCA', 2),
    (103, 'Aman Yadav', 'B. Tech', 4);

INSERT INTO Course (CourseID, Title, Credits) VALUES
    (501, 'Database Systems', 4),
    (502, 'Data Structures', 3),
    (503, 'Advanced Java', 3);

INSERT INTO Faculty (FacultyID, Name, Department) VALUES
    (201, 'Dr. Arun Singh', 'MCA'),
    (202, 'Dr. Gagandeep Kaur', 'MCA'),
    (203, 'Dr. Maajid Bashir', 'B. Tech');

INSERT INTO Enroll (StudentID, CourseID) VALUES
    (101, 501),
    (101, 502),
    (102, 501),
    (103, 503);

INSERT INTO Teaches (FacultyID, CourseID) VALUES
    (201, 501),
    (201, 502),
    (203, 503);
```

Values:

SELECT * FROM Student;

```
+-----+---------------+---------+------+
| ID  | Name          | Program | Year |
+-----+---------------+---------+------+
| 101 | Rahul Saxena  | MCA     |    3 |
| 102 | Lakshya Singh | MCA     |    2 |
| 103 | Aman Yadav    | B. Tech |    4 |
+-----+---------------+---------+------+
```

SELECT * FROM Course;

```
+----------+------------------+---------+
| CourseID | Title            | Credits |
+----------+------------------+---------+
|      501 | Database Systems |       4 |
|      502 | Data Structures  |       3 |
|      503 | Advanced Java    |       3 |
+----------+------------------+---------+
3 rows in set (0.00 sec)
```

SELECT * FROM Faculty;

```
+-----------+--------------------+------------+
| FacultyID | Name               | Department |
+-----------+--------------------+------------+
|       201 | Dr. Arun Singh     | MCA        |
|       202 | Dr. Gagandeep Kaur | MCA        |
|       203 | Dr. Maajid Bashir  | B. Tech    |
+-----------+--------------------+------------+
3 rows in set (0.00 sec)
```

SELECT * FROM Enroll;

```
+-----------+----------+
| StudentID | CourseID |
+-----------+----------+
|       101 |      501 |
|       102 |      501 |
|       101 |      502 |
|       103 |      503 |
+-----------+----------+
4 rows in set (0.00 sec)
```

SELECT * FROM Teaches;

```
+-----------+----------+
| FacultyID | CourseID |
+-----------+----------+
|       201 |      501 |
|       201 |      502 |
|       203 |      503 |
+-----------+----------+
3 rows in set (0.00 sec)
```

Question 2:

An e-commerce startup hires you to create a minimal product and inventory database using **PostgreSQL**. The system must maintain product details, their categories, and inventory status with strict constraints to ensure data integrity.

Tasks:

1. **DDL Implementation:**
   - Write SQL DDL statements to:
     - Create a Category table with fields: category_id, name (unique), and description.
     - Create a Product table with fields: product_id, name, price, stock_quantity, and a foreign key referencing Category.
     - Enforce the following constraints:
       - product_id must be unique and not null.
       - price must be greater than zero.
       - stock_quantity must default to 0 and cannot be negative.
       - name in both tables should not be null.
2. **Table Modification:**
   - Write an ALTER TABLE command to:
     - Add a new column discount to the Product table with a default value of 0 and a CHECK constraint ensuring it's between 0 and 50.
3. **Data Validation:**
   - Discuss how the defined constraints prevent the insertion of invalid data.
   - Explain the difference between TRUNCATE and DROP in the context of resetting the Product table during testing.

Answer:

Create Table

```
CREATE TABLE Category (

       category_id SERIAL PRIMARY KEY,

       name VARCHAR(100) UNIQUE NOT NULL,

       description TEXT

   );

 CREATE TABLE Product (

       product_id SERIAL PRIMARY KEY,

       name VARCHAR(100) NOT NULL,
       price NUMERIC(10, 2) CHECK (price > 0),

       stock_quantity INT DEFAULT 0 CHECK (stock_quantity >= 0),

       category_id INT REFERENCES Category(category_id)

   );
```

Table Modification

```
ALTER TABLE Product

ADD COLUMN discount NUMERIC(5, 2) DEFAULT 0 CHECK (discount >= 0 AND discount <= 50);
```

Data Validation

- NOT NULL: Prevents missing required values (e.g., product name).
- UNIQUE (Category name): Prevents duplicate category entries.
- CHECK (price > 0): Ensures logical consistency; no free or negative-priced items.
- CHECK (stock_quantity >= 0): Avoids storing invalid negative stock.
- Foreign key (category_id): Prevents orphaned products by requiring a valid category.
- CHECK (discount between 0 and 50): Ensures discounts are within business rules.

TRUNCATE vs DROP

Feature	TRUNCATE	DROP
Purpose	Removes all rows from a table	Deletes the entire table structure
Speed	Faster (minimal logging)	N/A (removes table itself)
Table Exists?	Yes (structure retained)	No (table removed completely)
Resets Identity?	Yes, can reset SERIAL with RESTART IDENTITY	N/A
Use Case	Used for testing data reset	Used when table is no longer needed