Real-Life Web Application Simulation: **Library Management System**

Each of you is required to create a dedicated GitHub repository to store all your work and progress for the database project.

Repository Name:

Library Management System - DB Project Part1

What to Include in the Repository:

- Your ERD diagram (image or PDF)
- Relational schema (as text or image)
- SQL scripts (DDL, DML, and DQL)
- Sample data inserts
- SQL Error Log File
- Optional: reflections or notes in a README file

Please ensure your commits are meaningful and consistent, and push your updates regularly. This repository will be used to evaluate your project structure, code organization, and professionalism.

Description of database:

The **Library Management System** is designed to manage books, members, staff, loans, and transactions efficiently. The system includes libraries where each library has a unique ID, name, location, contact number, and established year. Each library must manage books, where each book is identified by a unique ID, ISBN, title, genre, price, availability status, and shelf location. A book belongs to exactly one library, and a library may own many books.

Members can register with personal information such as ID, full name, email, phone number, and membership start date. A member can borrow zero or more books. Each loan links one member with one book and includes loan date, due date, return date, and status.

Each loan may have zero or more fine payments, where a payment is uniquely identified and includes payment date, amount, and method. Payment always corresponds to one specific loan.

Staff work at a specific library, identified by staff ID, full name, position, and contact number. Each library must have at least one staff member, but each member of staff works at only one library.

Members may also review books, where a review includes a rating, comments, and review date. Each review is linked to a specific book and a specific member. A member can provide multiple reviews, and a book may receive many reviews.

Reminder: This is a **solo project**. You're playing every role so own it all and ***learn deeply***.

➢ Objective:

You are now part of a ***development team*** hired to build a full-stack ***Library Management System***. Your team is currently in the ***Backend + Database*** phase. Your responsibility is to ensure ***data integrity***, support ***real-world operations***, and prepare the database to connect to a web-based front-end.

Developer Roles (You're All in One!)

Although this is an **individual project**, you'll simulate working as a full development team by playing multiple roles yourself → yes :) , you are the entire team!

- **Database Engineer**
  You're designing and building the database→ tables, keys, relationships, constraints →the entire data foundation.
- **Backend Developer**
  You write the logic that runs the system: **only using DDL, DML, and DQL**.
  That means creating tables, inserting real data, updating records, deleting safely (or unsafely!), and writing queries that give the frontend what it needs.
- **QA Analyst**
  Your mission: *Break things intentionally!* Delete what shouldn't be deleted, violate constraints, insert invalid data → then log every error in your **SQL Error Log File** just like a tester would in a real company.
- **Business Analyst**
  This role is mine!
  I'll be checking if your system meets the client's needs. So, make sure your database isn't just technically correct, it should make sense in the real world too.

Project Timeline & Tasks

Day 1: System Analysis & Database Design

1. **Draw the ERD**
    - Include entities, attributes, keys, relationships, cardinality, and participation.
    - Use clear notation and include weak entities and M: N relationships.
2. **Map the ERD to Relational Schema**
    - Convert the ERD into relational tables with PKs and FKs defined.
3. **Normalization Practice**
    - Choose 2–3 tables to normalize.
    - Show step-by-step conversion to 1NF → 2NF → 3NF.
    - Justify each normalization step.

Day 2: Database Implementation & Realistic Data

Tasks:

- Use **DDL commands** to create the physical schema.

Write SQL scripts that:

- Create all tables using appropriate data types
- Set IDs as IDENTITY
- Define PKs, FKs (ON DELETE CASCADE, ON UPDATE CASCADE)
- **Apply constraints:**
    - NOT NULL, UNIQUE, CHECK, and DEFAULT
    - Genre values: 'Fiction', 'Non-fiction', 'Reference', 'Children'
    - Loan status: 'Issued', 'Returned', 'Overdue'
    - Prices and amounts > 0
    - Rating between 1 and 5
    - Defaults: IsAvailable = TRUE, LoanStatus = 'Issued', ReviewComments = 'No comments'

- Insert **real-world data**, Minimum required data:
- **2–3 Libraries**
- **10+ Books**
- **6+ Members**
- **8–10 Loans**
- **4+ Payments**
- **4+ Staff**
- **6+ Reviews**
- Use **DML** to simulate real application behavior:
    - Mark books as returned
    - Update loan status
    - Delete reviews/payments

Error-Based Learning (Live Testing Phase)

Act like the app is live and perform actions that simulate user operations and business logic.

Exploration & Failure Testing:

Create a **"SQL Error Log File"** that contains:

- Your attempted action
- The SQL statement
- The error message
- What caused it
- How you resolved it

Encouraged actions: Each action is a **realistic scenario** that could happen in a live web app your job is to test it and observe the error

Try deleting a member who:

- Has existing loans
- Has written book reviews

Try deleting a book that:

- Is currently on loan
- Has multiple reviews attached to it

Try inserting a loan for:

- A member who doesn't exist
- A book that doesn't exist

Try updating a book's genre to:

- A value not included in your allowed genre list (e.g., 'Sci-Fi')

Try inserting a payment with:

- A zero or negative amount
- A missing payment method

Try inserting a review for:

- A book that does not exist
- A member who was never registered

Try updating a foreign key field (like MemberID in Loan) to a value that doesn't exist.

SELECT Queries – Think Like a Frontend API

Imagine the following queries are API endpoints the frontend will call:

- GET /loans/overdue → List all overdue loans with member name, book title, due date
- GET /books/unavailable → List books not available
- GET /members/top-borrowers → Members who borrowed >2 books
- GET /books/:id/ratings → Show average rating per book
- GET /libraries/:id/genres → Count books by genre
- GET /members/inactive → List members with no loans
- GET /payments/summary → Total fine paid per member
- GET /reviews → Reviews with member and book info

Developer Reflection

At the end of the project, reflect on:

- What part was the most difficult?
- Which SQL command (DDL, DML, DQL) did you learn the most from?
- What did you discover from your error logs that made you think like a real developer?