CMSC461: Project

In this project, you will design, implement, and analyze an advanced database system for hospital management. This project can be done either individually or in a team of two. The project can be implemented on your own laptop or on MariaDB server on the VM provided.

I. Schema Definition

1. Patient: Patient_ID (PK), Full_Name, DOB, Gender, Contact_Number, Insurance_Number, Address
2. Doctor: Doctor_ID (PK), Full_Name, Department, Contact_Number, Email
3. Appointment: Appointment_ID (PK), Patient_ID (FK), Doctor_ID (FK), Department, Appointment_Date, Appointment_Start_Time, Status

   Note: Consider each appointment is for 30 minute duration.
4. LabTest: Lab_Order_ID (PK), Patient_ID (FK), Test_Type, Order_Date, Scheduled_Date, Result
5. Prescription: Prescription_ID (PK), Patient_ID (FK), Doctor_ID (FK), Date_Issued, Medication_Name, Dosage, Frequency
6. Billing: Bill_ID (PK), Patient_ID (FK), Service_Type, Service_Date, Amount_Charged, Amount_Paid, Payment_Date, Payment_Status

II. Tasks

1. Database Design and ER Diagram

- Create a detailed ER diagram that includes:
    - Entities with all attributes (marking PKs and FKs).
    - Relationships with names and cardinality:
        - One patient can have many appointments
        - One appointment is with one doctor.

Note: you can use any online tools or draw on paper and submit photo capture.

2. Database Implementation

- Create the hospital database by creating all tables with appropriate constraints. If you are using MariaDB on the server, then use your assigned database (as you are not permitted ot create a new database). The database should be implemented to avoid cascading rollback i.e. it should allow only committed read operation, and no dirty reads should be allowed.
- Insert the data in the project_data.txt file

3. Querying the Database

a. Write a query that, for a given Patient_ID (for example, P1001), retrieves the complete history of appointments. The output should include:

- Appointment_ID
- Appointment_Date
- Appointment_Time
- Department
- Appointment Status
- Doctor’s Full_Name (from the Doctor table)

b. Write a query to identify all patients who have outstanding bills. For each such patient, return:

- Patient_ID
- Full_Name (from the Patient table)
- Total Outstanding Amount (calculated as the sum of the differences between Amount_Charged and Amount_Paid across all billing records with Payment_Status “Pending”).

c. Assume “today” is 2025-06-20. Write a query to list all lab tests that have a result of “Pending” and a Scheduled_Date earlier than 2025-06-20. Return:

- Lab_Order_ID
- Patient_ID
- Test_Type
- Scheduled_Date

d. Write a query that lists all prescriptions issued within a date range between 2025-06-15 and 2025-06-25 and displays:

- Prescription_ID
- Patient_ID
- Doctor's Full_Name
- Medication_Name
- Dosage
- Date_Issued

e. Write a query that returns, for each doctor:

- Doctor_ID
- Full_Name
- Total number of appointments scheduled (regardless of status)
- Number of appointments that were "Completed"

4. <u>Transaction Processing and Concurrency Control</u>

Note: Whenever a transaction updates two or more tables, insert a delay of 2 seconds between these updates to permit for deadlock situation etc.

a. Write program (in your chosen language) for a transaction that takes following input (via command line arguments

   i. Patient id (e.g. P1000)
   ii. Doctor id (e.g. D2001)
   iii. Date and time

b. The program should do the following as a transaction.

- Book an appointment with a doctor for a patient for the given date and time.
- The transaction must first check that there is no existing appointment for the patient at that date/time and that the doctor is available (no other appointment for that doctor overlaps).
- If the checks pass, wait for 5 seconds (a configuration parameter for the program) and then insert the new appointment record and commit the transaction.

- The wait for 5 seconds is to be used for checking that atomicity and isolation are maintained so that no conflicting appointments are made, no two or more concurrent transaction can schedule overlapping appointments

c. Create two concurrent transactions for patient (e.g. P1001) w.r.t. billing
- *Transaction T1*: Updates a billing record for patient by recording a payment of $100 for Bill_ID B6001.
- *Transaction T2*: Generates a billing report that reads the billing records for this patient.

d. Create a scenario where:
- *Transaction T3:* Updates an appointment record for a patient (e.g. P1001) from scheduled to completed and then requests to update the billing record for the same patient.
- *Transaction T4:* Updates a billing record for a patient for a given appointment by setting the payment status to Void and update the appointment record for the patient (e.g. P1001) from scheduled to cancelled.

5. Advanced SQL – Stored Procedures, Functions, and Triggers

a. Create a stored procedure named BookAppointment that accepts the following parameters:
- Patient_ID
- Doctor_ID
- Appointment_Date
- Appointment_Time
- Department

The procedure should:
  - Check for conflicting appointments (for both the patient and the doctor).
  - If no conflict exists, insert a new record into the Appointment table.
  - Return a status code (0 for success, negative for failure).

b. Develop a function named CalculateOutstandingBill that takes a Patient_ID as input and returns the total outstanding amount for that patient (i.e., the sum of all (Amount_Charged – Amount_Paid) for billing records with Payment_Status "Pending").

c. Create a trigger that fires before an INSERT or UPDATE on the Billing table to ensure:
- Neither Amount_Charged nor Amount_Paid can be negative.
- If a negative value is detected, the trigger should prevent the operation (raise an error).

d. Create a trigger on the Appointment table that prevents scheduling an appointment with an Appointment_Date earlier than the current date.

Provide the code for each stored procedure, function, and trigger along with a brief explanation of how they enforce the business rules.

6. <u>Transactional Program for Operations</u>

Develop a command-line (or web-based) program that simulates the following hospital operations in a transactional manner:

a. Book Appointment:
- Prompt the user to enter details: Patient_ID, Doctor_ID, Appointment_Date, Appointment_Time, and Department.
- Call the BookAppointment stored procedure to attempt booking the appointment.
- Display the outcome (success or failure message).

b. Update Lab Test Result:
- Prompt the user for a Lab_Order_ID and the new test Result.
- Update the LabTest record with the new result, ensuring that the update occurs within a transaction.
- If the update fails (e.g., due to a trigger violation), rollback the transaction.

c. Process Billing Payment:
- Prompt the user for a Bill_ID and the payment amount.
- Update the Billing record accordingly (i.e., add the payment to Amount_Paid and update Payment_Status if fully paid).
- Ensure that this operation is atomic and that the changes are not visible to other
- operations until the transaction commits.

d. Generate Patient History Report:

   Allow the user to input a Patient_ID, then retrieve and display a consolidated report that includes:

   - Appointment history (dates, departments, statuses)
   - Lab test history (test types, scheduled dates, results)
   - Prescription history (medication names, dosages, date issued)
   - Billing summary (total outstanding amount, payment history)

Your program should handle errors gracefully. If any operation violates a business rule (e.g., entering a negative payment amount), the program should rollback the transaction and display an appropriate error message.

III. REQUIREMENTS

Readme File:

Your readme should include:

- Team details (Name, UMBC Id of one or both team members)
- A summary of your design decisions and learning outcomes.
- A list of all SQL files, diagrams, and pseudocode documents with descriptions.
- Instructions on how to set up and run your database system (including the database software used, e.g., MySQL, Postgres, or MariaDB).
- Challenges faced and how you addressed them.

Demonstration:

Prepare a short demo (recorded video or live presentation) that:

- Walks through your ER diagram.
- Shows key SQL queries and outputs.
- Explains the transaction and concurrency control scenarios.

III. EVALUATION CRITERIA

Phase 1:

1. **Database Design and ER Diagram: 10 marks**
   Completeness, clarity (including primary/foreign keys and relationship cardinalities), and correctness.

2. **Database Implementation: 10 marks**
   Accuracy in creating the schema, inserting data, and ability to execute basic queries.

Phase 2:

3. **SQL Queries: 10 marks**
   Defining and executing queries.

4. **Transaction and Concurrency Control: 20 marks**
   Correctness of pseudocode, simulation diagrams, and the effectiveness of your concurrency control and deadlock resolution explanations.

Phase 3:

5. **Advanced SQL Features: 15 marks**
   Quality and functionality of stored procedures, functions, and triggers.

6. **Transactional programs for operations: 15 marks**
   Writing and running these transactional programs.

7. **Documentation and Presentation: 20 marks**
   Clarity of your readme and the quality of your demonstration.

8. **← end of project description →**