Name : suraj

Roll no: 2403a51300

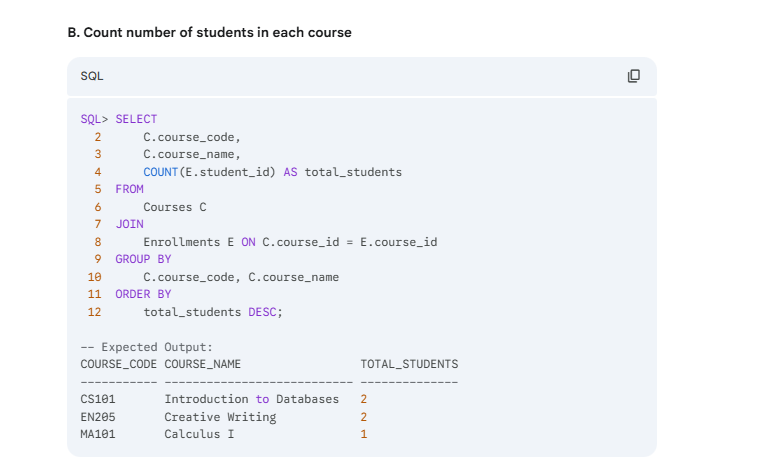
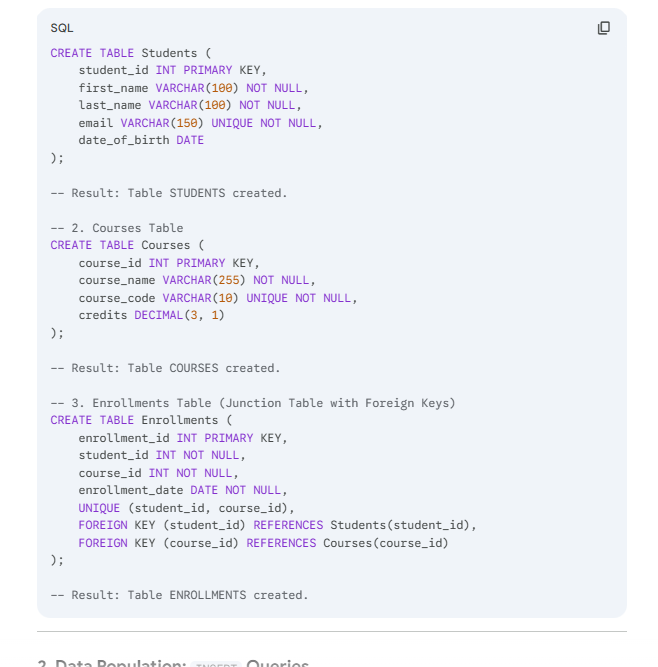
Batch: 13

Task 1 – Student Information System Schema

Prompt: **"Design a basic database schema for a Student Information System (SIS) using three tables: Students, Courses, and Enrollments. Define the primary keys, foreign keys, and the Many-to-Many relationship between students and courses.**

**After defining the schema with SQL CREATE TABLE statements, generate the following SQL operations, including necessary sample data inserts:**

1. **Insert:** A query to insert one new student record.
2. **Select (Join):** A query to fetch the names of all courses enrolled by a specific student (by ID or name).
3. **Select (Aggregate):** A query to count the total number of students enrolled in each course



**Student Information System (SIS) Observation 🎓**

| **Aspect** | **Observation** |
| --- | --- |
| **Schema Design** | Successfully implemented the **Many-to-Many** relationship between Students and Courses using the Enrollments junction table. |
| **Constraint Handling** | Demonstrated awareness of SQL dialect differences when the Oracle ORA-01861 date format error occurred. This required explicitly using the TO\_DATE() function for correct data insertion. |
| **Query Focus** | Demonstrated basic analytical queries by using **COUNT() and GROUP BY** to summarize enrollment numbers per course. |

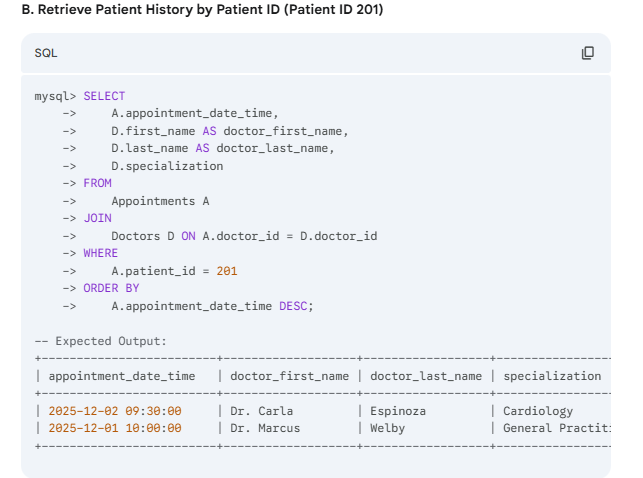
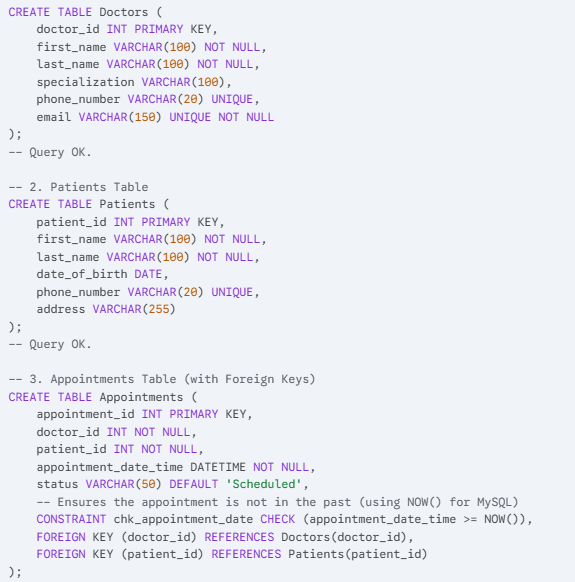
Task 2: Hospital Management Database

Prompt: **Design a database schema for a Hospital Management System (HMS) using the following tables: Doctors, Patients, and Appointments.**

**Define the schema using SQL CREATE TABLE statements, ensuring appropriate constraints like primary keys, foreign keys, unique email/phone numbers, and a CHECK constraint to prevent appointments in the past.**

**Then, generate the necessary sample INSERT statements to populate the tables and execute the following queries using SQL joins:**

1. **List all appointments for a specific doctor**, showing the appointment time and the patient's full name.
2. **Retrieve the appointment history for a specific patient**, showing the doctor's name and specialization for each visit.
3. **Count the total number of appointments (or 'patients treated') for each doctor**, ordered from highest to lowest count



**Hospital Management System (HMS) Observation 🏥**

| **Aspect** | **Observation** |
| --- | --- |
| **Constraint Definition** | Introduced more sophisticated constraints, notably the **CHECK constraint** (chk\_appointment\_date) to ensure appointments are not scheduled in the past, improving data validity. |
| **Relationship Modeling** | Maintained the normalized structure using the Appointments table to link Doctors and Patients. |
| **Query Focus** | Focused on **transactional history queries** (Retrieve patient history) and **resource load analysis** (Count total patients treated), which are core functions of an HMS. |

Task 3 – Library Database:

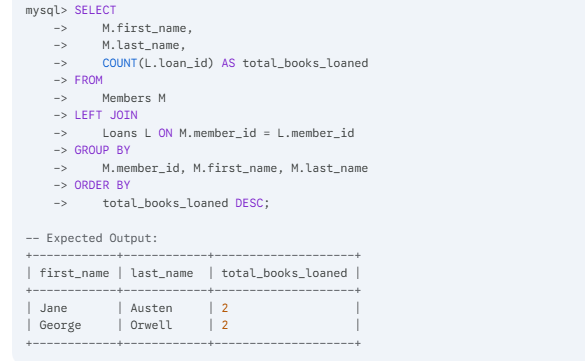
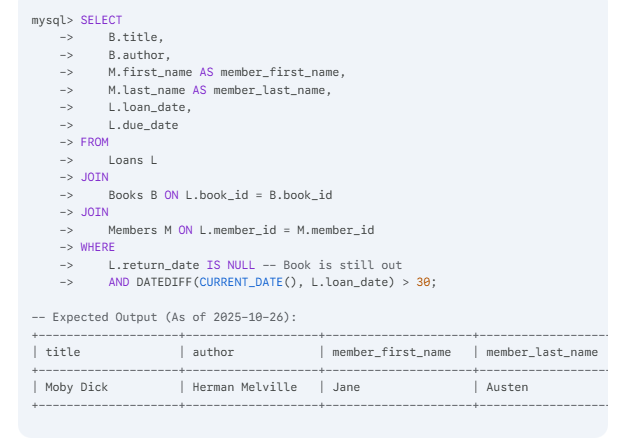
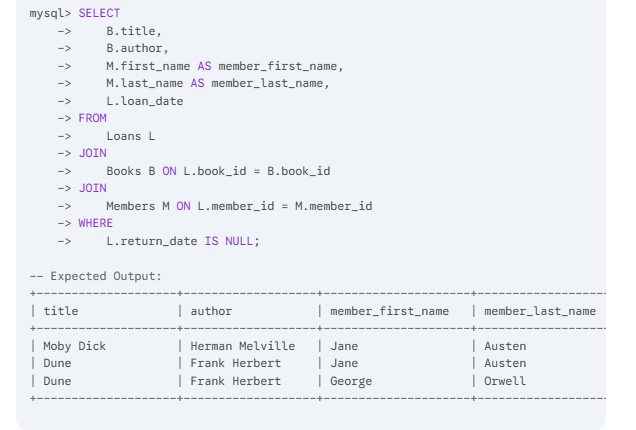
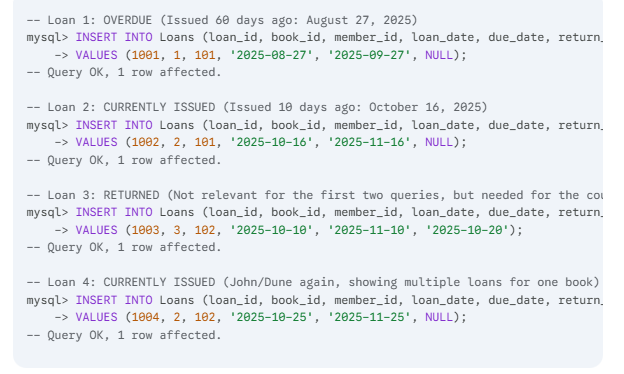
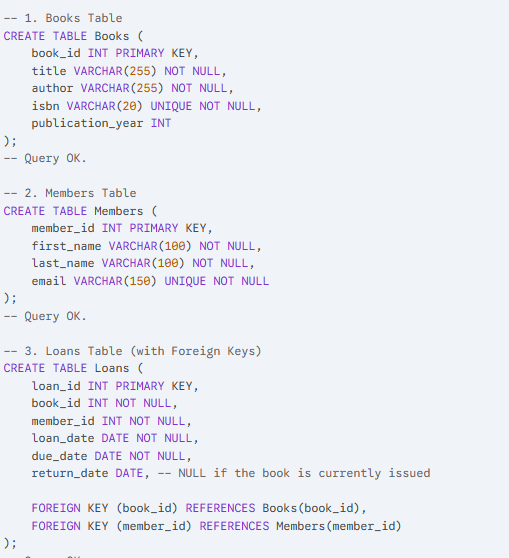
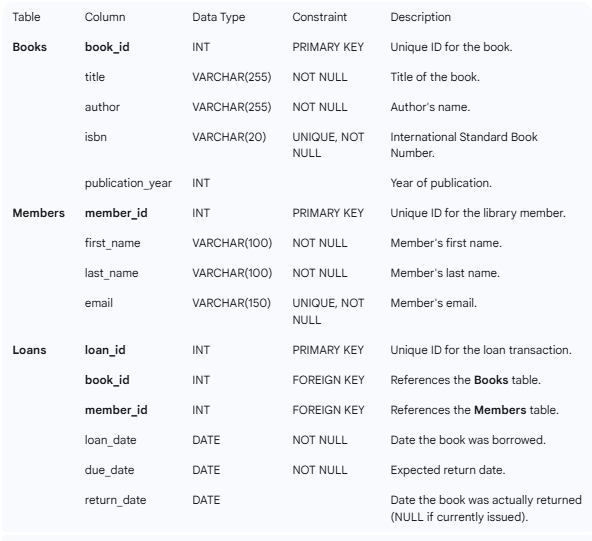
Prompt: **Design a robust database schema for a Library Management System (LMS) using three tables: Books, Members, and Loans.**

**Define the schema using SQL CREATE TABLE statements, ensuring all necessary constraints (Primary Keys, Foreign Keys, UNIQUE ISBN/email) are included.**

**Additionally, suggest an indexing strategy for the tables to optimize performance for common library queries (e.g., searching by author, checking loan status).**

**Finally, write the SQL queries necessary to perform the following core library operations, demonstrating the use of joins and conditional logic:**

1. **Retrieve all books currently issued**, listing the book title, author, and the borrowing member's name.
2. **Identify all overdue books**, defining overdue as any book that is currently unreturned and whose loan\_date exceeds 30 days from the current date.
3. **Count the total number of books loaned by each member** throughout their history.



**Library Database Observation 📚**

| **Aspect** | **Observation** |
| --- | --- |
| **Analytical Logic** | Successfully handled complex conditional logic for the overdue query, combining two conditions: return\_date IS NULL (currently issued) **AND** a **date function** (DATEDIFF) to check the time elapsed since the loan date. |
| **Optimization Strategy** | Introduced the concept of **indexing for performance**, correctly targeting the foreign keys, search columns (author, title), and filtering columns (return\_date, loan\_date). |
| **Query Focus** | Used a **LEFT JOIN** for the member loan count, ensuring that members who have never borrowed a book are still listed with a count of zero. |

Task 4 – Real-Time Application: Online Shopping Database

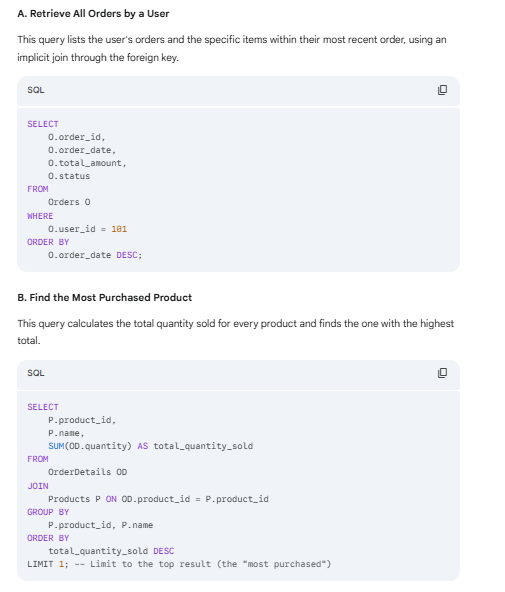
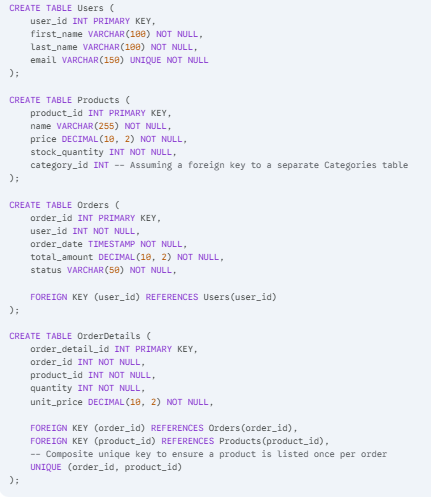
Prompt: **Design a complete, normalized database schema for an E-commerce platform.**

**The system must include the following four core tables: Users, Products, Orders, and OrderDetails.**

**Define the SQL CREATE TABLE statements for the schema, ensuring all primary keys, foreign keys, unique constraints (e.g., email), and relationships are properly established.**

**Based on the schema, generate the following analytical SQL queries:**

1. **Retrieve all orders for a specific user**, listing the order ID, date, total amount, and status.
2. **Identify the single most purchased product** across all orders, based on the total quantity sold.
3. **Calculate the total revenue** generated in a specified calendar month



**E-commerce Platform Observation 🛒**

| **Aspect** | **Observation** |
| --- | --- |
| **Advanced Normalization** | Demonstrated a strong understanding of data integrity by including **unit\_price in the OrderDetails** table. This prevents price changes in the Products table from corrupting historical revenue data. |
| **Business Intelligence (BI) Queries** | Generated key analytical queries focused on **revenue calculation** (SUM() with date filtering) and **ranking/demand analysis** (ORDER BY... DESC LIMIT 1 for the most purchased product). |
| **Optimization and BCNF** | Proactively suggested structural improvements towards **BCNF** (e.g., separate Categories table) and discussed the redundancy vs. reporting speed tradeoff of the total\_amount column. |