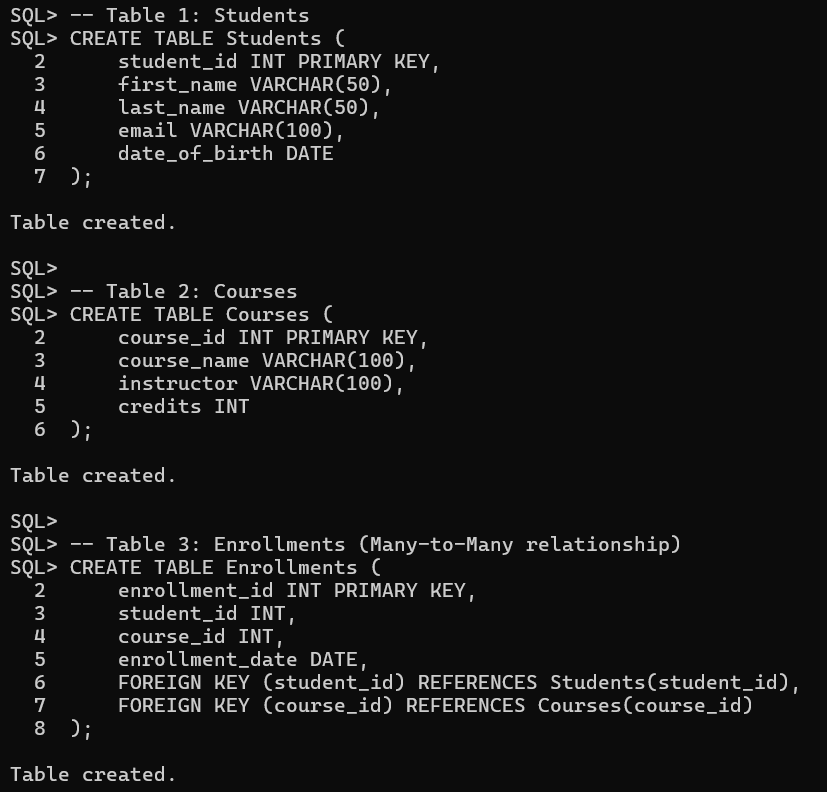
**AI LAB – 16.1**

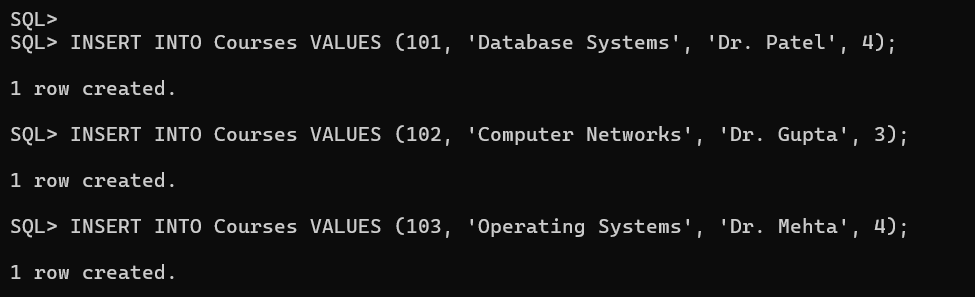
Name: Ananthula Avinash

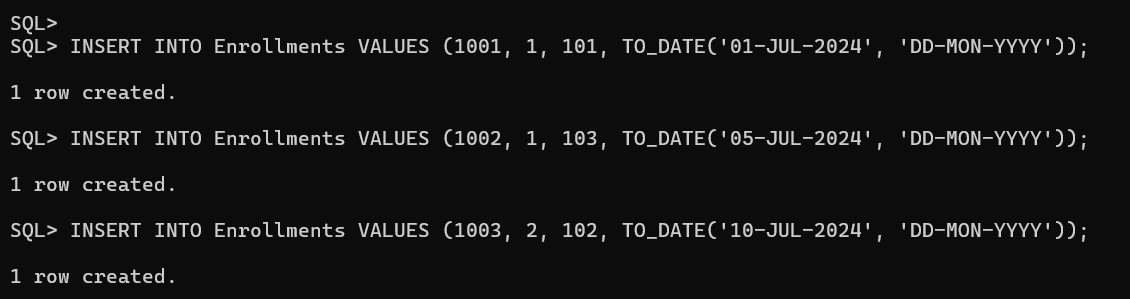
Batch: 14

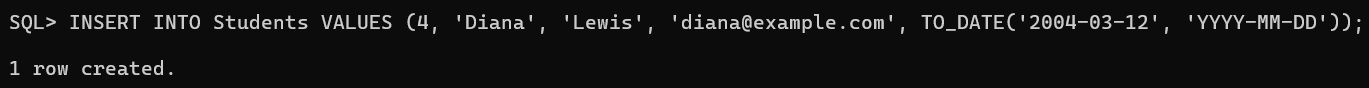
**Task 1 – Student Information System Schema**Design a database schema for a Student Information System and generate queries using AI.

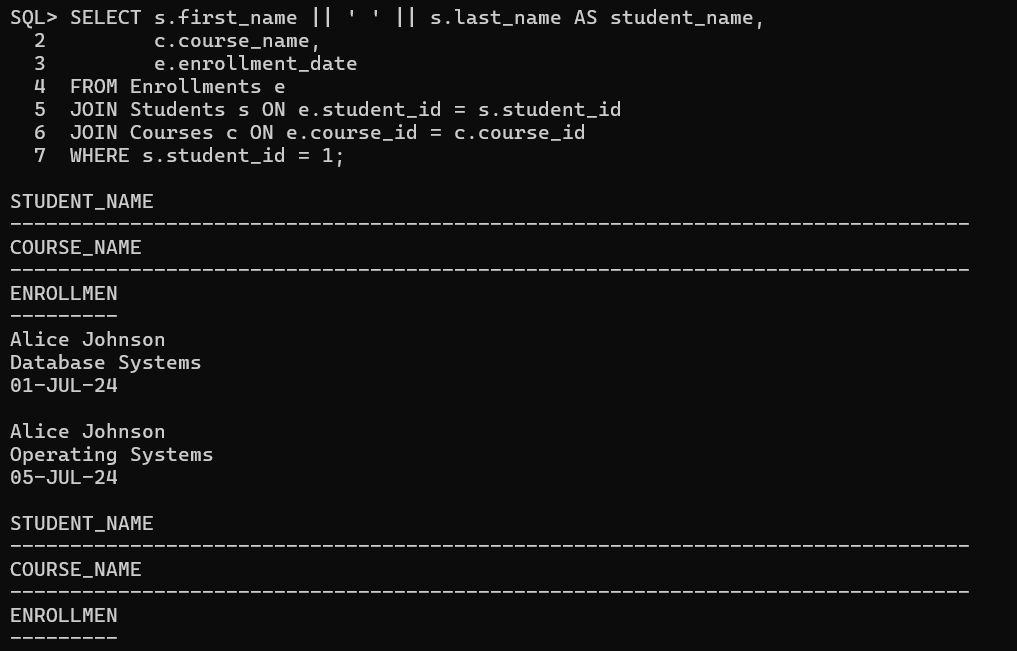


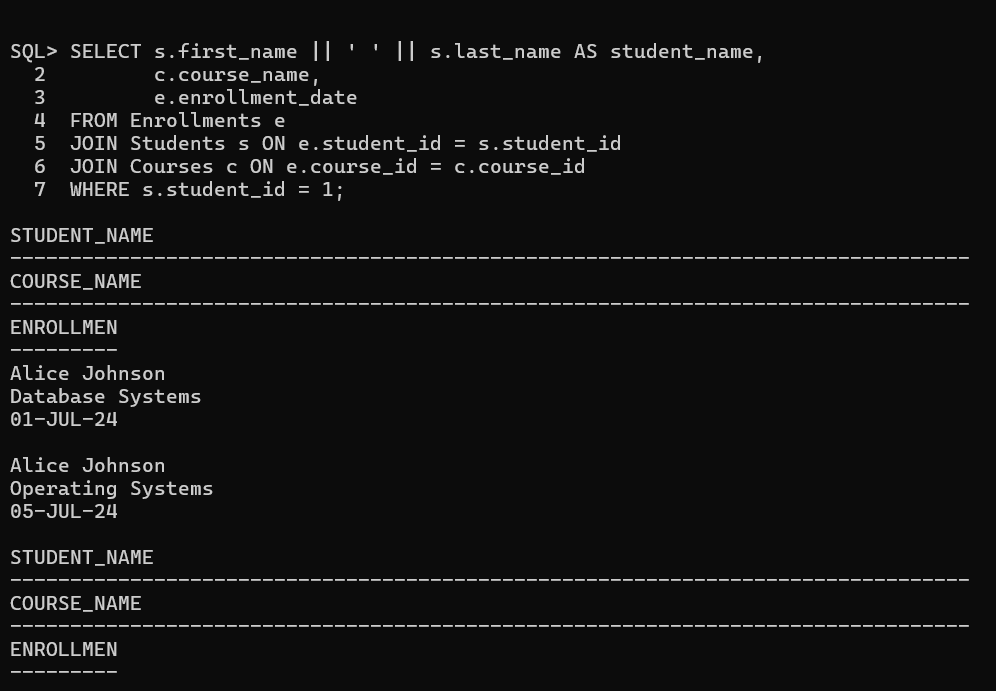


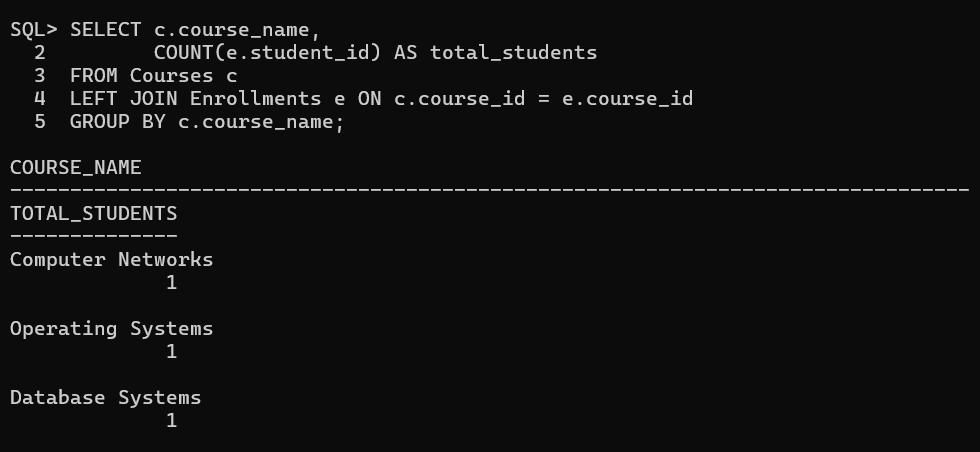








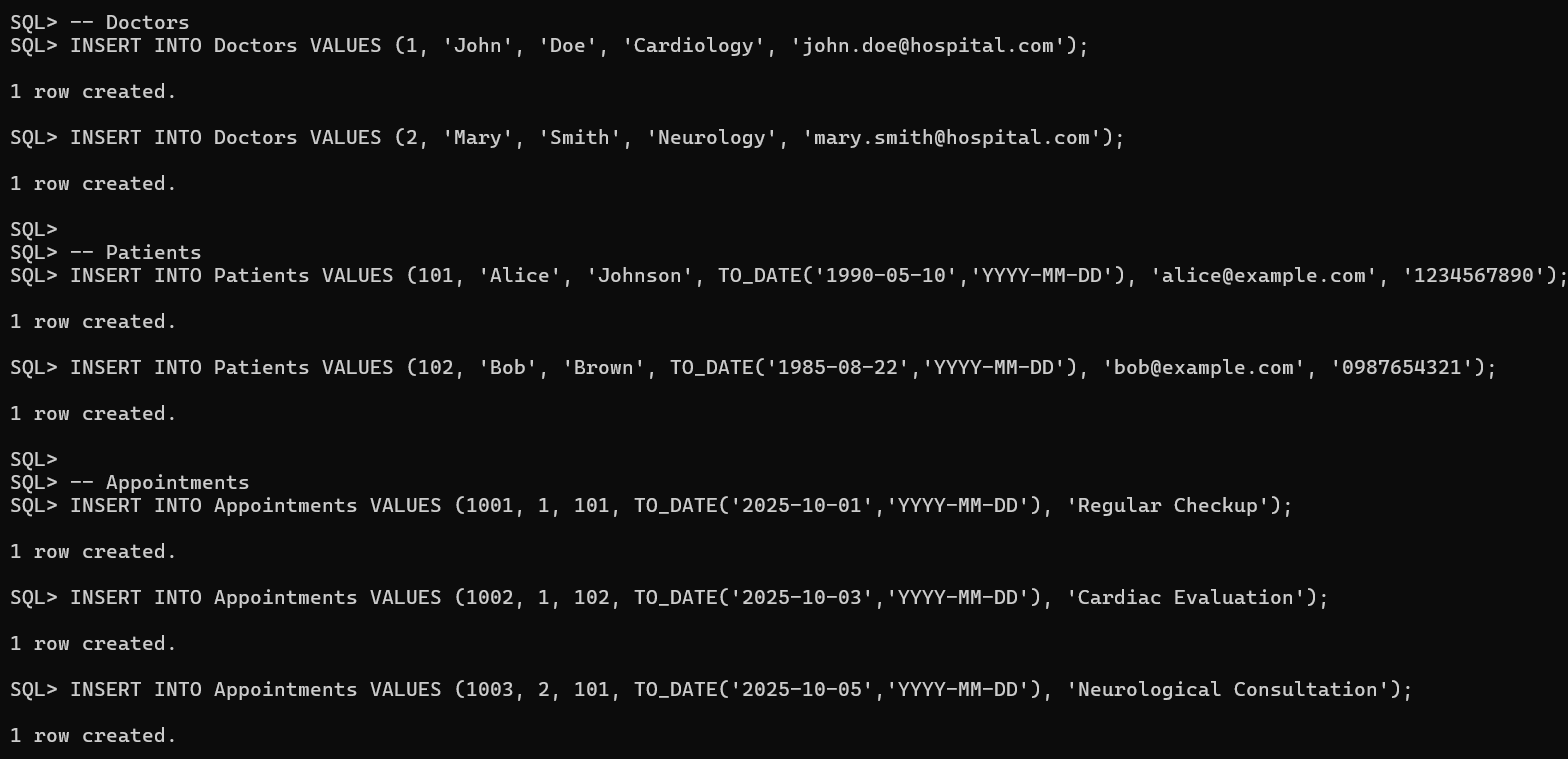
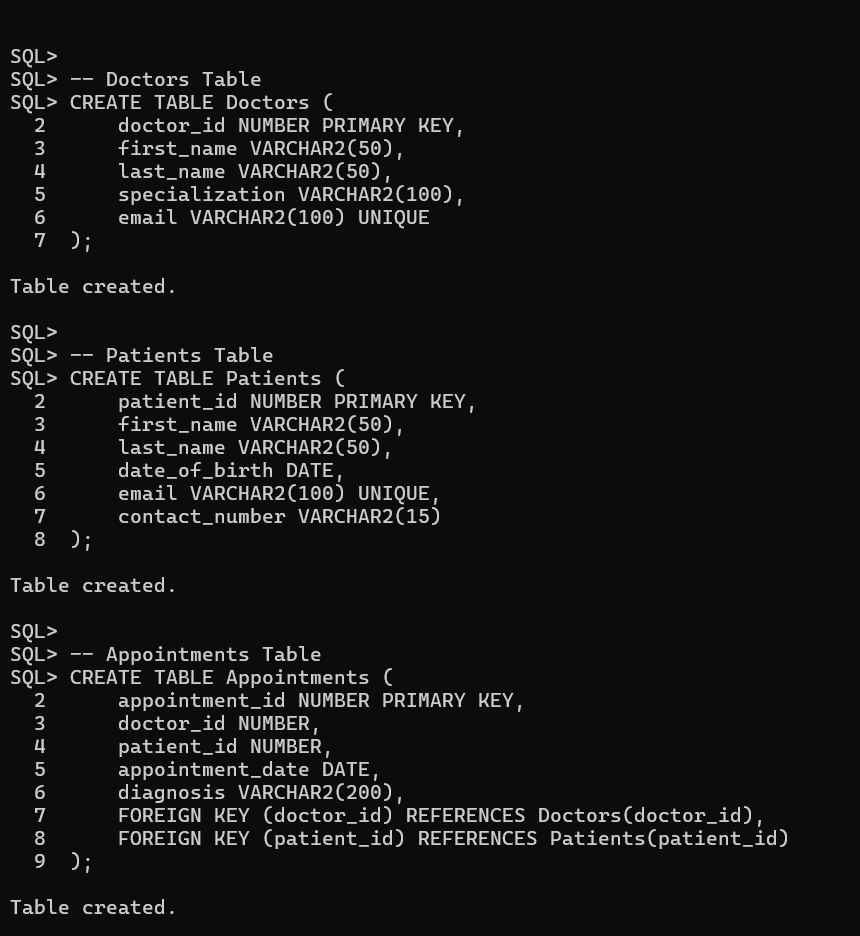


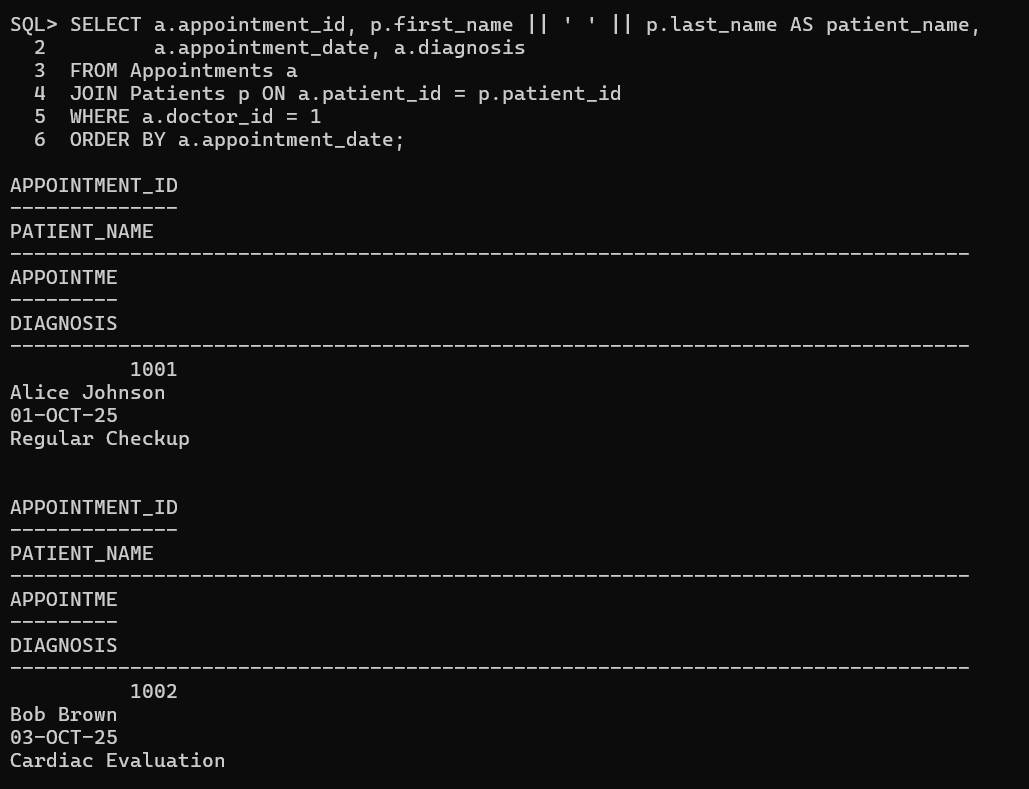


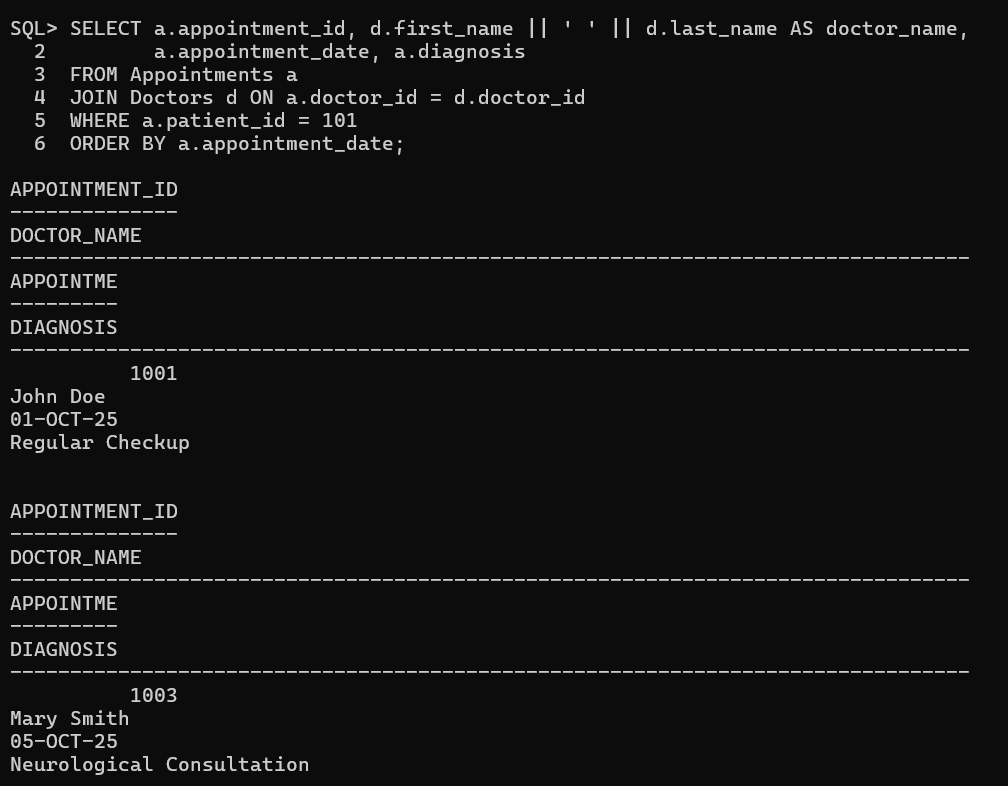
**Observation:**

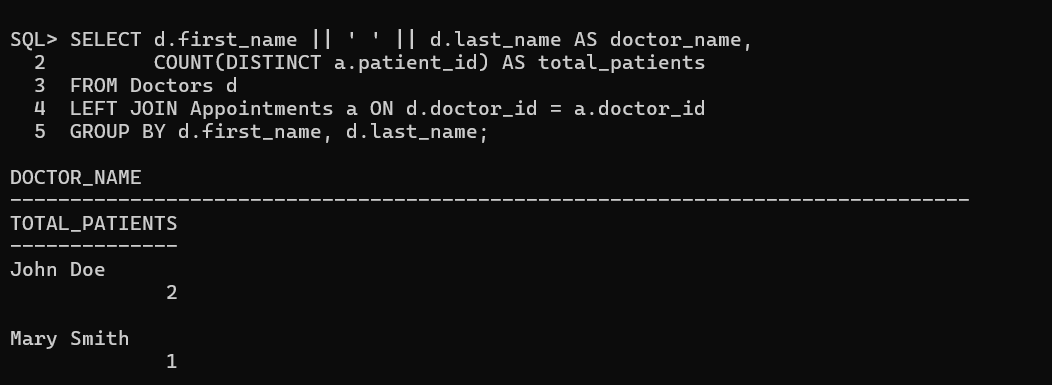
In this Student Information System, the database schema is designed with three tables: Students, Courses, and Enrollments, implementing a many-to-many relationship between students and courses through the junction table Enrollments. The DATE data type is used for date\_of\_birth and enrollment\_date, ensuring accurate storage, comparison, and calculation of dates. Primary keys (student\_id, course\_id, enrollment\_id) guarantee uniqueness, while foreign keys enforce referential integrity, preventing orphaned records. Using TO\_DATE() with a specified format avoids errors like ORA-01861 and ensures consistent date entry. Queries for fetching all courses enrolled by a student and counting the number of students per course work efficiently due to properly normalized tables and key constraints. Overall, the schema supports scalability, data integrity, and accurate query results, demonstrating good database design principles.

**Task 2** – Hospital Management Database  
Create schema and queries for a Hospital Management System.





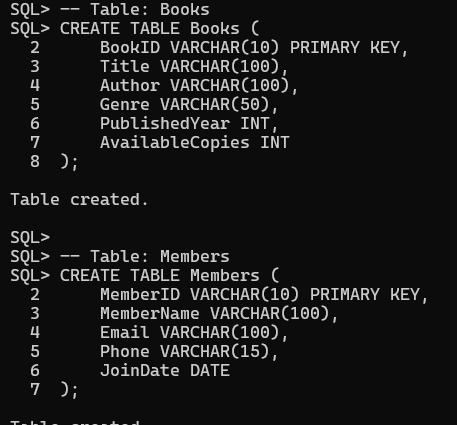


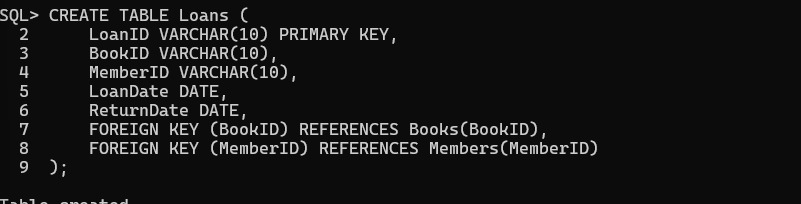


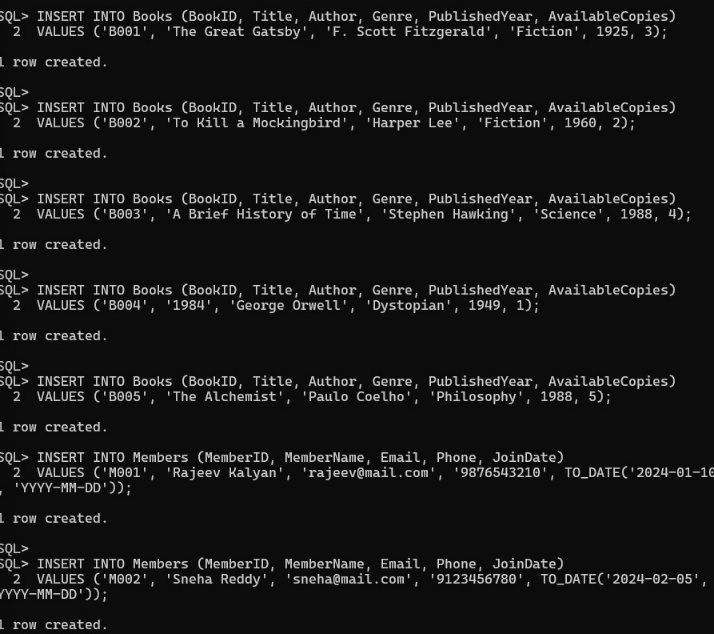
**Observation:**

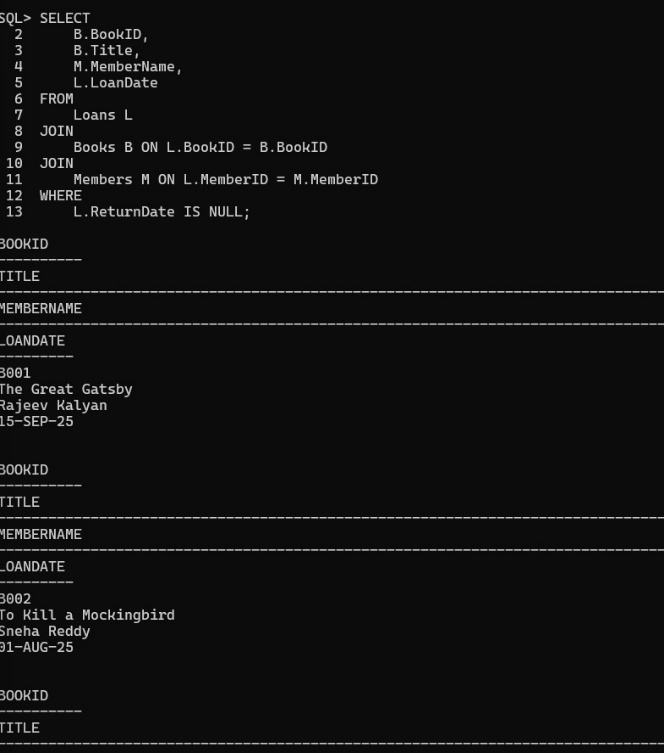
In this Hospital Management System, the database schema is designed with three normalized tables: Doctors, Patients, and Appointments. The schema ensures data integrity through primary keys and foreign keys, linking appointments to both doctors and patients. The DATE data type is used for date\_of\_birth and appointment\_date, allowing accurate storage and retrieval of dates. Queries demonstrate practical functionality: listing all appointments for a specific doctor, retrieving the complete patient history, and counting the total number of patients treated by each doctor. The structure avoids data redundancy, supports referential integrity, and facilitates efficient joins and aggregations. By manually assigning IDs, the schema maintains control over unique identifiers without relying on auto-increment, providing a clear, scalable foundation for hospital data management.

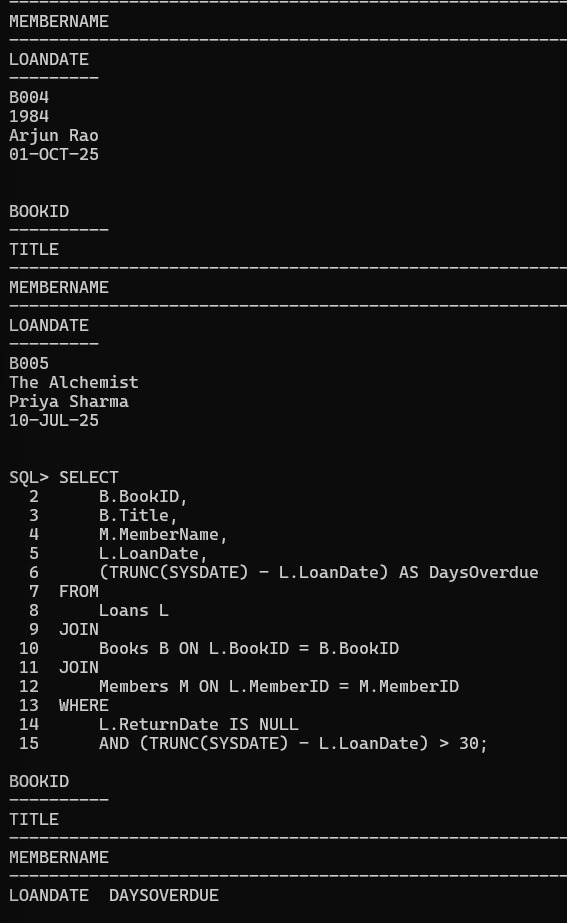
**Task 3** – Library Database  
Design schema for a Library Management System.

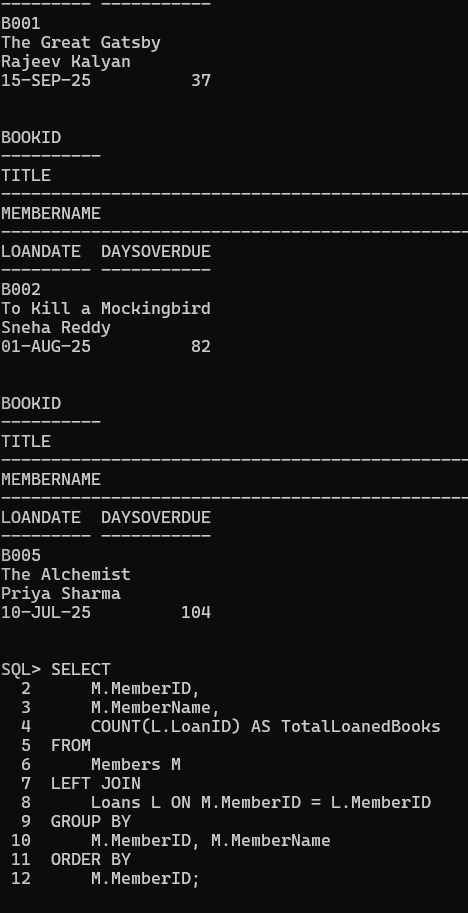


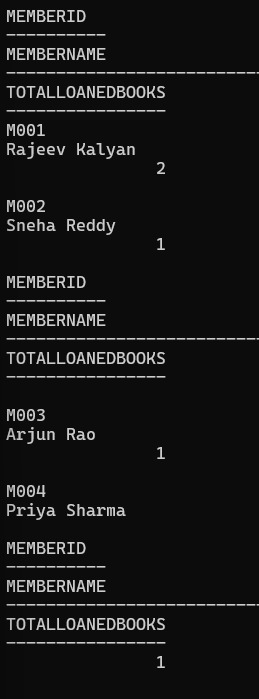








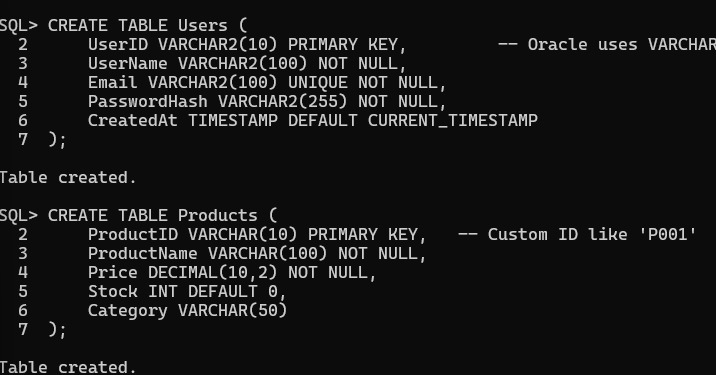


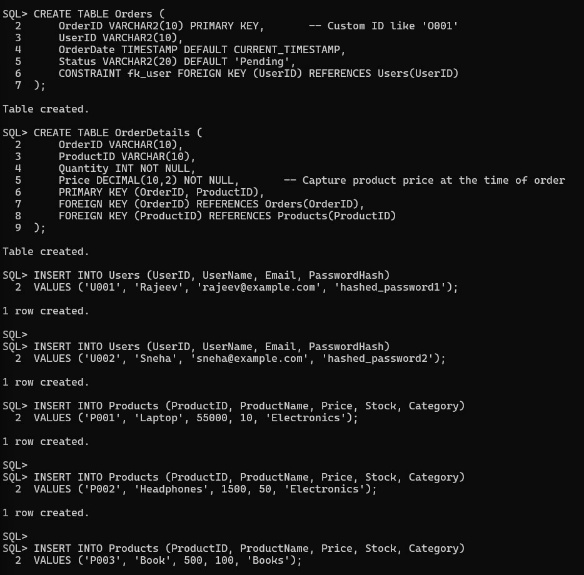


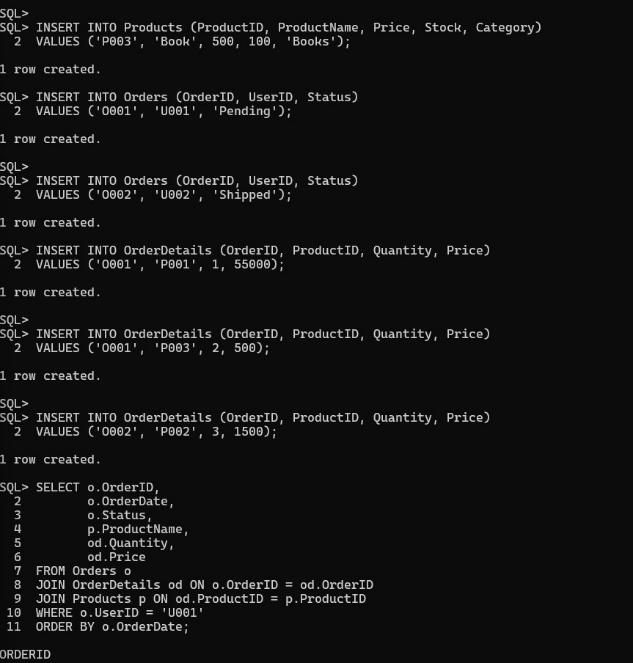
**Observation:**

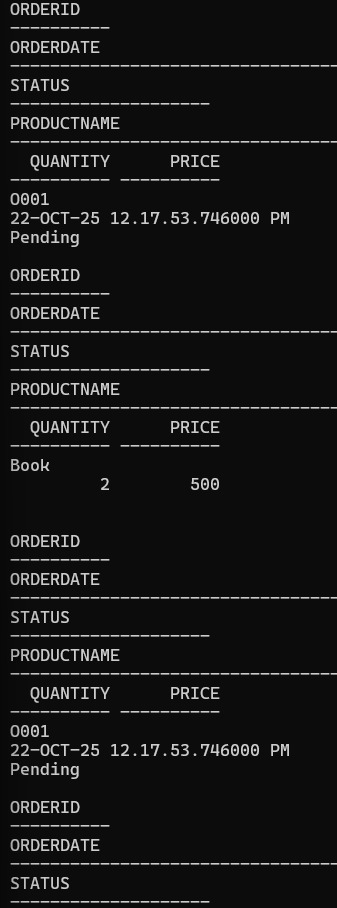
The Library Management System effectively tracks the relationships between books, members, and loans, providing real-time insights into borrowing activity. The query for currently issued books identifies all items that have not yet been returned, allowing librarians to monitor active loans. The overdue books query highlights items borrowed for more than 30 days, enabling timely reminders to members and efficient management of overdue returns. Counting the number of books loaned by each member gives a clear view of borrowing patterns, showing member engagement and resource usage. Overall, the system demonstrates efficient use of joins, date calculations, and aggregation, while the indexing strategy ensures faster query performance for searches, reporting, and library operations.

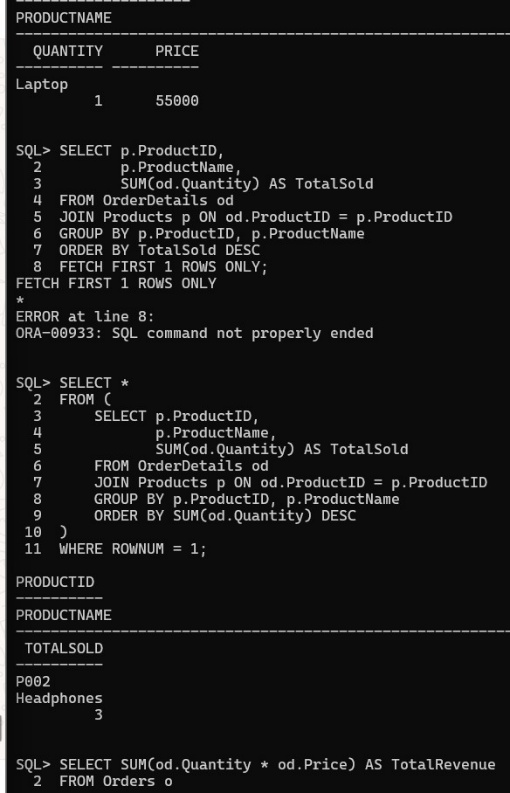
**Task 4** – Real-Time Application: E-commerce Product Data Cleaning  
An e-commerce company has product catalog data with inconsistencies.

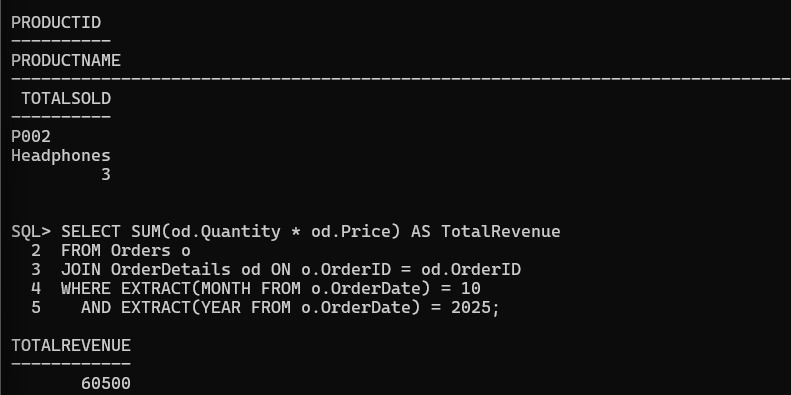












**Normalization Improvements:**

Your current schema is mostly in 3NF, but here are suggestions:

Historical Prices:

Already handled by storing Price in OrderDetails → prevents price changes in Products from affecting past orders.

User Email Uniqueness:

Already enforced with UNIQUE(Email) constraint.

Optional Denormalization for Performance:

Add TotalAmount in Orders table to avoid joining OrderDetails repeatedly for revenue queries.

**Query Optimization Suggestions:**

Indexes:

Orders(UserID) → speeds up fetching user orders.

OrderDetails(ProductID) → speeds up most purchased product query.

Orders(OrderDate) → speeds up revenue queries.

Composite Indexes (if needed):

(OrderDate, UserID) → for combined date + user queries.

Avoid Functions on Indexed Columns:

Instead of EXTRACT(MONTH FROM o.OrderDate) on indexed column, consider storing a derived column OrderMonth to directly filter.

**Observation:**

The designed Oracle database for the e-commerce platform effectively organizes data into four main tables: Users, Products, Orders, and OrderDetails, ensuring clarity and relational integrity. Each table uses custom string IDs (VARCHAR2) instead of auto-increment values, which allows for more readable and deterministic identifiers. The schema is normalized up to 3NF, eliminating redundancy while preserving historical data, such as storing product prices in OrderDetails to maintain accurate order records even if product prices change. Oracle-specific data types like VARCHAR2 and TIMESTAMP are used correctly, and default values (CURRENT\_TIMESTAMP) automate timestamp recording. Sample queries demonstrate how to retrieve all orders by a user, identify the most purchased product, and calculate monthly revenue, with proper aggregation, joins, and filtering. Optimization considerations include indexing Orders(UserID), OrderDetails(ProductID), and Orders(OrderDate) for faster query performance. Some queries, such as “most purchased product,” require Oracle-compatible syntax adjustments like using a subquery with ROWNUM instead of FETCH FIRST for versions prior to 12c. Overall, the database design is efficient, normalized, and optimized for querying large datasets, ensuring reliable e-commerce operations.