|  |
| --- |
| Student’s Full Name: SHREYA VIKAS KOLTE |
| Course Title: INFO 579: SQL/NoSQL Databases for Data and Information Sciences |
| Term name and year: Spring 2024 |
| Submission Week: Week 16 Assignment |
| Instructor’s Name: Prof. Nayem Rahman |
| Date of Submission: 08-05-2024 |

**Problem Statement:**

The Hospital Management System (HMS) project aims to resolve the challenges healthcare facilities face in managing critical patient information, staff allocations, and billing processes. Current systems often lack efficient data organization, resulting in delayed decision-making, billing errors, and a lack of comprehensive insights into healthcare operations. By creating a scalable database model with well-structured entities and relationships, the HMS will streamline data collection, improve patient record accessibility, and facilitate seamless management of healthcare processes.

**Data Explanation:**

* First data sheetcontains unstructured data representing individual patient details like name, contact information, birthdate, gender, address, and email. It also includes links to associated doctors and other healthcare professionals.
* The second data sheet presents normalized patient data organized into multiple columns for:
  + **Patient Information**: ID, name, contact, birth date, gender, and email.
  + **Medical History**: Highlights health conditions or surgical needs.
  + **Billing Details**: Associates each patient with a billing ID.

|  |
| --- |
| C:\Users\Shreya Kolte\OneDrive\Desktop\SEM 2 UOA\SQL - NO SQL\Project\Conceptual Diagram update-2.png |

Conceptual Model

The conceptual model provides a high-level overview of the entities and their relationships within the Hospital Management System (HMS).

|  |
| --- |
| C:\Users\Shreya Kolte\OneDrive\Desktop\SEM 2 UOA\SQL - NO SQL\Project\Logical Diagram Update-2.png |

Logical Model

The logical diagram expands upon the conceptual model by introducing more granular relationships and attributes. It includes primary (PK) and foreign keys (FK) that enforce relationships among entities.

**Many-to-Many Relationships:**

* **Patient-Doctor (DoctorPatient Table):**

Description: A patient may consult multiple doctors, and a doctor may treat multiple patients.

* Implementation: This relationship is managed through the DoctorPatient junction table, which links PatientID to DoctorID.
* **Patient-Nurse (PatientNurse Table):**
* Description: Multiple nurses can attend to one patient, and a nurse can provide care to multiple patients.
* Implementation: The PatientNurse junction table links PatientID to NurseID.
* **Patient-Surgery (PatientSurgery Table):**
* Description: A patient may undergo multiple surgeries, and a surgery may involve multiple patients.
* Implementation: The PatientSurgery table links PatientID to SurgeryID.

**One-to-Many Relationships:**

* **Doctor-Department:**

Description: A department typically has many doctors, but a doctor is usually associated with only one department.

Implementation: The Doctor table contains a DeptID field, which is a foreign key referencing the Department table's primary key.

* **Surgery-Doctor:**

Description: Multiple surgeries can be handled by a single doctor.

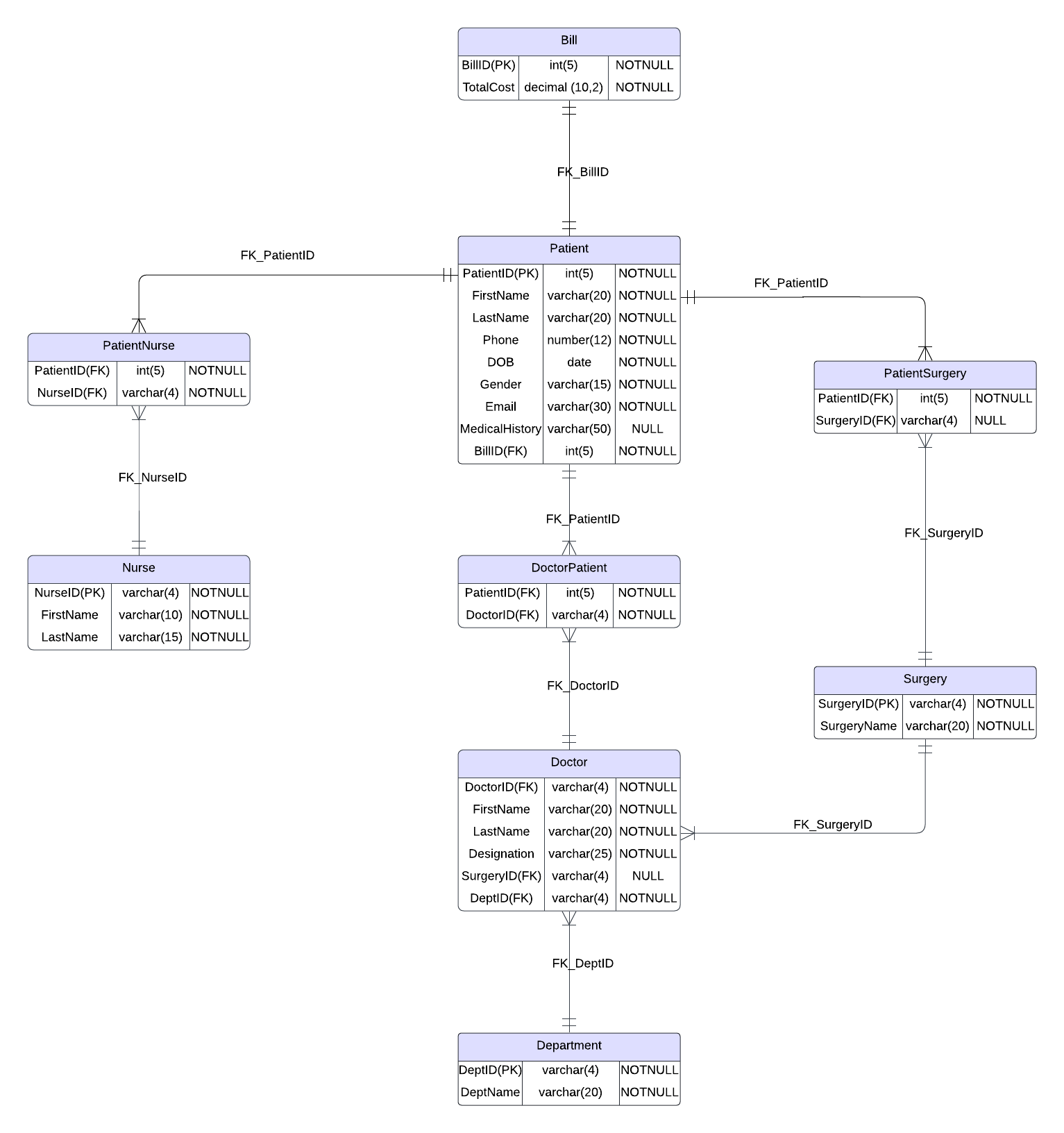
Implementation: The Doctor table contains a SurgeryID field linking to the primary key in the Surgery table.

**One-to-One Relationships:**

* **Patient-Bill:**
  + Description: Each patient has a unique billing record.
  + Implementation: The Patient table has a foreign key BillID, referencing the Bill table's primary key.

Week 16 Final Project Report: **Total Points - 100**

5. Develop the physical model based on the Logical Model

 Physical Model

The physical model is constructed using the logical design as a reference. It provides a detailed structure for each table and column within the Hospital Management System (HMS). It defines the database schema with constraints, data types, and relationships, ensuring proper data integrity and consistency.

6. Create tables using a database system. Insert data into the database tables. You must provide the DDL (CREATE TABLE statements), INSERT statements, and SELECT statements.

Details: Create the tables that you have come up with (the table must be based on the Physical Model).

(a) Columns, Primary Key (PK), Data Type and length, and NULL/NOT NULL need to be implemented, per the Physical Model.

(b) Show the table definition (DDL) that you implemented (not in a graphical view).

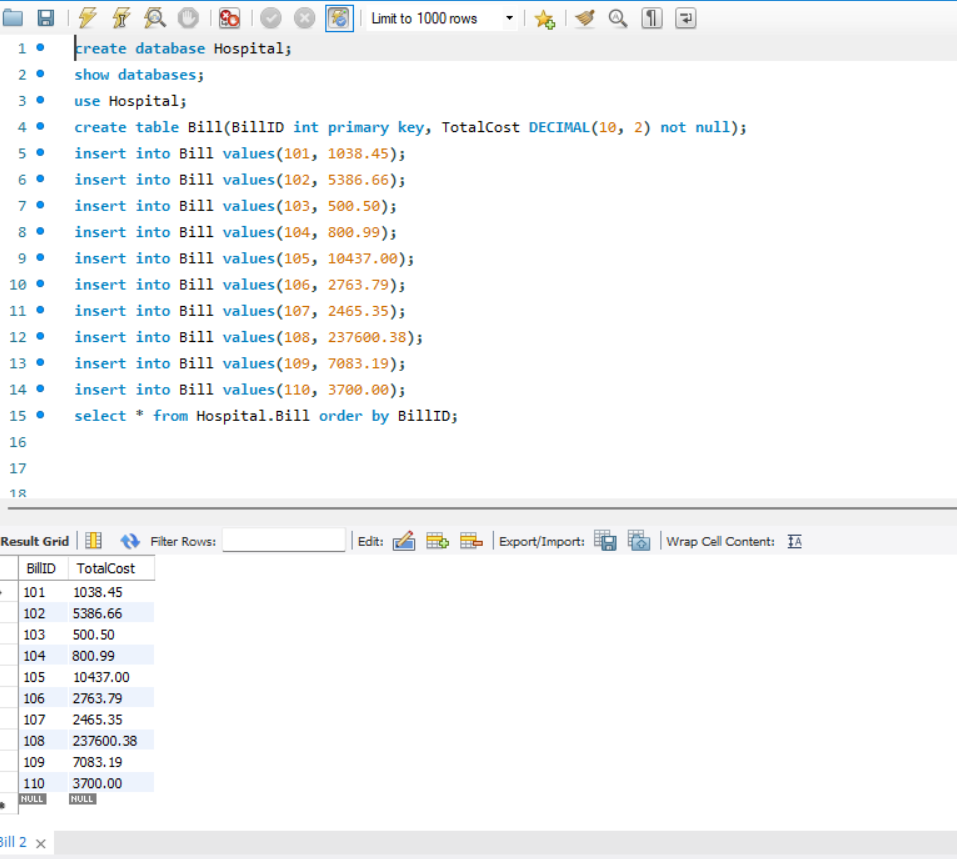
(c) Insert the complete set of data that you have come up with and show the insert statements used.

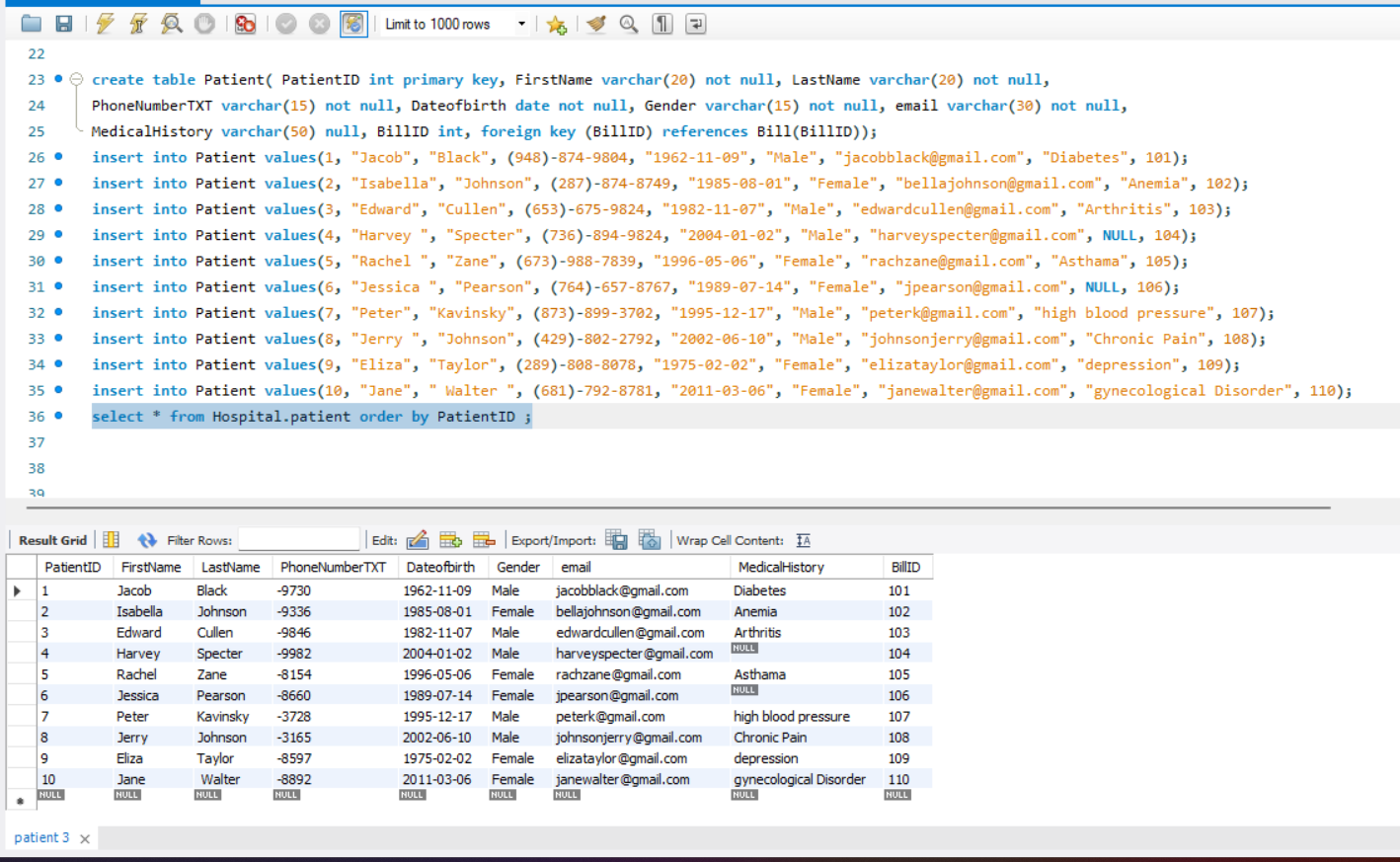
7. Create a variety of SQL queries to retrieve data from one or many tables:

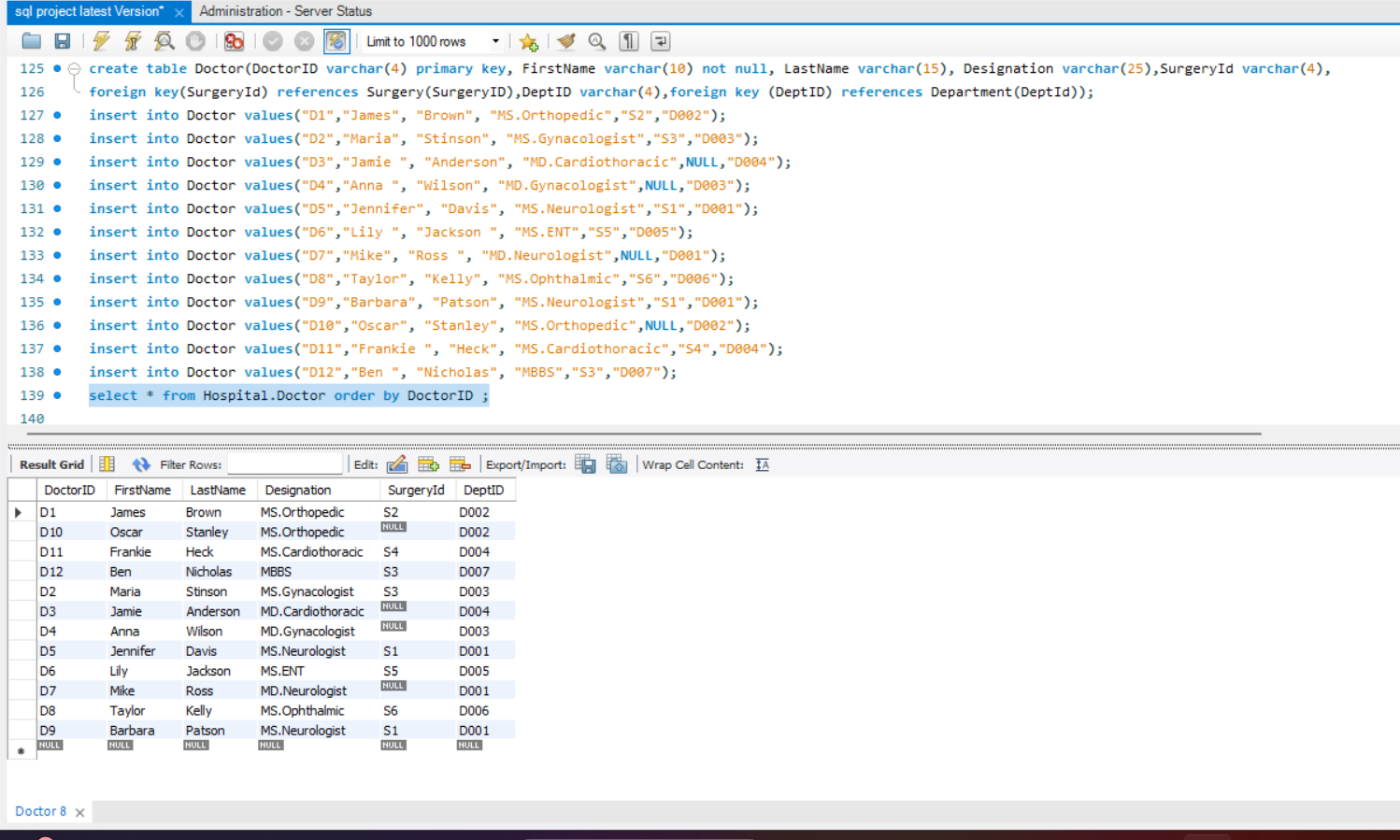
1. Retrieve the data from each table by using the SELECT \* statement and order by PK column(s).

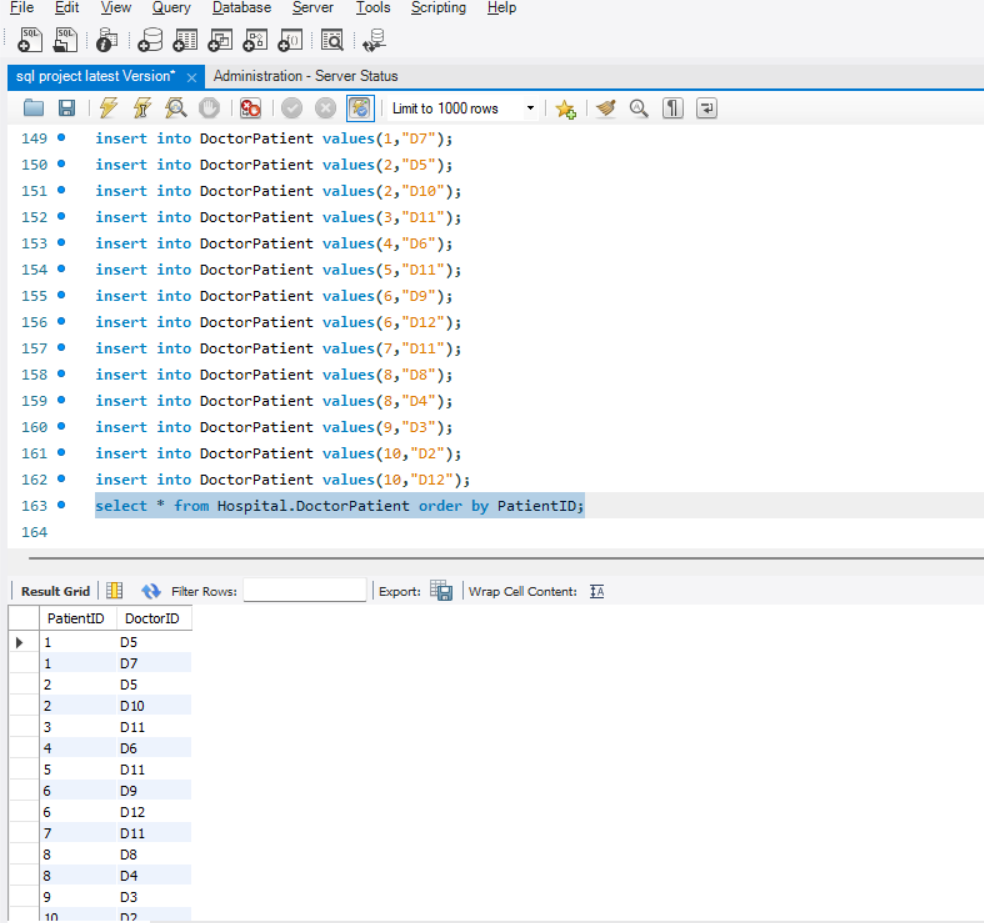
Show the output. Make sure you show the print screen of the complete set of rows and columns.

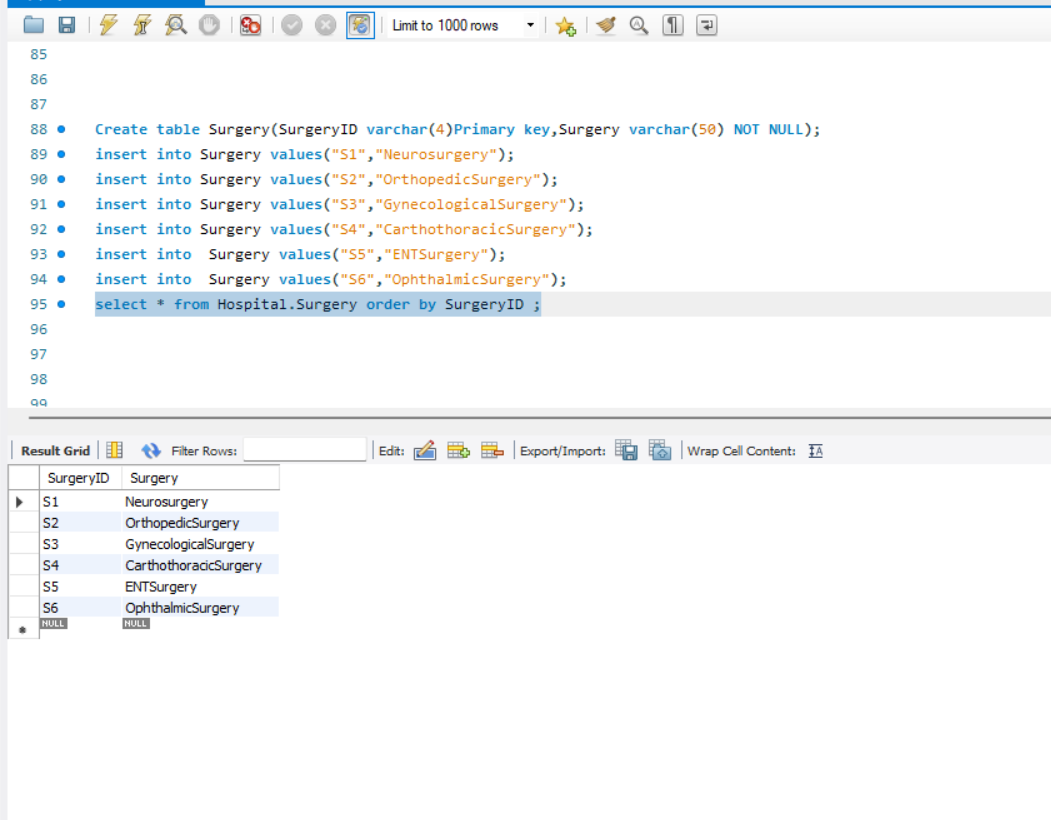
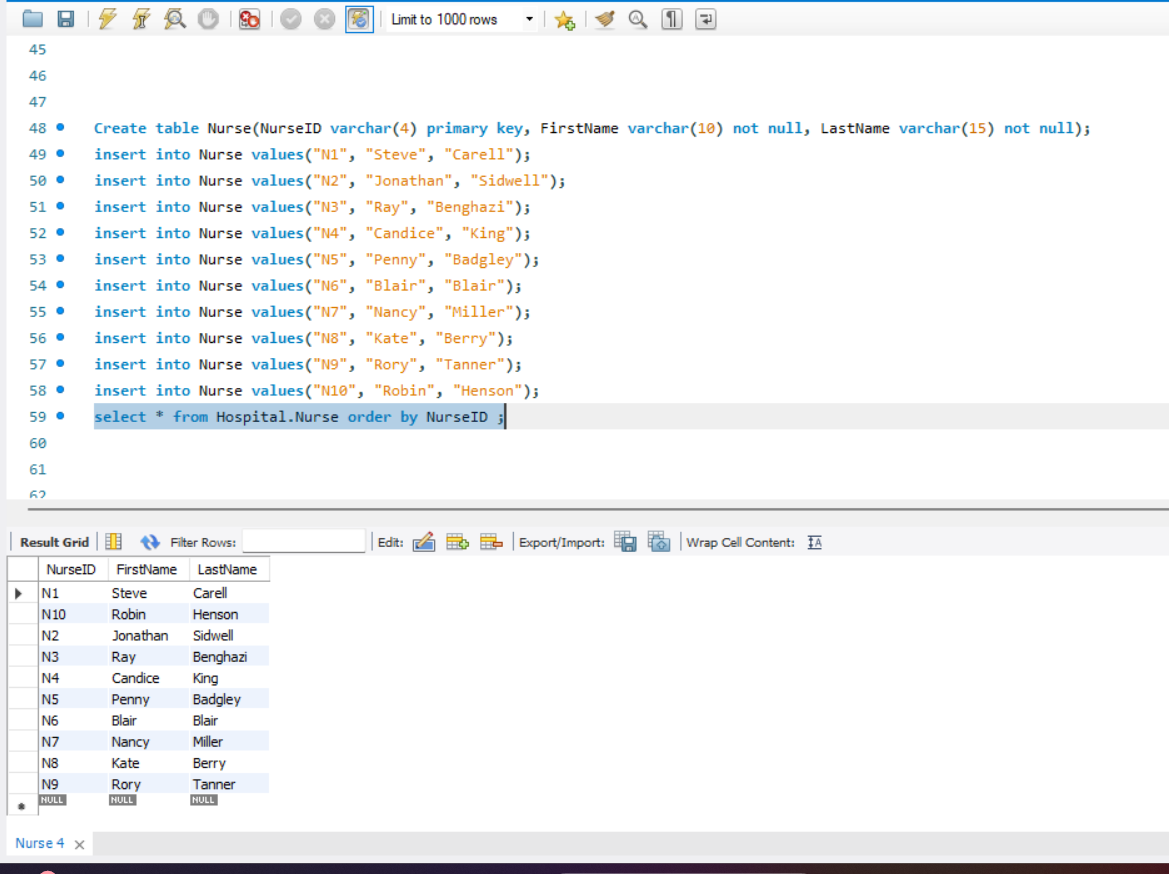
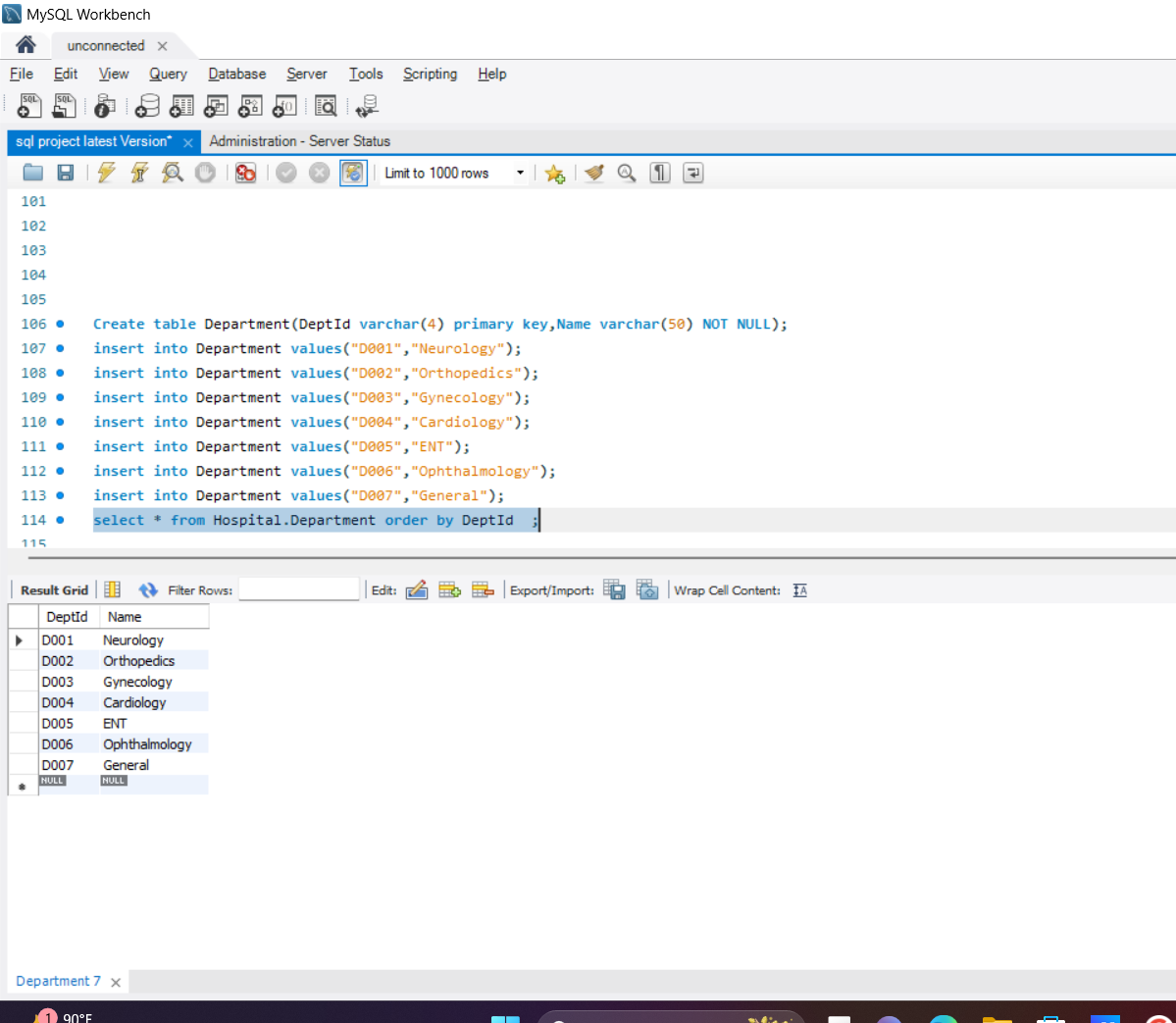
The rows must be ordered by PK column(s).

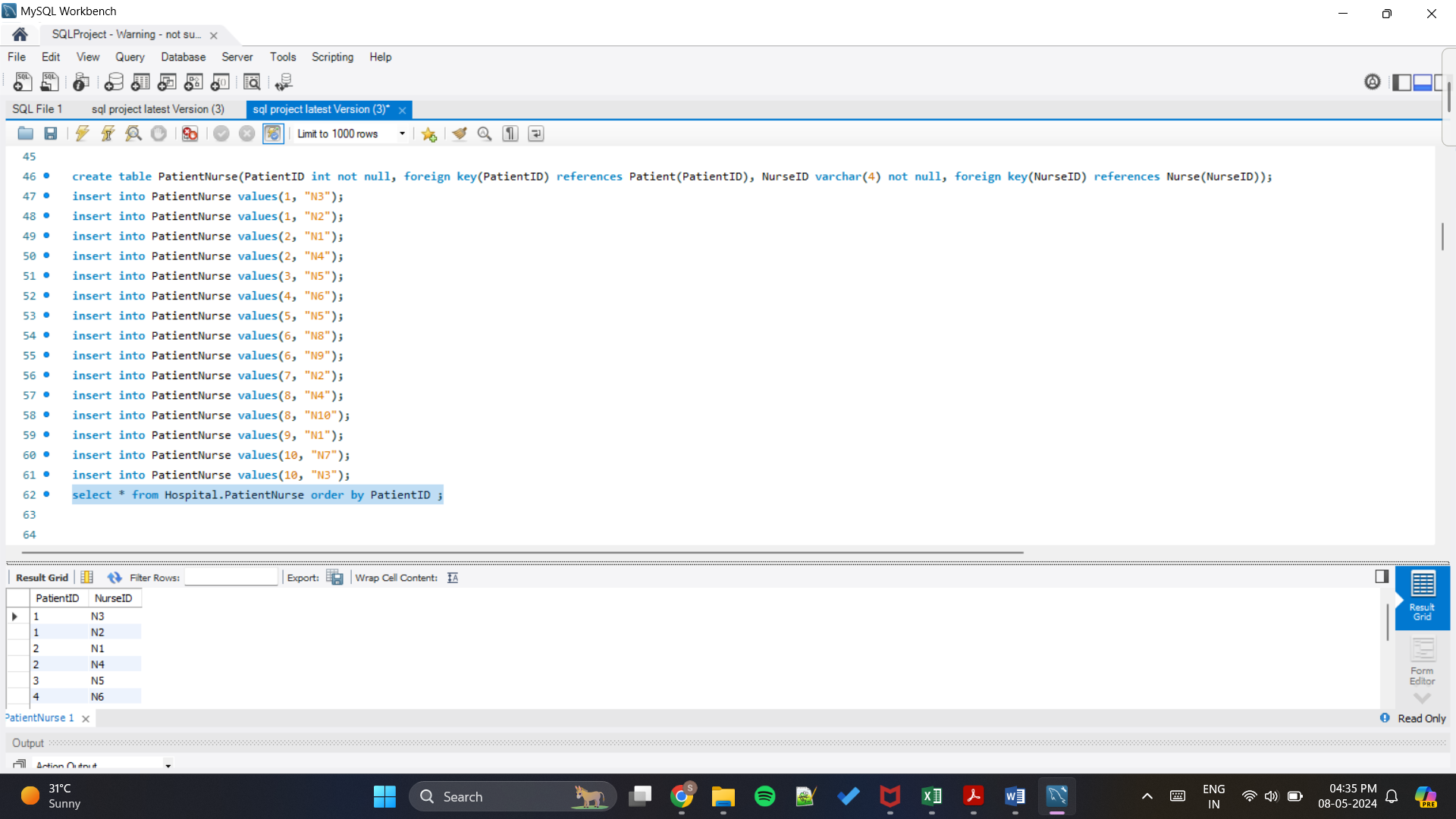


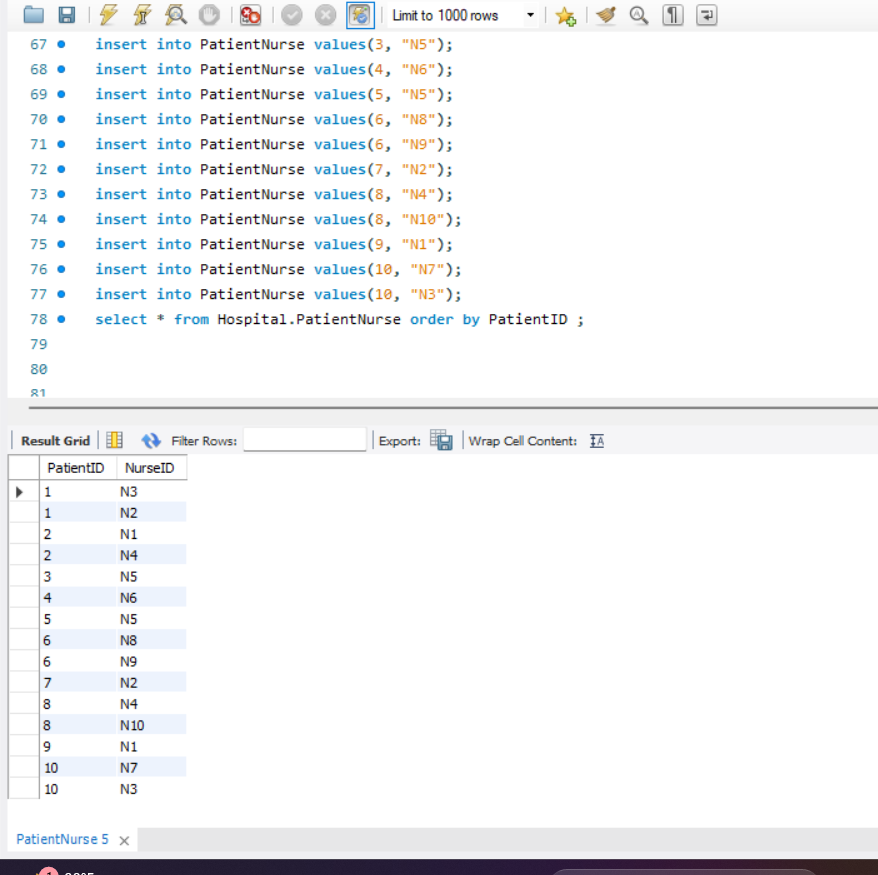


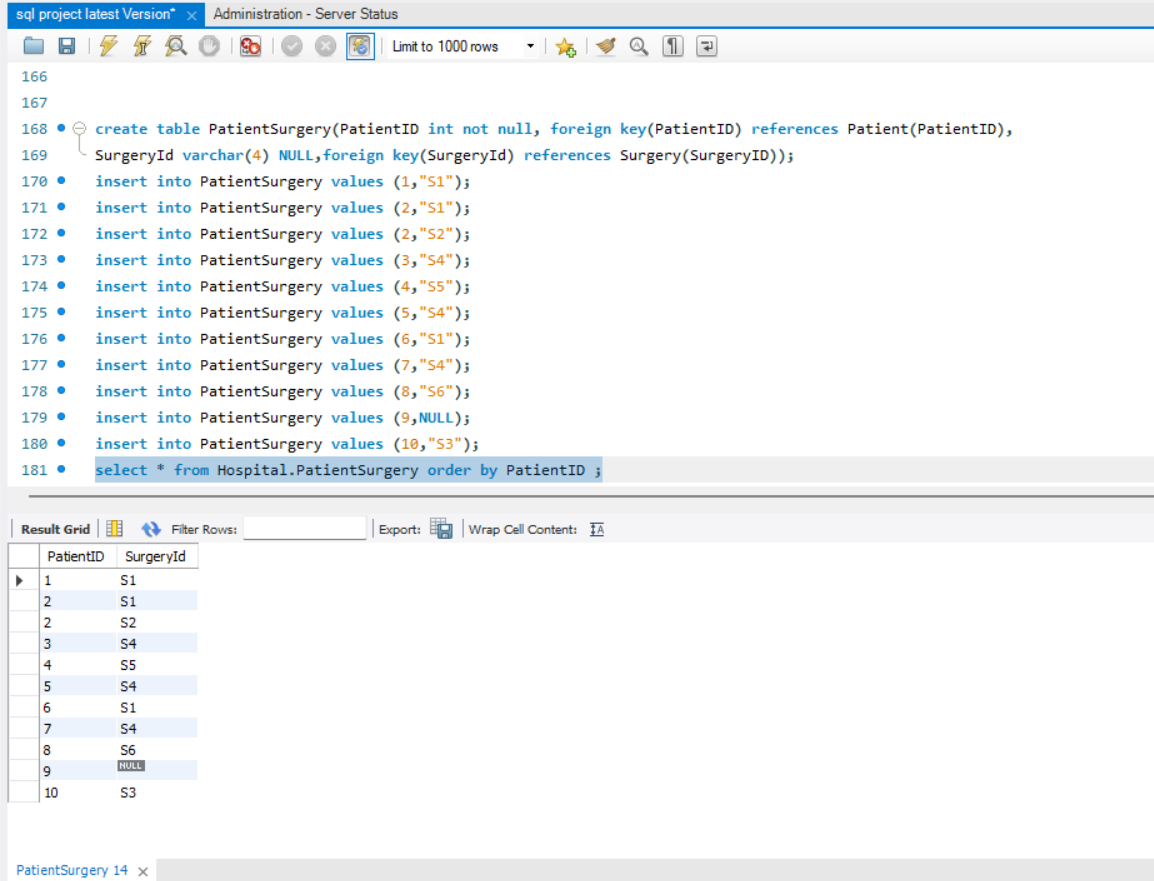




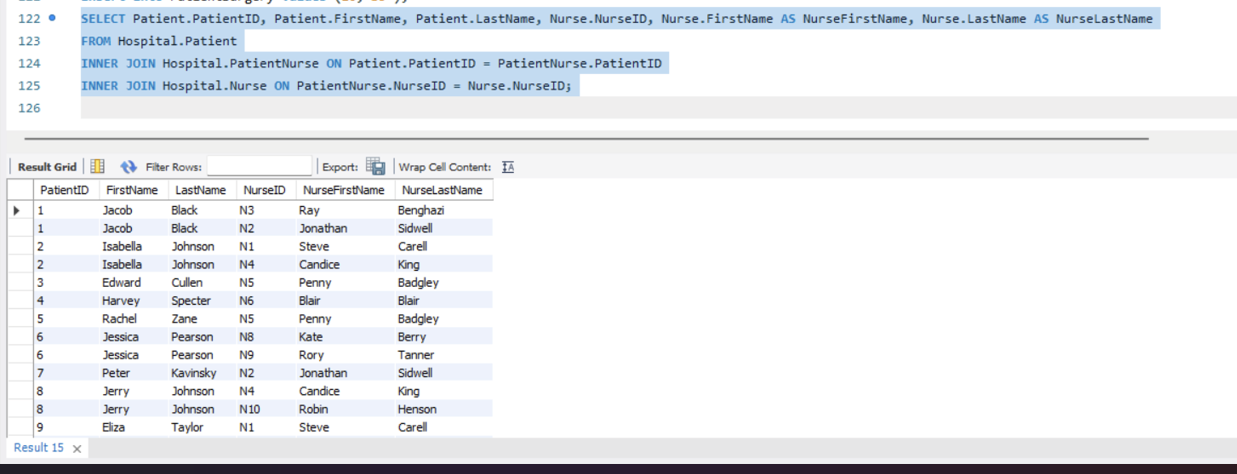






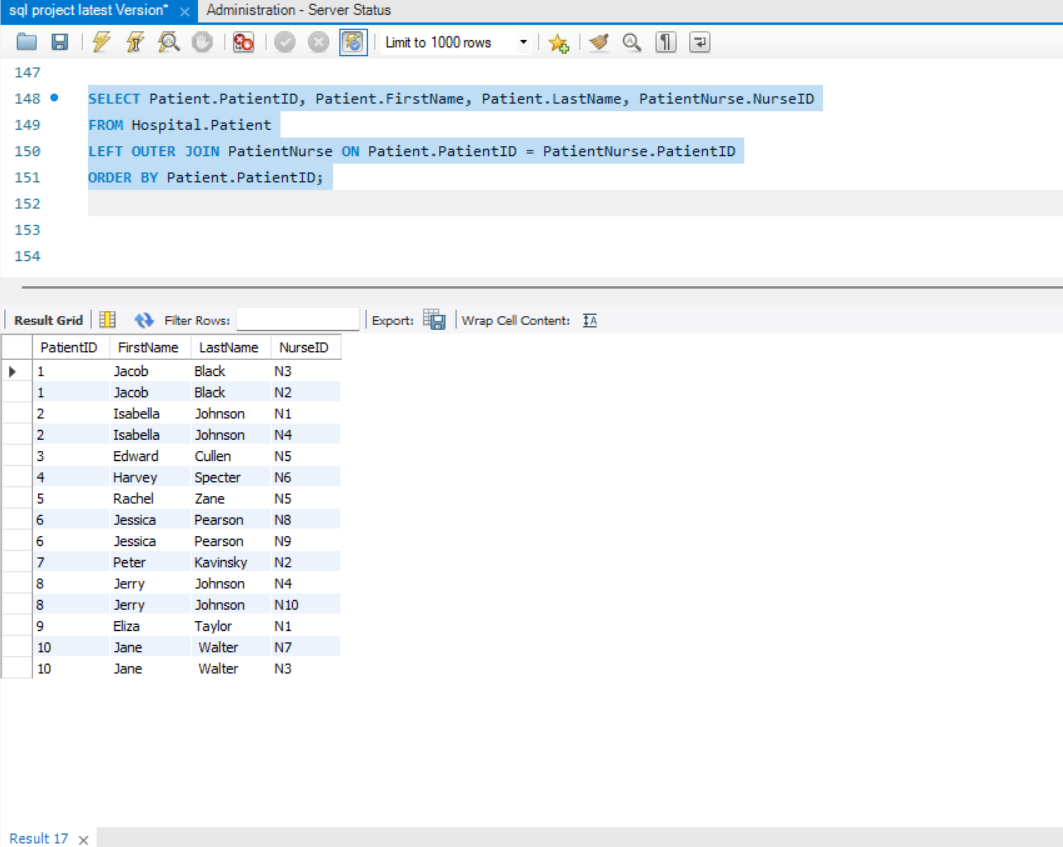


2. Write an SQL involving the junction table and two other related tables. You must use the INNER JOIN to connect with all three tables. The database that you created must be included in your SQL queries.



**Interpretation:** This query uses an INNER JOIN to retrieve data from the tables Patient, PatientNurse, and Nurse. It selects the PatientID, FirstName, and LastName columns from the Patient table, along with the NurseID, FirstName, and LastName columns from the Nurse table. It joins the Patient and Nurse tables based on the NurseID column in the PatientNurse table. This query will only return rows where there are matching records in all three tables. In other words, it will return patients who have been assigned a nurse, and it will display the details of both the patient and the nurse they are assigned to.

3. Write an SQL by including two or more tables and using the LEFT OUTER JOIN. Show the results and sort the results by key field(s). Interpret the results compared to what an INNER JOIN does.



**Interpretation:** LEFT OUTER JOIN retrieves data from the Patient and PatientNurse tables. It selects the PatientID, FirstName, and LastName columns from the Patient table and the NurseID column from the PatientNurse table. It orders the results by the PatientID.

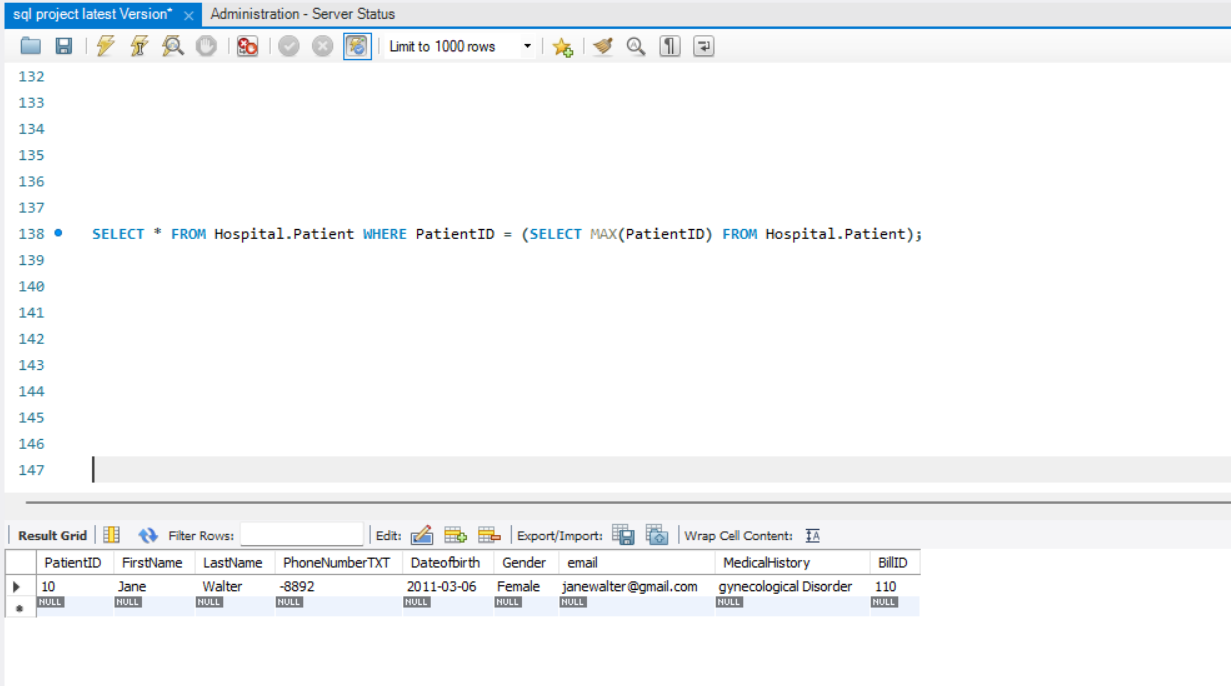
A LEFT OUTER JOIN returns all rows from the left table (Patient in this case) and the matched rows from the right table (PatientNurse), with NULL values in the columns from the right table for rows that have no matching rows in the left table. So, in this query it will return all patients, whether or not they have been assigned a nurse, and it will display the NurseID if the patient has been assigned one or NULL if not.

Comparing the results:

The INNER JOIN will only return patients who have been assigned a nurse, along with details of their assigned nurse.

The LEFT JOIN will return all patients, regardless of whether they have been assigned a nurse or not. If a patient has been assigned a nurse, it will display the NurseID, otherwise, it will display NULL.

4. Write a single-row subquery. Show the results and sort the results by key field(s). Interpret the output.

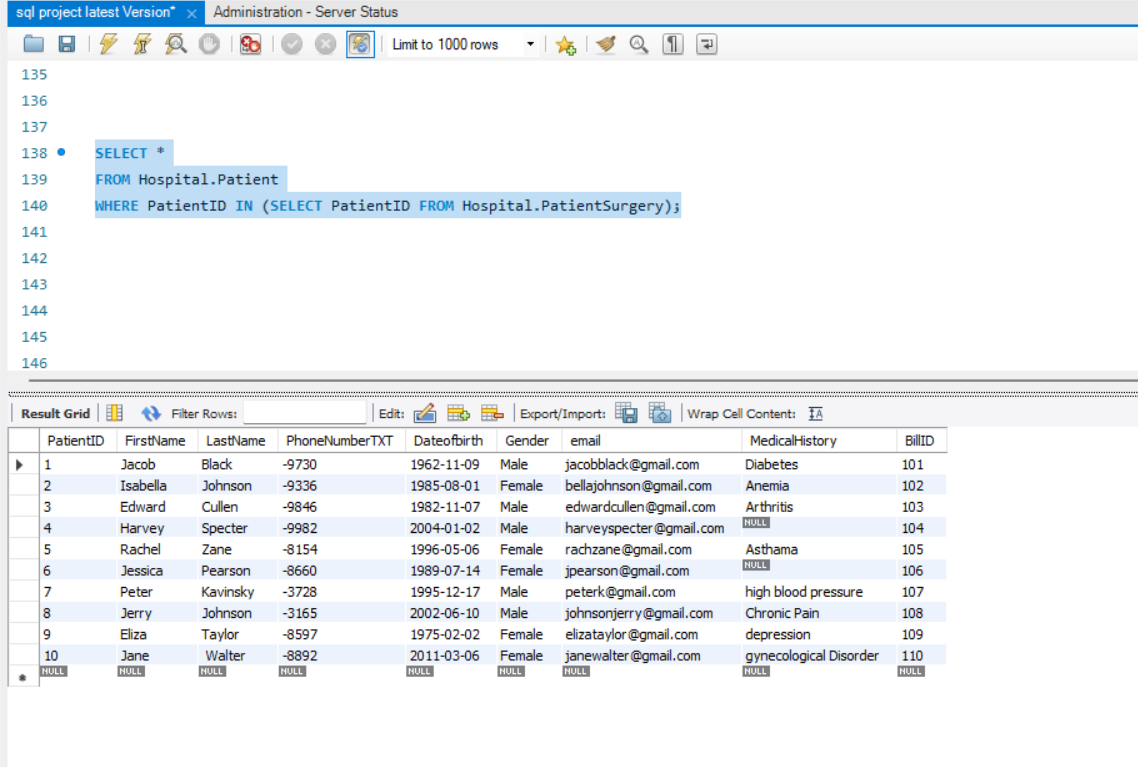


**Interpretation:** The outer query retrieves all columns from the Hospital.Patient table where PatientID matches the value found by the inner query.

The subquery, SELECT MAX(PatientID) FROM Hospital.Patient, identifies the maximum PatientID from the Patient table. This subquery returns a single value: the highest numerical PatientID.

The output shows the patient record with the highest PatientID, meaning that the most recently added or last updated patient record is Jane Walter.

5. Write a multiple-row subquery. Show the results and sort the results by key field(s). Interpret the output.



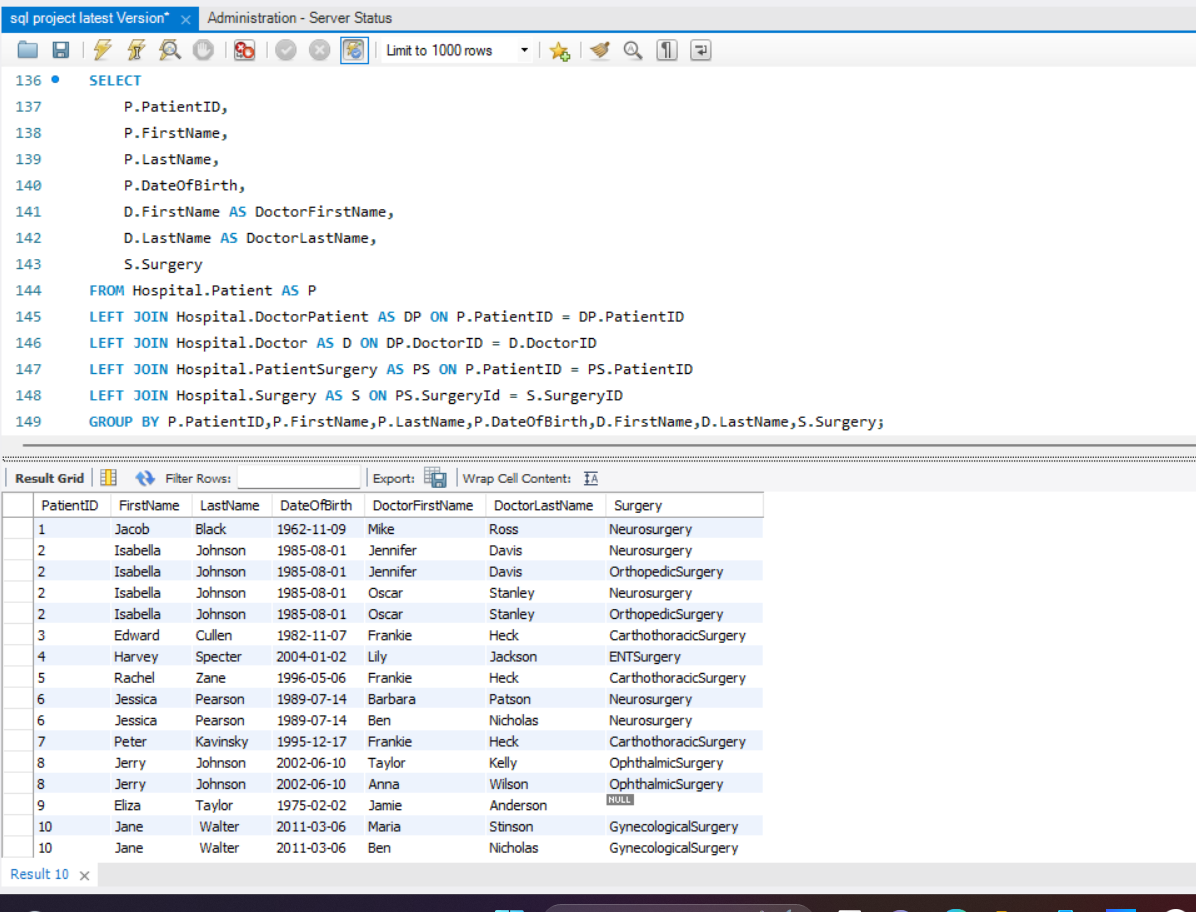
**Interpretation:** The main query retrieves all patients whose PatientID values are present in a subquery result set.

The subquery returns multiple PatientID values based on certain criteria (like those involved in surgeries).

The output shows a list of patients sorted by the key field PatientID.

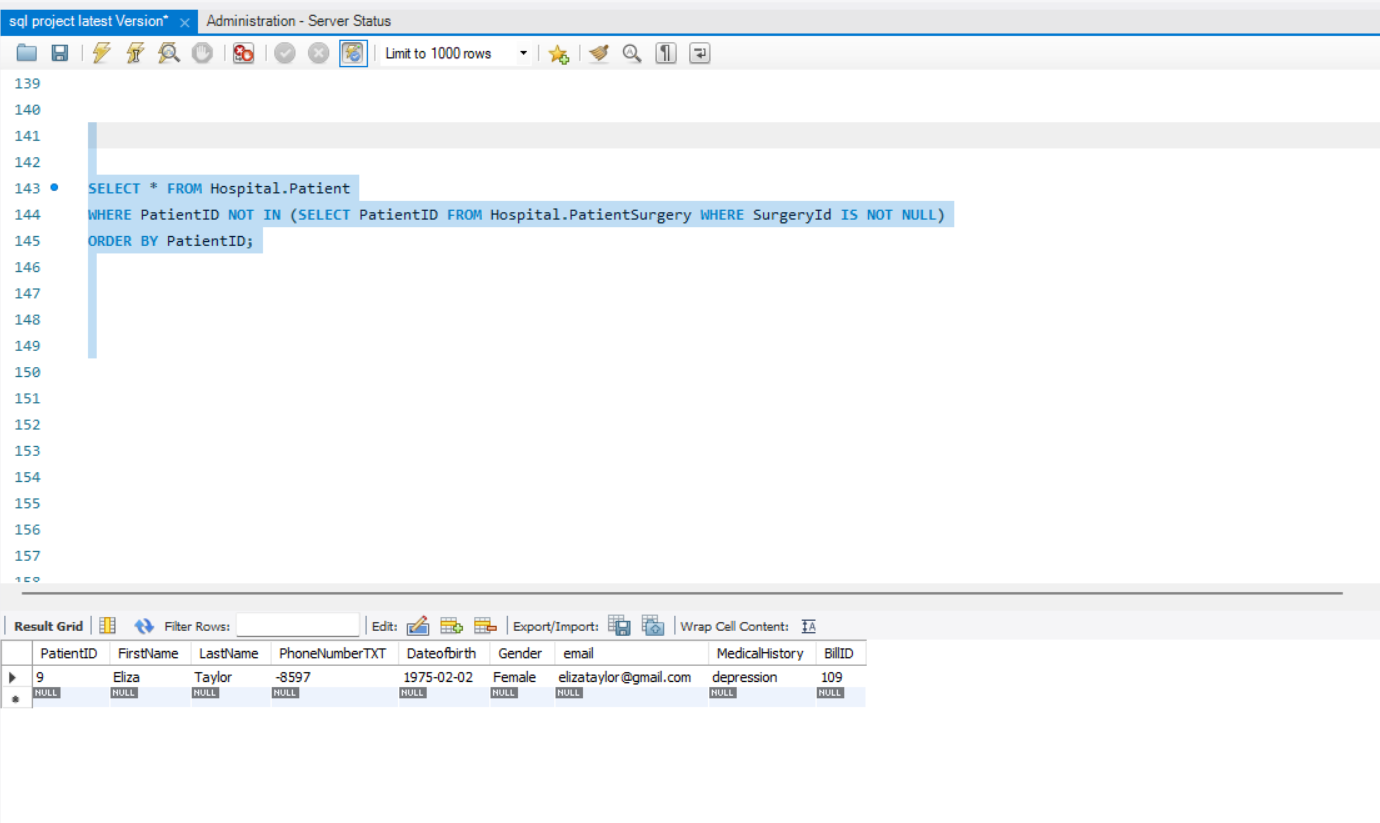
Each patient is associated with their name, contact information, medical history, and billing details.

6. Write an SQL to aggregate the results by using multiple columns in the SELECT clause. Interpret the output.

**Interpretation:** The query aggregates patient information by combining multiple columns to provide a comprehensive view of patients, their associated doctors, and surgeries.

The output shows each patient along with their treating doctors and associated surgeries. For example, patient Jacob Black (PatientID: 1) is being treated by doctors Jennifer Davis and Mike Ross, with surgery identified as "Neurosurgery."

The aggregation ensures the information is grouped based on patient attributes and links each patient to the relevant doctors and surgeries.

7. Write a subquery using the NOT IN operator. Show the results and sort the results by key field(s). Interpret the output.

**Interpretation:** Subquery: The inner query retrieves the PatientID values from the PatientSurgery table where the SurgeryID is not null. These IDs represent patients who have undergone one or more surgeries.

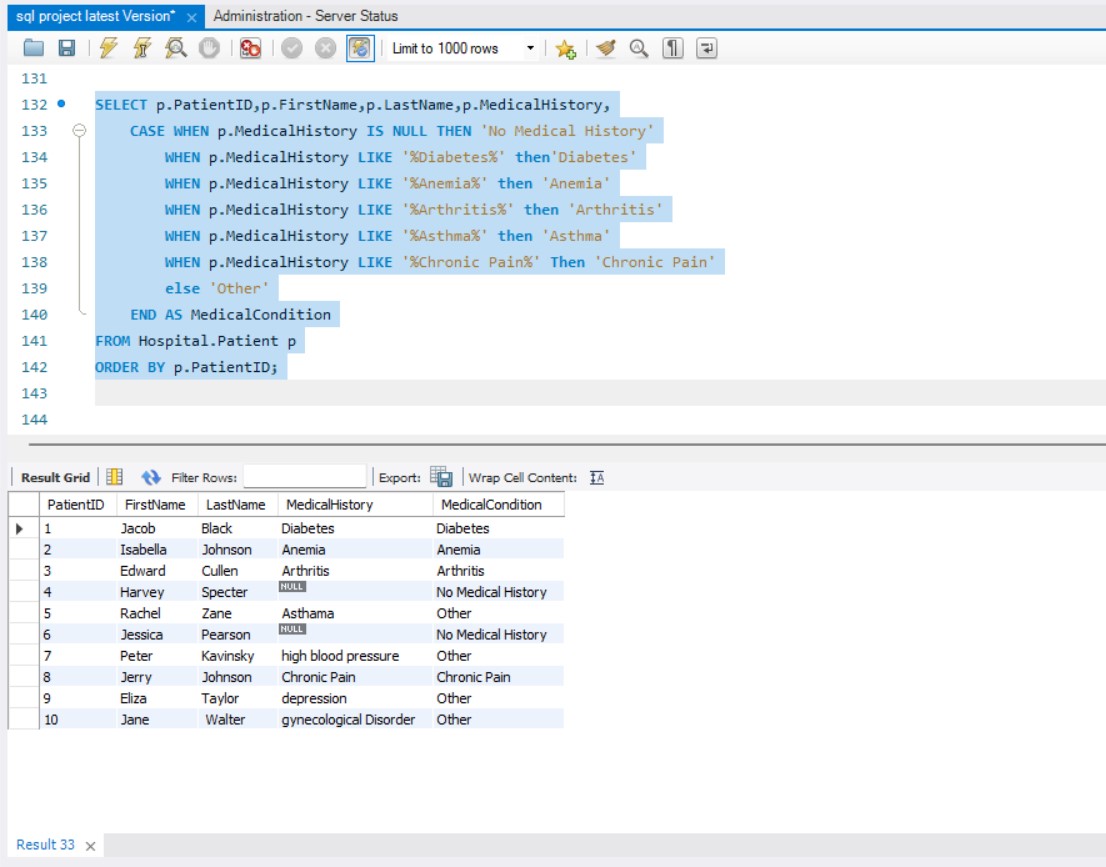
Main Query: The outer query retrieves all columns from the Hospital.Patient table where PatientID values are not in the subquery result set. This means only patients who have not had a surgery are selected.

The output shows the patient record for Eliza Taylor (PatientID: 9).

She has not undergone any surgery, as indicated by her absence in the PatientSurgery table.

Her medical history mentions "depression," and she is linked to BillID 109.

8. Write a query using a CASE statement. Show the results and sort the results by key field(s). Interpret the output.



**Interpretation:** The CASE statement evaluates the MedicalHistory column of each patient record.

It checks for specific medical conditions like diabetes, anemia, arthritis, and asthma using pattern matching (LIKE).

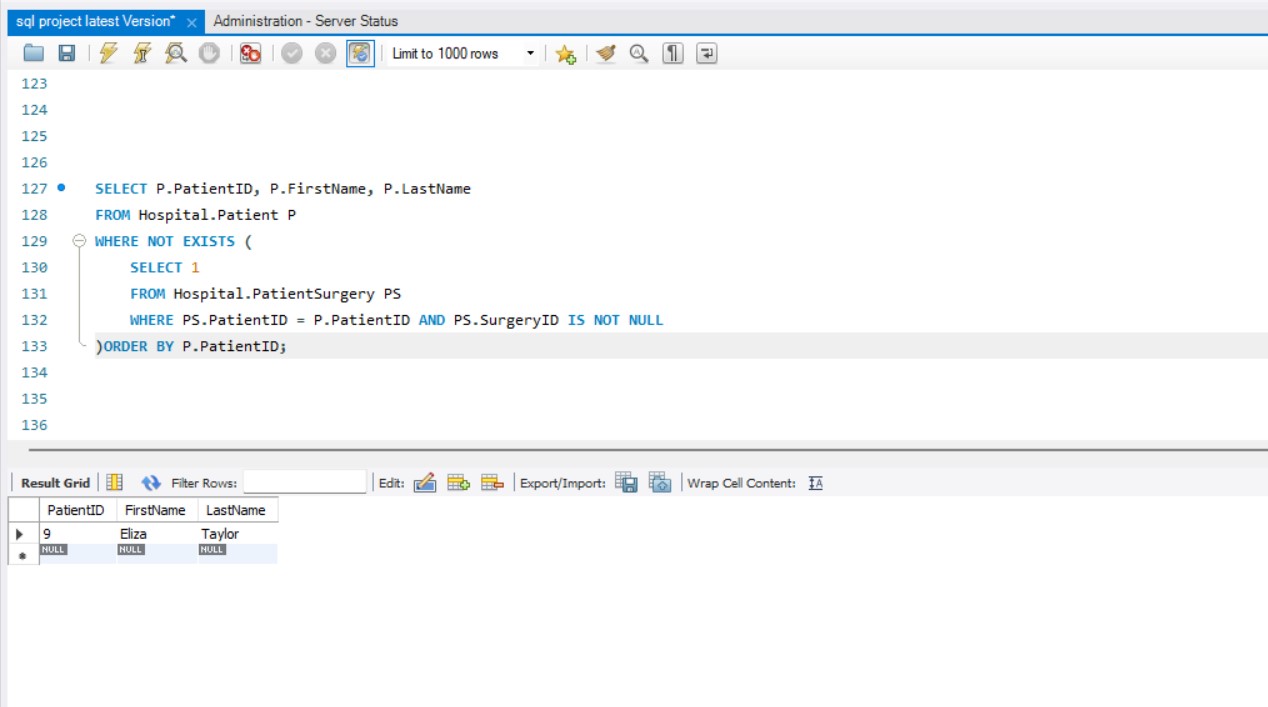
Depending on the matched pattern, the patient is categorized under a particular medical condition.

If no known condition is found, the result defaults to "Other."

The query outputs the following columns for each patient: PatientID, first name, last name, MedicalHistory, and a new MedicalCondition field that categorizes each patient's condition.

The result is sorted by PatientID, ensuring that patients are listed in the order of their IDs.

9. Write a query using the NOT EXISTS operator. Show the results and sort the results by key field(s). Interpret the output.



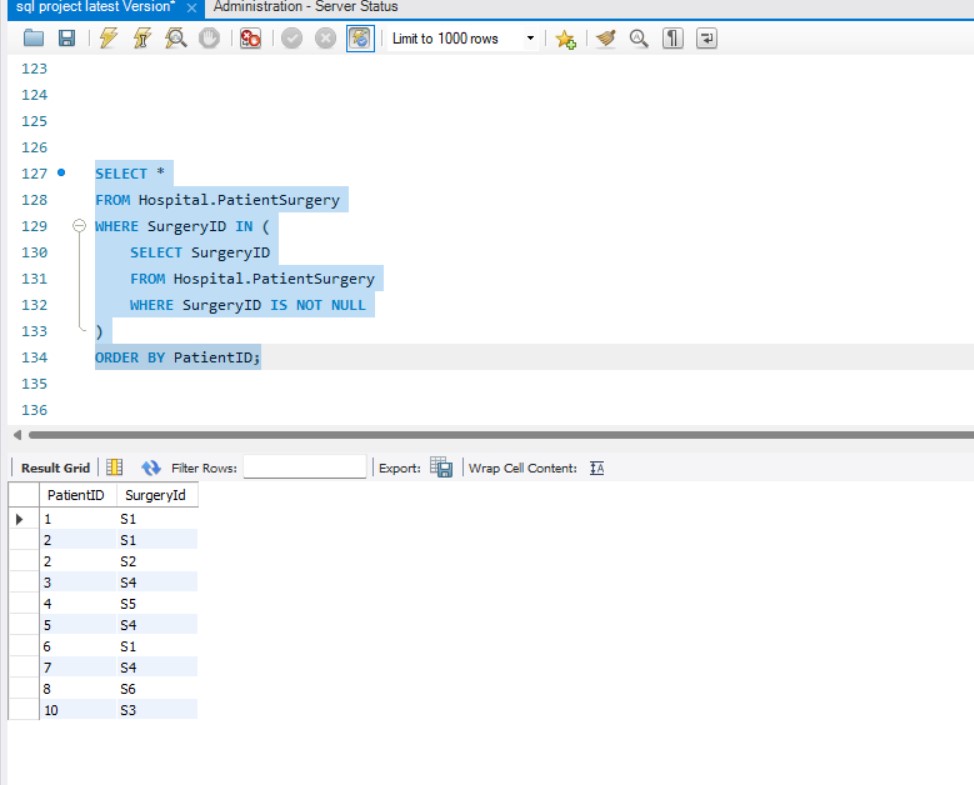
**Interpretation:** Subquery: The inner query looks for PatientID values in the PatientSurgery table, filtering for records where SurgeryID is not null. This identifies patients who have undergone at least one surgery.

Main Query: The outer query retrieves patient records from the Hospital.Patient table where the PatientID is not present in the subquery results, meaning these patients have not had any surgeries.

Sorting: The results are sorted by PatientID.

The query helps identify patients who are not scheduled for any surgery, making them a separate group for potential future planning or different treatment options.

10. Write a subquery using the NOT NULL operator in the inner query. Show the results and sort the results by key field(s). Interpret the output.



**Interpretation:** Subquery:The inner query filters out patients with null values in critical fields by checking for NOT NULL.This ensures that only patients with complete information are included.

Main Query: The outer query retrieves patient details (PatientID, first name, last name) for patients whose IDs are included in the subquery results.

It excludes any patient records with null values in specific columns.

The results are sorted by PatientID.

The result shows Eliza Taylor (PatientID: 9) as the only patient listed, meaning she meets the condition that her relevant fields are not null in the subquery.

Her information, like PatientID, name, and billing details, is fully provided in the output.

**Real-World Implications of the Project**

The Hospital Management System (HMS) project is designed to provide a streamlined and efficient way to handle the critical operations of a healthcare facility.

1. **Patient:**
   * **Real-World Implication:** Central to any healthcare facility, patients are at the heart of all activities. Maintaining comprehensive patient data is crucial for accurate diagnoses, effective treatments, and personalized care plans.
   * **Relationships:** Links with all other entities, including doctors (via many-to-many relationships), surgeries, departments, and billing. This ensures seamless coordination of care and efficient patient management.
2. **Doctor:**
   * **Real-World Implication:** Doctors play a pivotal role in delivering quality healthcare. By maintaining up-to-date records of doctors' specialties, departments, and patient assignments, the system ensures that patients are treated by the appropriate professionals.
   * **Relationships:** Connected to patients, surgeries, and departments, emphasizing a multidisciplinary approach to treatment.
3. **Nurse:**
   * **Real-World Implication:** Nurses are indispensable in patient care, providing continuous support before, during, and after treatments. Their records ensure accurate tracking of which nurses have attended to which patients.
   * **Relationships:** Tied to patients through many-to-many junctions, highlighting their involvement in multiple patients' care.
4. **Surgery:**
   * **Real-World Implication:** Surgeries are often complex and involve collaboration between various healthcare professionals. Detailed records of surgical procedures help coordinate teams, reduce errors, and improve patient outcomes.
   * **Relationships:** Associated with patients via many-to-many relationships and doctors through one-to-many links.
5. **Department:**
   * **Real-World Implication:** Healthcare departments specialize in specific medical fields and provide the infrastructure for specialized care. Keeping an accurate department allocation ensures patients are routed to the correct departments and receive the appropriate care.
   * **Relationships:** Connected to doctors in one-to-many relationships, as doctors typically belong to one department.
6. **Bill:**
   * **Real-World Implication:** Accurate billing is crucial for a hospital's financial health. It ensures transparency, minimizes errors, and facilitates payment processing. Each patient has a unique bill linked directly to their treatments.
   * **Relationships:** Linked one-to-one with each patient to ensure that every patient has an accurate and unique billing record.

**Efficient Coordination:** The many-to-many and one-to-many relationships ensure that patients are attended by multiple professionals efficiently while allowing collaboration across departments.

**Accurate Billing and Documentation:** A unified view of patients' treatments ensures accurate billing and documentation, reducing potential errors.

**Improved Decision-Making:** Aggregate data from relationships between doctors, nurses, departments, and patients provides management insights to allocate resources better and improve care quality.

**Enhanced Patient Experience:** The structured data model helps provide a holistic view of patient care, making follow-ups and treatment consistency easier to manage.

Overall, the Hospital Management System aims to make patient care safer, more efficient, and more transparent, ultimately improving patient outcomes and operational workflows for healthcare facilities.