SQL DATABASE For Amina_MedicalCare Hospital

AMINA IBRAHIM

Author

INTRODUCTION

The purpose of this report is to design and implement a fully normalized hospital database system using SQL. The database is structured to manage essential hospital operations such as patient management, doctor scheduling, medical records tracking, and appointment booking. Through proper normalization techniques, the database aims to ensure data integrity, avoid redundancy, and support efficient query execution. The report will document the design process, SQL statements for table creation and population, and queries to retrieve critical hospital information.

- **Objective**:

The goal is to design and implement a fully normalized hospital database in **SQL Server**. This database will manage patients, doctors, appointments, and medical records while ensuring data consistency, integrity, and efficient retrieval. This report documents the database schema, queries, and justifications for design decisions.

- **Overview**:

The database covers the following functionalities:

- Managing patient records
- Scheduling and tracking appointments
- Storing medical records
- Department management

Database Design and Normalization

- **Entity Identification**:

The main entities identified for the hospital database are: **(T-SQL)**

CREATE DATABASE Amina_MedicalCare

1. **PATIENTS**: Stores patient personal information and contact details.

   PatientID: This column serves as the unique identifier for each patient.
   Data Type: INT is chosen as it's an efficient and scalable way to store unique numbers.
   Constraint: PRIMARY KEY ensures uniqueness, while NOT NULL guarantees that every patient must have an ID.

   FullName: Stores the full name of the patient.
   Data Type: VARCHAR (100) is used because names vary in length but typically don't exceed 100 characters.

   Address: Holds the address of the patient.

Data Type: VARCHAR (100) is appropriate for most residential or postal addresses.

DateOfBirth (Captures the patient's date of birth.

Data Type: DATE is ideal for storing dates efficiently. (yyyy-mm-dd)

Insurance Stores the insurance provider information.

Data Type: VARCHAR (50) is used since insurance names or policies typically fit within this length.

Username: Holds the username for patient login or identification.

Data Type: VARCHAR (50) ensures flexibility for username lengths.

Password: Stores the patient's password for access to their records or hospital account.

Data Type: VARCHAR (50) is sufficient to handle encrypted or hashed password storage.

Email: Used to store the patient's email address.

Data Type: VARCHAR (100) is appropriate as email addresses generally don't exceed this length.

PhoneNumber: Captures the patient's contact number.

Data Type: VARCHAR (20) is chosen as phone numbers, including country codes, can be lengthy.

DateLeft: Stores the date the patient was discharged or left the hospital.

Data Type: DATE is appropriate for capturing this specific date information.

```
CREATE TABLE Patients (
 PatientID INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
 FullName VARCHAR (100) NOT NULL,
 Address VARCHAR (100),
 DateOfBirth DATE NOT NULL,
 Insurance VARCHAR (50),
 Username VARCHAR (50) UNIQUE,
 Password VARCHAR (50) NOT NULL,
 Email VARCHAR (100) NOT NULL,
 PhoneNumber VARCHAR (20) NOT NULL,
 DateLeft DATE
);
```

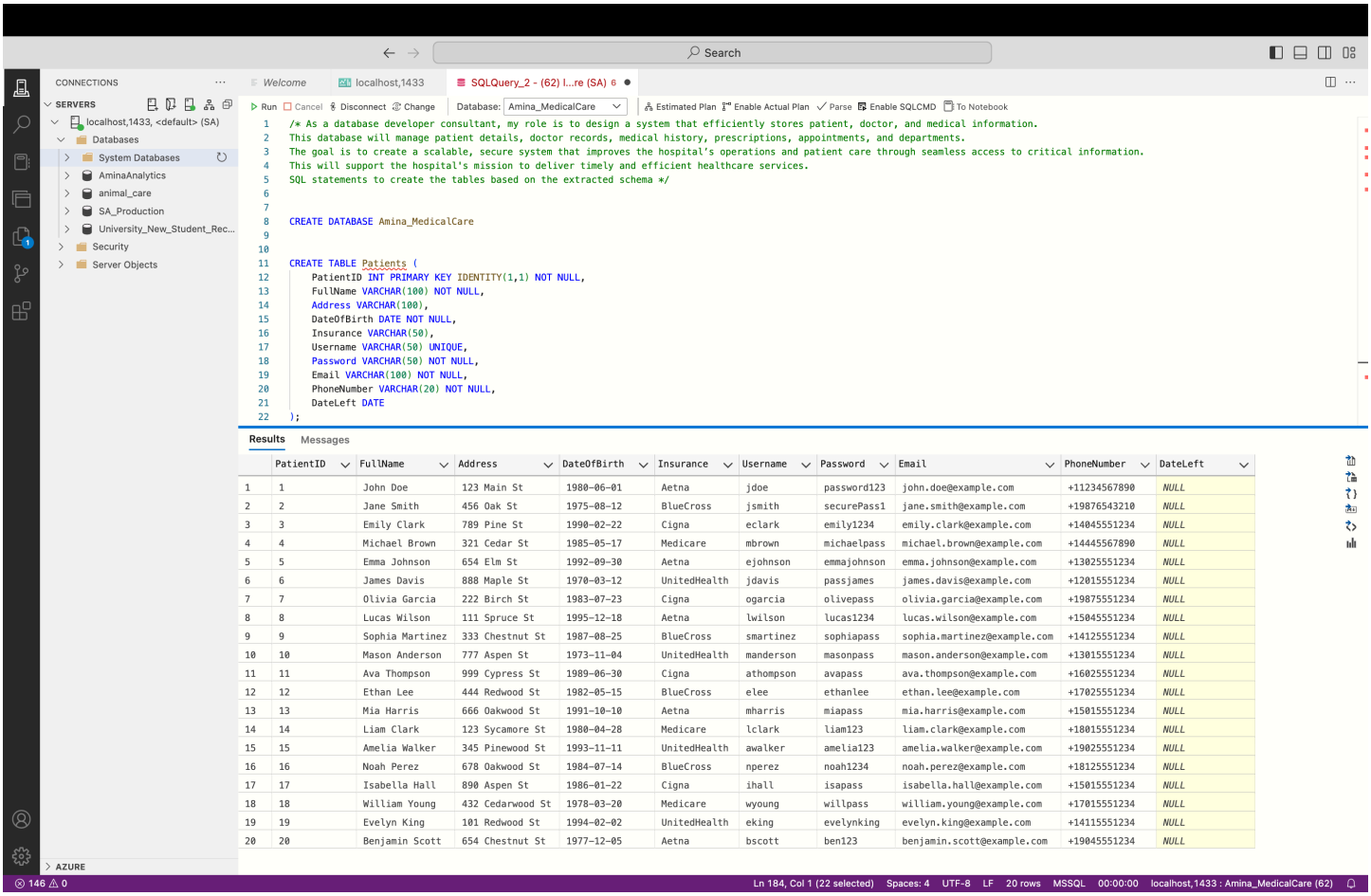


Figure 1: Patient Records Showing Patient Table

2. **DOCTORS**: Stores information on doctors and their specialties.

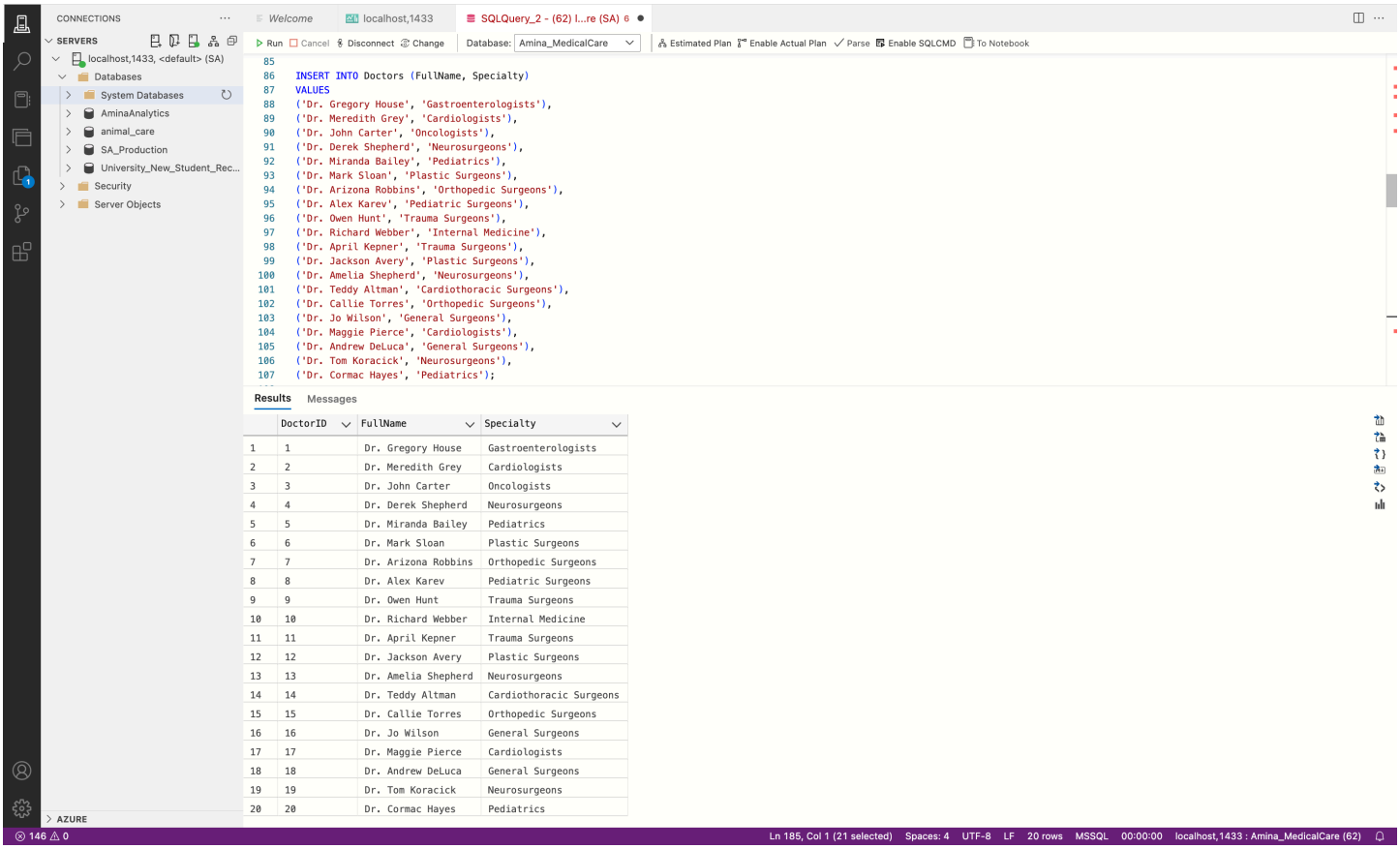
DoctorID: (INT, PRIMARY KEY, NOT NULL): This column uniquely identifies each doctor.
Data Type: INT allows for an efficient primary key.
Constraint: PRIMARY KEY ensures each doctor has a unique identifier.

FullName: Stores the full name of the doctor.
Data Type: VARCHAR (100) allows for a wide range of name lengths.

Specialty: Stores the medical specialty of the doctor, such as "Cardiology" or "Pediatrics."
Data Type: VARCHAR (50) is sufficient for most medical specialty names.

```
CREATE TABLE Doctors (
DoctorID INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
FullName VARCHAR (100) NOT NULL,
Specialty VARCHAR (50) NOT NULL
);
```

Figure 2: Doctors Records Showing Doctors Table



3. **APPOINTMENTS**: Tracks appointments between patients and doctors, including feedback.

Appointment: Unique identifier for each appointment.
Data Type: INT is an efficient choice for primary keys.
Constraint: PRIMARY KEY ensures uniqueness.

PatientID: Links the appointment to the patient.
Data Type: INT refers to the PatientID from the Patients table.
Constraint: NOT NULL and a FOREIGN KEY ensure it corresponds to an existing patient.

DoctorID: Links the appointment to the doctor.
Data Type: INT refers to the DoctorID from the Doctors table.
Constraint: NOT NULL and FOREIGN KEY ensure it references a valid doctor.

Appointment Date: Stores the date of the appointment.
Data Type: DATE stores this information efficiently.
Constraint: The CHECK constraint ensures that appointments cannot be scheduled in the past.

Appointment Time: Captures the time of the appointment.

Data Type: TIME is used to store the specific time of day.

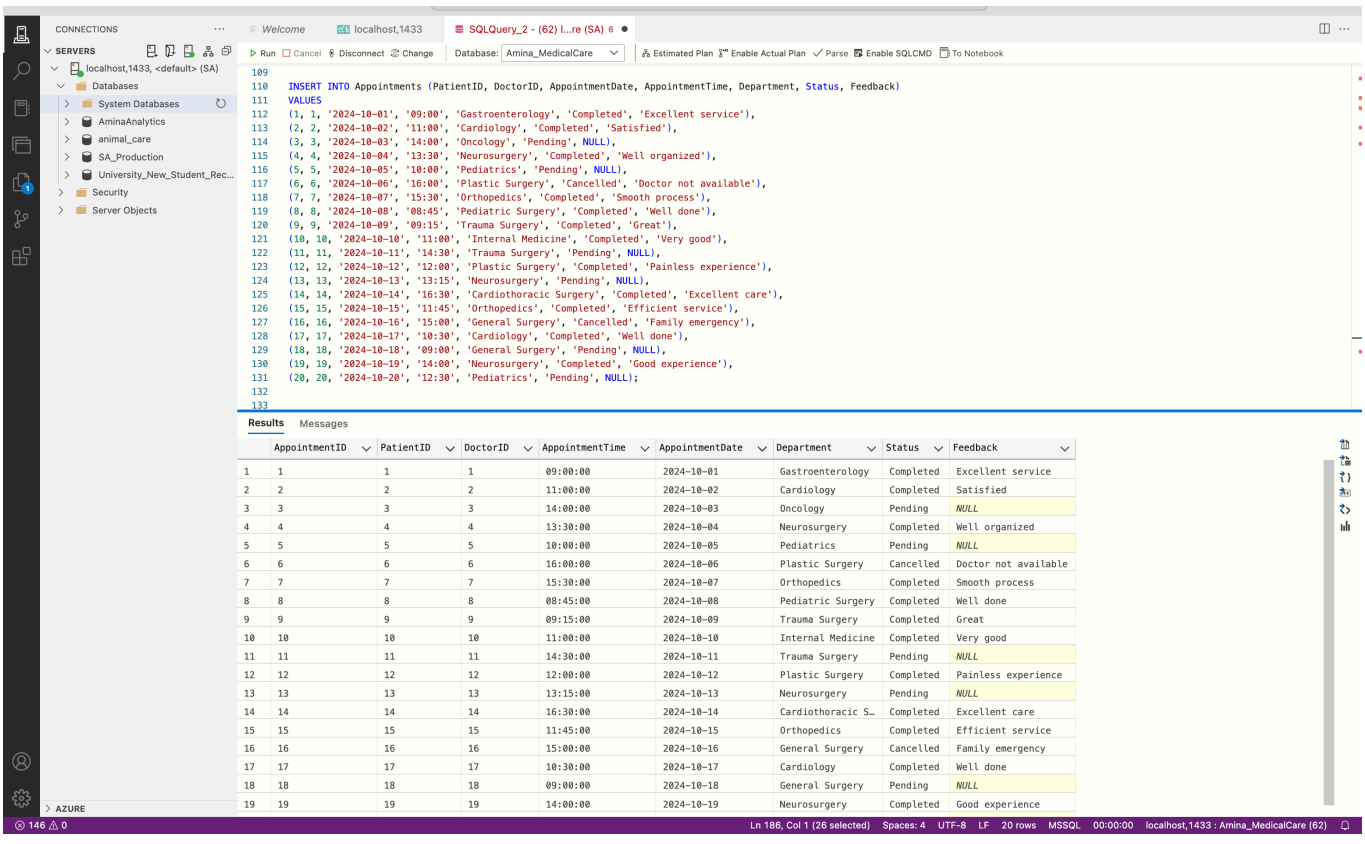
Department: Indicates which department the appointment is for (e.g., Cardiology):
Data Type: is suitable for department names.

Status: Stores the status of the appointment.
Data Type: VARCHAR (20) to store the current appointment state.
Constraint: The CHECK constraint ensures only valid statuses are entered.

Feedback: Allows the patient or doctor to leave feedback on the appointment.
Data Type: VARCHAR (100) provides enough space for detailed comments.

```
CREATE TABLE Appointments (
  AppointmentID INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
  PatientID INT NOT NULL,
  DoctorID INT NOT NULL,
  AppointmentTime TIME,
  AppointmentDate DATE,
  Department VARCHAR (50),
  Status VARCHAR (20) CHECK (Status IN ('Pending', 'Cancelled', 'Completed')),
  Feedback VARCHAR (100),
  FOREIGN KEY (PatientID) REFERENCES Patients (PatientID),
  FOREIGN KEY (DoctorID) REFERENCES Doctors (DoctorID)
);
```

Figure 3: Appointment Records showing Appointment Table



4. **MEDICALRECORDS**: Stores a patient's medical history, diagnosis, and prescribed medicines.

MedicalRecordID (INT, PRIMARY KEY, NOT NULL): Unique identifier for each medical record.
Data Type: INT is used as the primary key for simplicity and efficiency.

PatientID (INT, NOT NULL): Links the medical record to a specific patient.
Data Type: INT refers to the PatientID in the Patients table.
Constraint: FOREIGN KEY ensures it corresponds to an existing patient.

Past Appointments: Lists past appointments for the patient.
Data Type: VARCHAR (100) provides enough space to store appointment summaries.

Diagnoses Stores diagnoses the patient has received.
Data Type: VARCHAR (100) allows for a detailed medical history.

Medicines: Lists medicines the patient has been prescribed.
Data Type: VARCHAR (100) provides sufficient space for multiple medications.

MedicinePrescribedDate: Captures the date when the medicine was prescribed.
Data Type: DATE is appropriate for storing this information.

Allergies: Stores known allergies for the patient.
Data Type: VARCHAR (100) provides enough space for detailed allergy descriptions.

```
CREATE TABLE Medical Records (
 MedicalRecordID INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
 PatientID INT NOT NULL,
 PastAppointments VARCHAR (100),
 Diagnoses VARCHAR (100),
 Medicines VARCHAR (100),
 MedicinePrescribedDate DATE,
 Allergies VARCHAR (100),
 FOREIGN KEY (PatientID) REFERENCES Patients (PatientID)
);
```

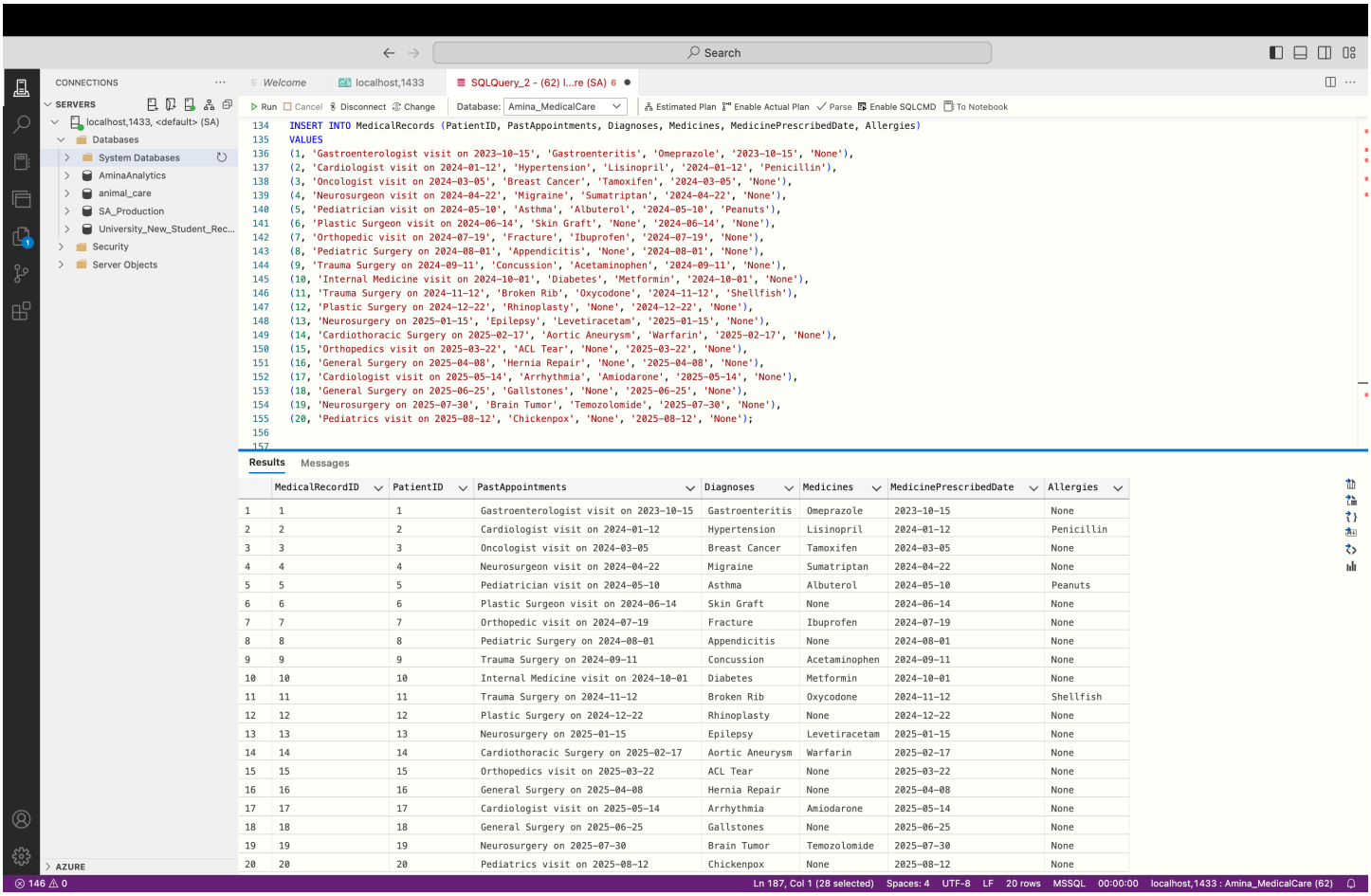


Figure 4: MedicalRecords showing MedicalRecords Table

5. **DEPARTMENTS**: Lists the departments in the hospital.

DepartmentID: Unique identifier for each hospital department
Data Type: INT is efficient for a small, manageable list of departments.

DepartmentName: Stores the name of the department
Data Type: VARCHAR (50) is sufficient for typical department names.

```
CREATE TABLE Departments (
DepartmentID INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
DepartmentName VARCHAR (50)
);
```

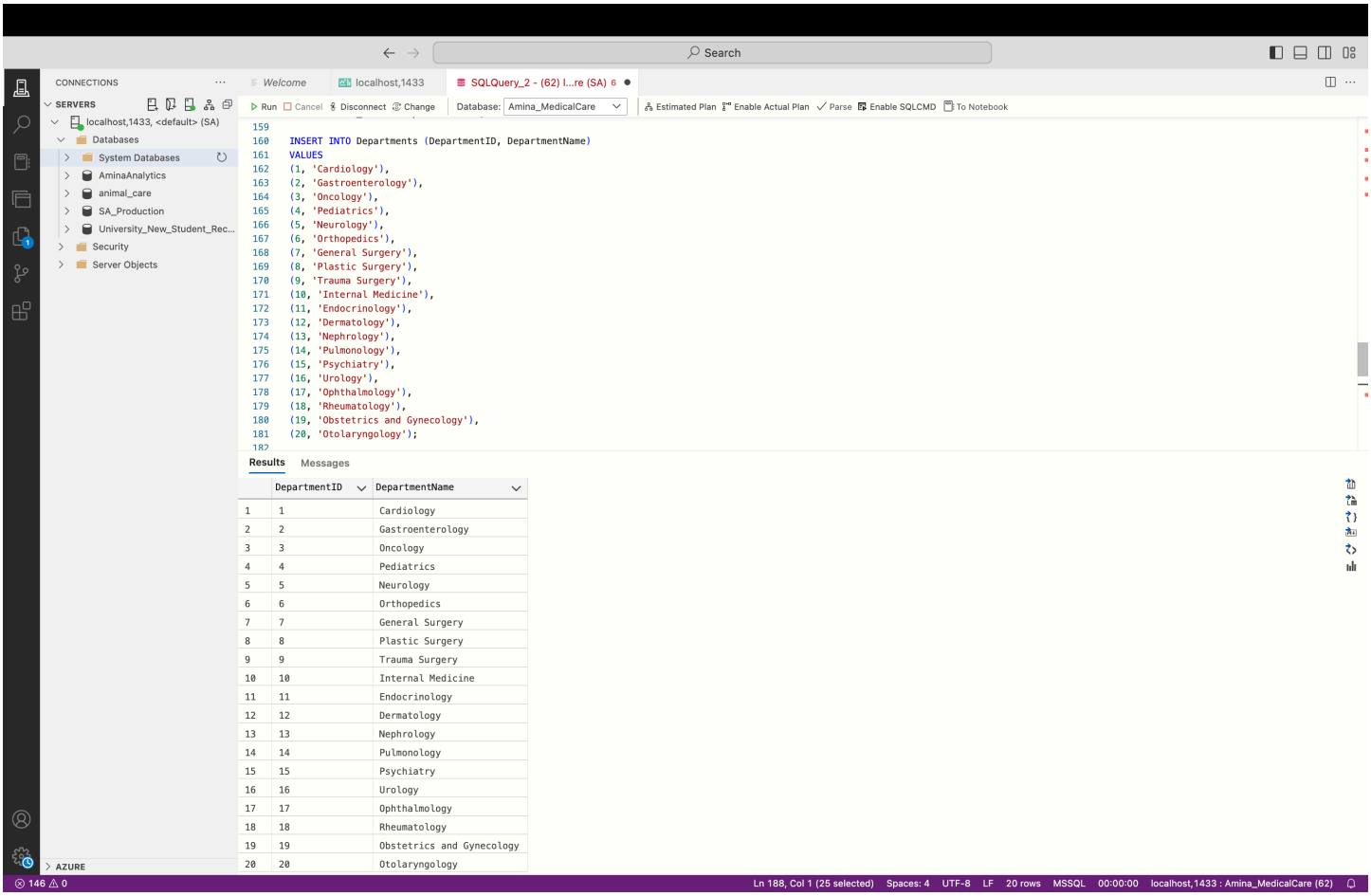


Figure 5: Department Records showing Department Table

-- ADDING CONSTRAINT TO CHECK APPOINTMENT DATE NOT IN THE PAST

-- This constraint ensures appointments are only scheduled for the current or future dates.

```
ALTER TABLE Appointments

ADD CONSTRAINT chk_AppointmentDate CHECK (AppointmentDate >= CAST (GETDATE () AS DATE));
```

-- QUERY: PATIENTS OLDER THAN 40 WITH CANCER DIAGNOSIS

```
SELECT FullName, DateOfBirth, Diagnoses

FROM Patients P

JOIN MedicalRecords MR ON P.PatientID = MR.PatientID

WHERE Diagnoses LIKE '%Cancer%' AND DATEDIFF(YEAR, P.DateOfBirth, GETDATE()) > 40;
```

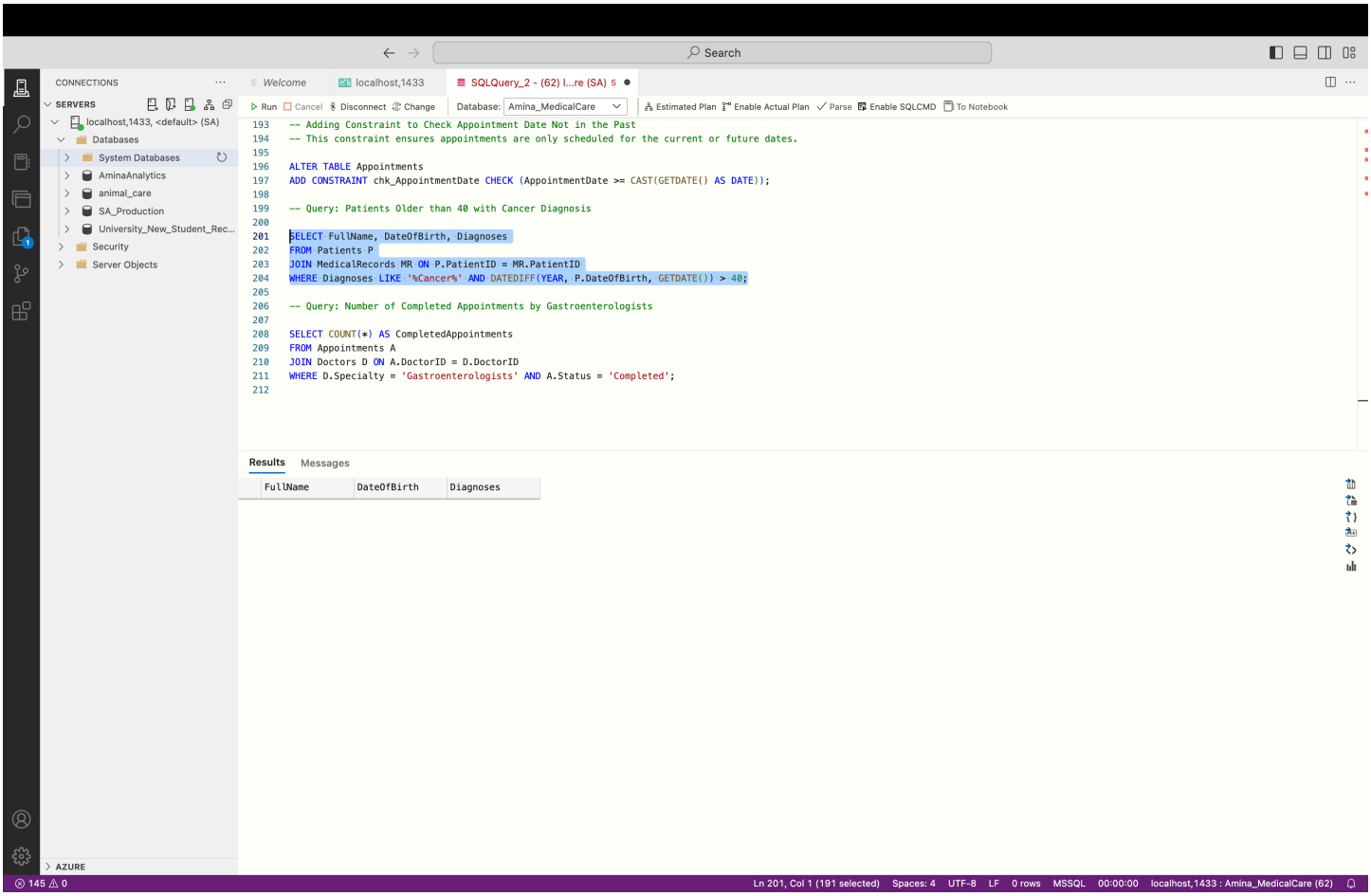


Figure 6: Patients Older than 40 with Cancer = 0

-- QUERY: NUMBER OF COMPLETED APPOINTMENTS BY GASTROENTEROLOGISTS

SELECT COUNT(*) AS CompletedAppointments

FROM Appointments A

JOIN Doctors D ON A.DoctorID = D.DoctorID

WHERE D.Specialty = 'Gastroenterologists' AND A.Status = 'Completed';

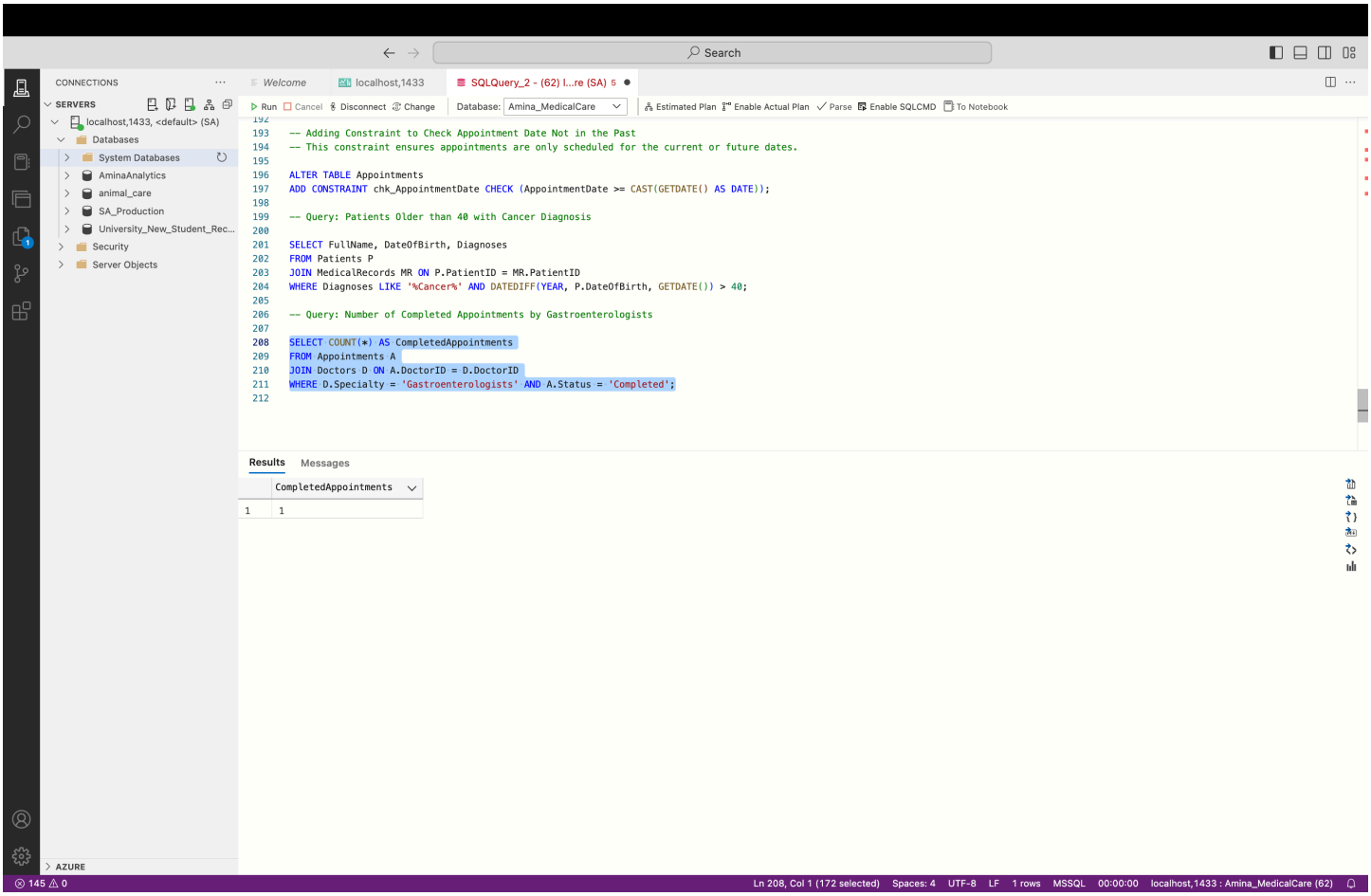


Figure 7: No. of completed Appointments with Gastroenterologists

NORMALIZATION

The database is designed in **Third Normal Form (3NF)** to eliminate redundancy, improve data integrity, and optimize queries. This involves:

- Removing repeating groups and ensuring atomicity in attributes (1NF).
- Eliminating partial dependencies by ensuring non-key attributes depend on the primary key (2NF).
- Removing transitive dependencies to ensure that all attributes are dependent solely on the primary key (3NF).

ERD (ENTITY-RELATIONSHIP DIAGRAM)

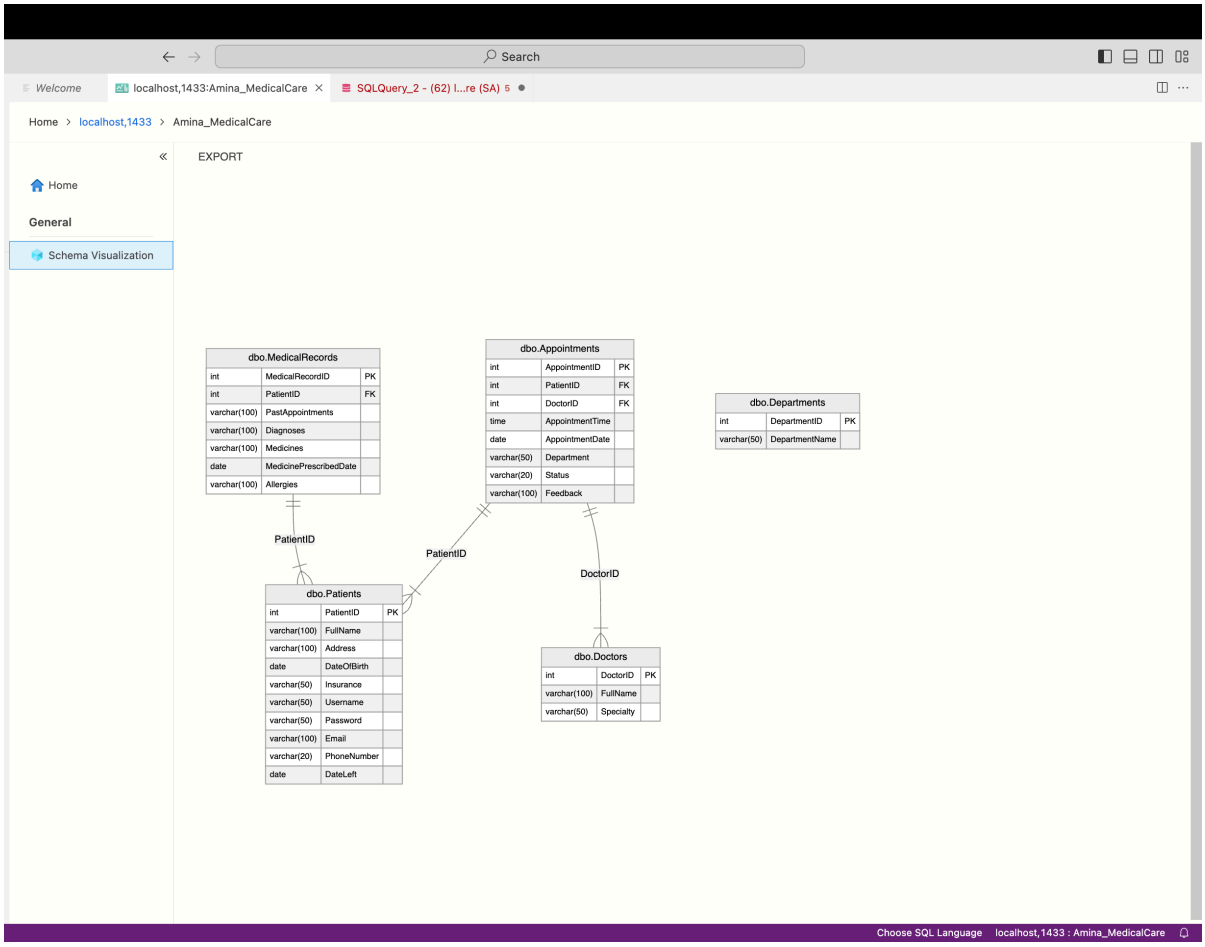


Figure 8: EDR visualization of all tables

CONCLUSION

In summary, the process of designing, creating, and populating the hospital database involved key steps in establishing a structured, normalized schema to ensure data integrity and optimize query performance. By creating tables for patients, doctors, appointments, departments, and medical records, we were able to efficiently manage essential data for a hospital setting. The constraints applied, such as foreign keys and check constraints, ensure the database maintains valid relationships and restricts invalid data entries.

The queries developed allow for practical use of the database, such as ensuring no appointment is scheduled in the past, identifying patients with specific conditions, and tracking completed appointments by specific doctor specialties. Through this implementation, the database is set up to support accurate reporting and data analysis, contributing to improved hospital operations and patient care.

The design and execution of this database also emphasize the importance of proper data types and constraints in ensuring efficiency, accuracy, and scalability.