University of Salford
MANCHESTER

University of Salford
Advanced Databases

MICHAEL IFECHUKWU OLU
@00755387

Task 1 Report

Introduction

This task involves developing a database system for a hospital. A variety of hospital-related data, such as patient biodata and portal information, doctor's data, medical records, appointments, and departmental information, should be able to be stored in this database. I have to create tables with the right columns and constraints for patients, doctors, medical records, appointments, departments, and reviews based on the specifications.

Additionally, this database will be improved with a variety of triggers, user-defined functions, and stored procedures to ensure optimal accuracy in data integrity and database management according to the client's requirements. The purpose of this work is to create a well-designed database which will satisfy the hospital's requirements for handling data related to day-to-day operations.

We will be making use of Microsoft's SQL Server Management Studio to develop and test this database. The code will be written in TSQL statements and our tables will be in their third normal form (3NF) to reduce redundancy and maintain data integrity.

Part 1

1.1 Database design

The first step to creating the database is to plan and design the proposed schema of the database. Below is a list of entities and database objects which we will create according to the requirements of the hospital.

1.1.1 Entities:

- Departments
- Doctors
- PatientsBioData
- PatientsPortal
- Appointments
- PastAppointments
- Diagnosis
- Medicines
- MedicinesPrescribed
- Allergies
- Reviews

1.1.2 Explanations:

Departments: This table will store all departments in the hospital where various doctors belong to.

Doctors: This table will store Bio information about each doctor and their specialties.

PatientsBioData: This table will store bio information about all patients registered to the hospital.

PatientsPortal: This table will store information about a patient's portal account including account credentials, insurance numbers and contact information.

Appointments: This table holds records about appointments which have been booked by a patient to see a doctor.

PastAppointments: This table holds records about past appointments which have been completed and moved from the appointments table.

Diagnosis: This table contains diagnosis for a patient by a doctor after an appointment.

Medicines: This table stores records of the medicines in the inventory.

Allergies: This table stores information about patient allergies.

MedicinesPrescribed: This table stores information about medicines prescribed by a doctor to a patient after an appointment.

1.1.3 Entity Relationship Diagram

The diagram below shows our database plan and the proposed relationships between the entities in our database.

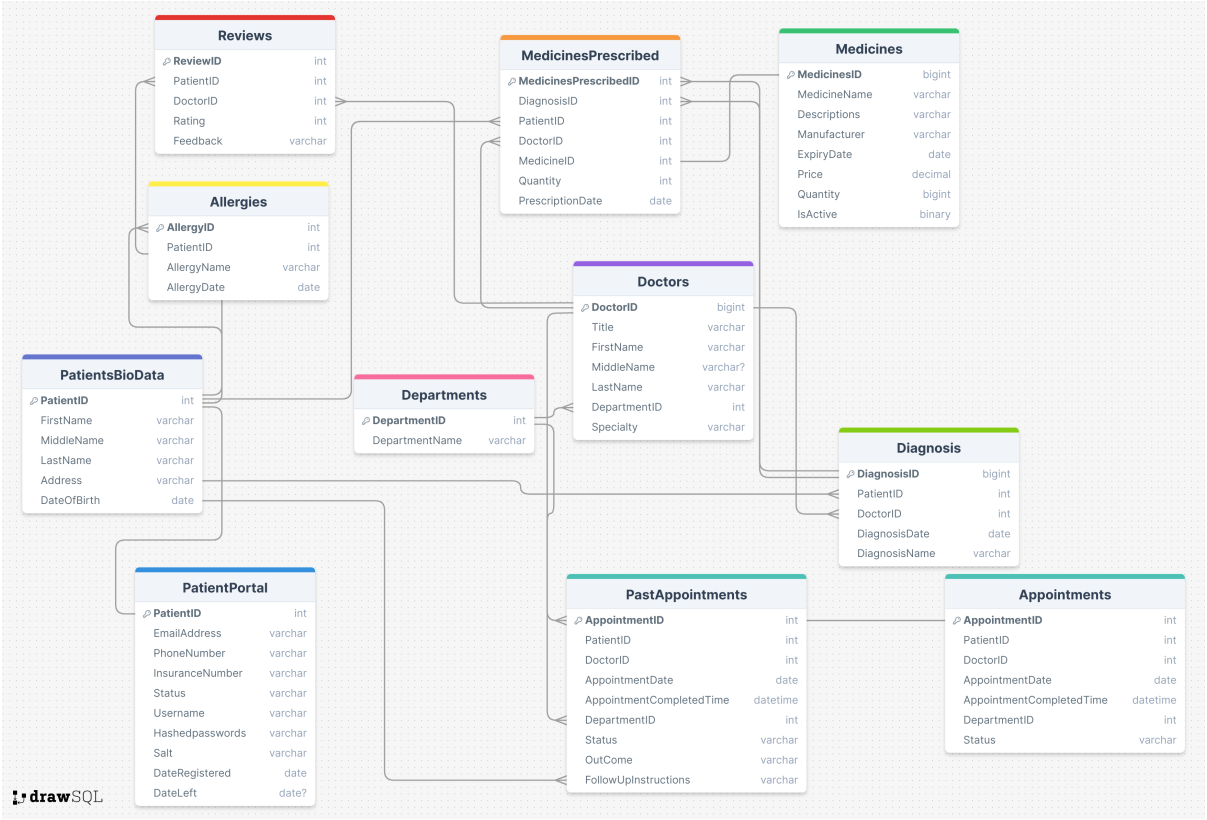


Figure 1

Our entity relationship diagram shows that all tables has been normalized in 3NF and has eliminated all transitive dependencies because no non-key attribute depends on another non-key attribute in each of the tables as seen in figure 1.

1.1.4 Table Schema

In this section, we are going to go through the table schema of our database along with the datatypes of their attributes.

Departments Table

Attributes	Datatype	Justification
DepartmentID	INT	Integer to be used along with auto incrementing ID.
FD	VARCHAR	Name values are always strings

Doctors Table

Attributes	Datatype	Justification
DoctorID	INT	Integer to be used along with auto incrementing ID.

Title	VARCHAR	Title values are always strings
FirstName, MiddleName, LastName	VARCHAR	Name values are always strings
DepartmentID	INT	To match the primary key's datatype of referenced table
Specialty	VARCHAR	Specialties are described with words/strings

PatientsBioData Table

Attributes	Datatype	Justification
PatientID	INT	Integer to be used along with auto incrementing ID.
FirstName, MiddleName, LastName	VARCHAR	Name values are always strings
Address	VARCHAR	Address values are always strings
DateOfBirth	DATE	Date of birth should be stored in the correct date format for consistency

di

PatientPortal Table

Attributes	Datatype	Justification
PatientID	INT	Integer to be used along with auto incrementing ID.
EmailAddress	NVARCHAR	Because email might contain foreign characters.
PhoneNumber	VARCHAR	To allow phone numbers stored with country code.
InsuranceNumber	VARCHAR	For alphanumeric insurance numbers
Status	VARCHAR	Stored as a string
Username	VARCHAR	Stored as a string
HashedPasswords	VARCHAR	Hashed value will be stored as a string
Salt	VARCHAR	String to be used as a key for HashedPasswords to authenticate passwords
DateRegistered	DATE	To be stored in correct date format for consistency.
DateLeft	DATE	To be stored in correct date format for consistency.

Appointments Table

Attributes	Datatype	Justification
AppointmentID	INT	Integer to be used along with auto incrementing ID.

PatientID	INT	To match the primary key's datatype of referenced table
DoctorID	INT	To match the primary key's datatype of referenced table
AppointmentDate	DATE	To be stored in correct date format for consistency
AppointmentCompletedTime	TIME	To be stored in correct time format for consistency
DepartmentID	INT	To match the primary key's datatype of referenced table
Status	VARCHAR	To store status as a string. Example: 'Pending' 'Completed', 'Available'

PastAppointments Table

Attributes	**Datatype**	**Justification**
PastAppointmentID	INT	Integer to be used along with auto incrementing ID.
PatientID	INT	To match the primary key's datatype of referenced table
DoctorID	INT	To match the primary key's datatype of referenced table
AppointmentDate	DATE	To be stored in correct date format for consistency
AppointmentCompletedTime	TIME	To be stored in correct time format for consistency
DepartmentID	INT	To match the primary key's datatype of referenced table
Status	VARCHAR	To store status as a string. Example: 'Pending' 'Completed', 'Available'
OutCome	VARCHAR	To store appointment outcome as a string
FollowUpInstructions	VARCHAR	Instructions are written in words(strings)

Diagnosis Table

Attributes	**Datatype**	**Justification**
DiagnosisID	INT	Integer to be used along with auto incrementing ID.
PatientID	INT	To match the primary key's datatype of referenced table
DoctorID	INT	To match the primary key's datatype of referenced table
DiagnosisDate	DATE	To store date in the correct format
DiagnosisName	VARCHAR	For storing as stings.

Medicines Table

Attributes	Datatype	Justification
MedicineID	INT	Integer to be used along with auto incrementing ID.
MedicineName	VARCHAR	Medicine names are words (strings)
Descriptions	VARCHAR	Descriptions are in words (strings)
Manufacturer	VARCHAR	Names are in words (string)
ExpiryDate	DATE	To store date in the correct format
Price	DECIMAL	To store currency value with lower denominations
Quantity	INT	To store in numbers
IsActive	BIT	1 or 0 to represent true or false

Allergies Table

Attributes	Datatype	Justification
AllergyID	INT	Integer to be used along with auto incrementing ID.
PatientID	INT	To match the primary key's datatype of referenced table
AllergyName	VARCHAR	Names are stored in words(strings)
AllergyDate	DATE	To store date in correct format

MedicinesPrescribed Table

Attributes	Datatype	Justification
MedicinesPrescribedID	INT	Integer to be used along with auto incrementing ID.
DiagnosisID	INT	To match the primary key's datatype of referenced table.
PatientID	INT	To match the primary key's datatype of referenced table.
DoctorID	INT	To match the primary key's datatype of referenced table.
MedicineID	INT	To match the primary key's datatype of referenced table.
Quantity	INT	For storing quantities in numbers
PrescriptionDate	DATE	To store dates in correct format

Reviews Table

Attributes	Datatype	Justification
ReviewID	INT	Integer to be used along with auto incrementing ID.

PatientID	INT	To match the primary key's datatype of referenced table.
DoctorID	INT	To match the primary key's datatype of referenced table.
Rating	INT	To store ratings in integers (1-5)
Feedback	VARCHAR	To store feedback in words (strings)

1.1.5 Database Objects

- RegisterPatient (stored procedure)
- BookAppointment (stored procedure)
- CancelAppointment (stored procedure)
- RebookAppointment (stored procedure)
- PendingAppointmentsView(View)
- GetPatientMedicalRecord (user defined function)
- UpdateMedicalRecords (stored procedure)
- CompleteAppointmentAndLeaveReview (stored procedure)
- DeregisterPatient (stored procedure)
- UpdateDateLeftOnInactive (trigger)

This is the general schema of our database, in the next section, we will go ahead and create the Database, tables and database objects.

1.2 Table Creation

Using the database design which we covered in the previous section, we will create tables using our proposed schema using TSQL statements in the Microsoft SQL server management studio.
First, we need to connect to our SQL server and create a new query to be able to write our SQL.

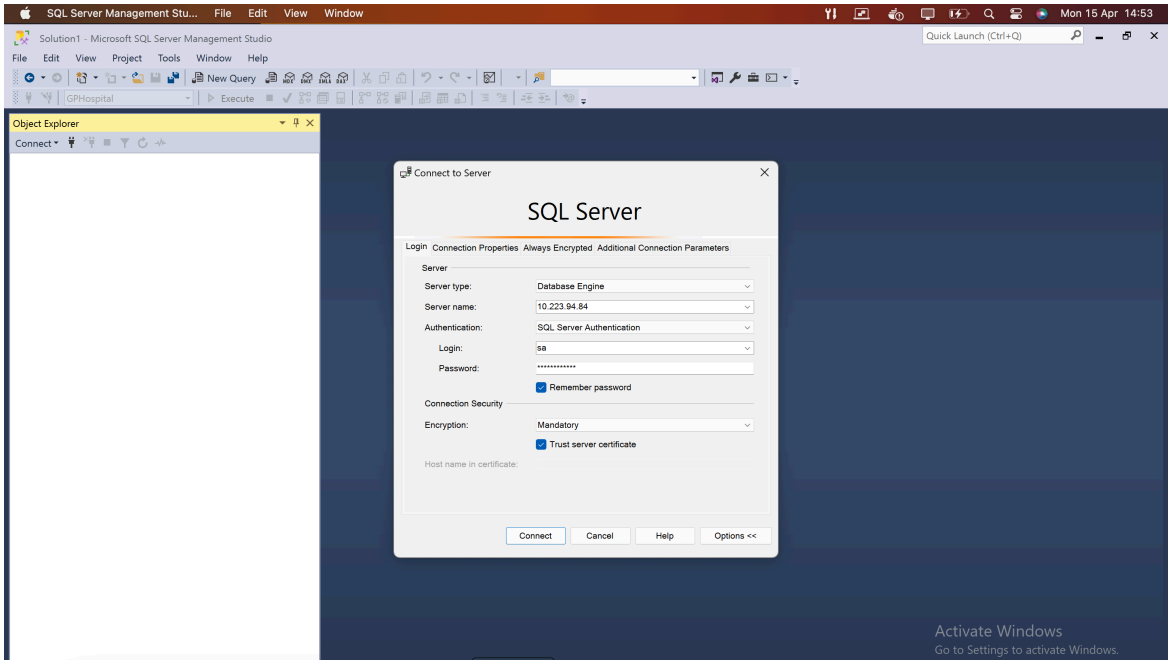


Figure 2

After inputting our credentials and selected the appropriate parameters, we can click the connect button to connect to our SQL server. After the server has been connected to, the server will appear on the left-hand side of our window as shown in figure 3 below.

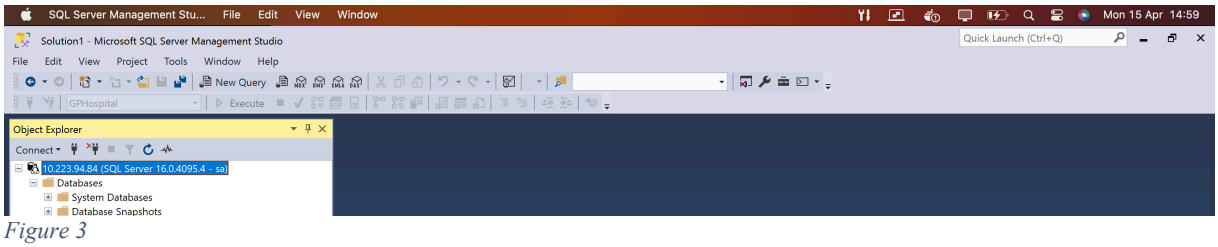


Figure 3

Our Server was created with the IP address specified. We can now open a new query and start writing our SQL.

1.2.1 New Database creation

Before our tables are created, we need to create a database to store our tables. The SQL query below creates a database called **GPHospital** if it does not already exist in our server.

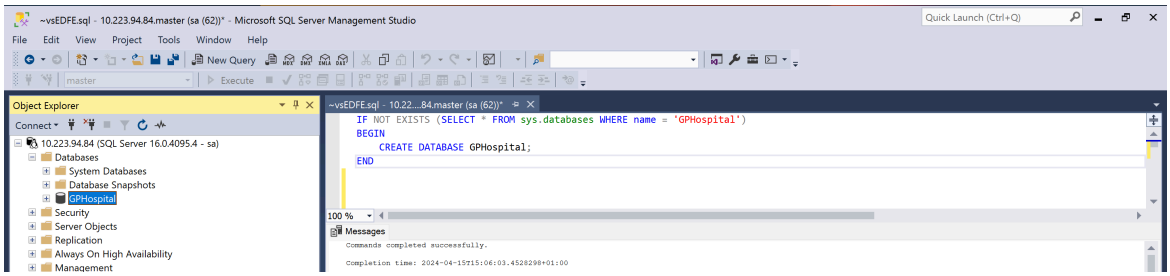


Figure 4

We ran our query and our database was successfully created as seen in figure 4 above.
We can proceed to the next step which is the creation of our tables according to our database design in section **1.1** inside the **GPHospital** database as seen in figure 5.

Departments Table

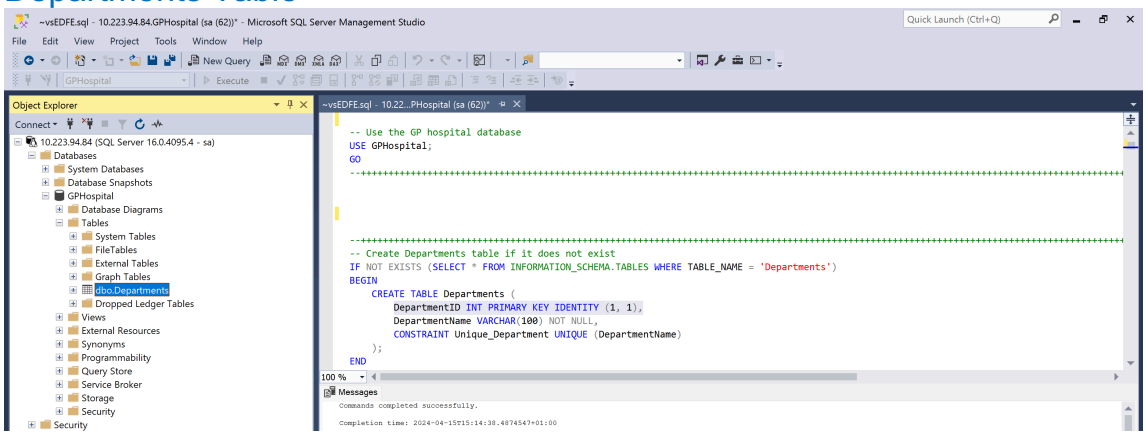


Figure 5

The Departments table has been created with 2 attributes: **DepartmentID** and DepartmentName with the **DepartmentID** as primary key, which is set to start from 1 and auto increment with 1 as new records are inserted. A constraint **Unique_Department** was also applied to the table in order to prevent duplicate records, thereby reducing data redundancy and maintaining data integrity.

Doctors Table

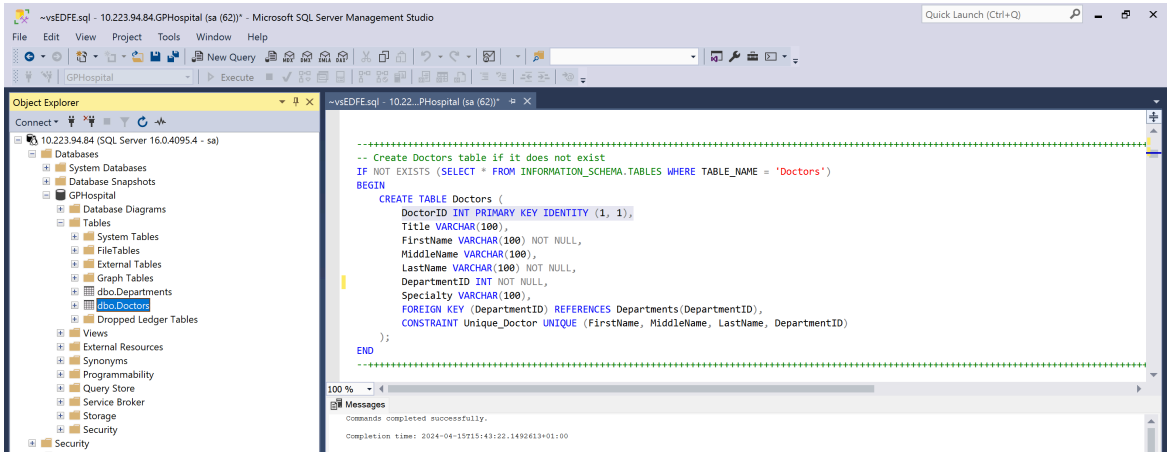


Figure 6

The Doctors table has been created with 6 attributes with **DoctorID** as primary key which auto increments by 1 on new insertions. The attribute **DepartmentID** is a foreign key which references the Departments table to form a many-to-one(N:1) relationship with the department table. A unique constraint **Unique_Doctor** was applied to the table which uses a combination of **FirstName**, **MiddleName**, **LastName** and **DepartmentID** to determine the uniqueness of each record in the table.

PatientsBioData Table

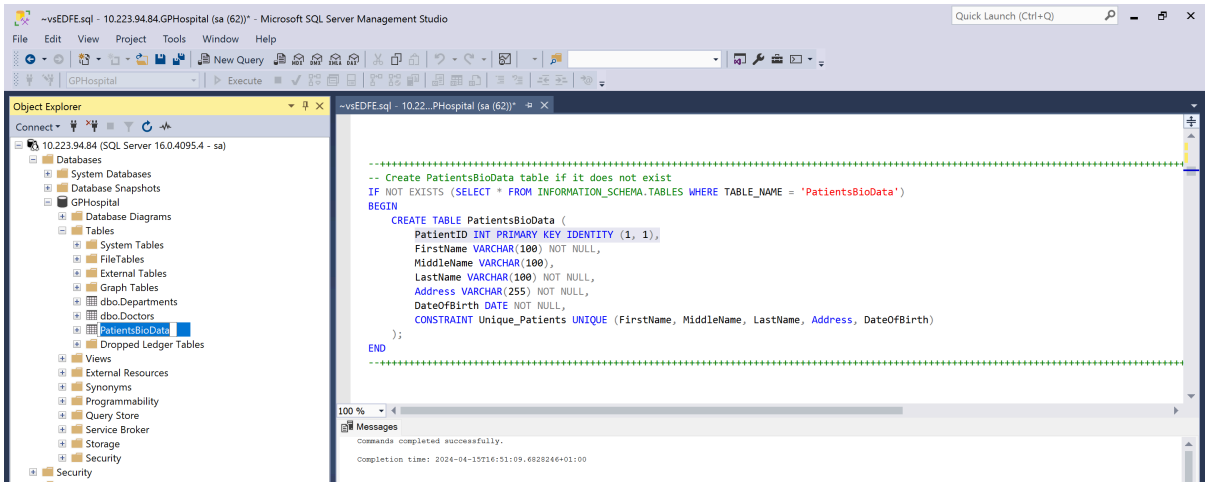


Figure 7

The patients have two tables, one for biodata and another for the portal information. This table in figure 7contains 6 attributes for the patients Biodata with **PatientID** as the auto incrementing primary key. It also has a unique constraint applied to which uses a combination of the **FirstName**, **MiddleName**, **LastName**, **Address**, and **DateOfBirth** to correctly determine unique records.

PatientsPortal Table

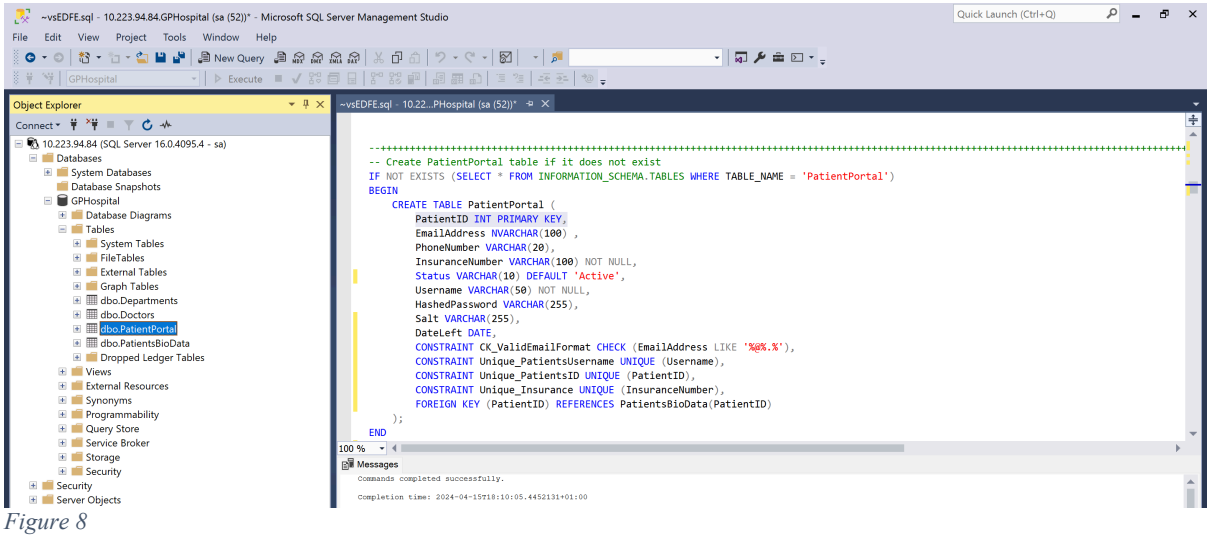


Figure 8

This table is closely associated with the PatientsBioData table, containing 9 attributes with PatientID as the primary key, establishing a one-to-one (1:1)

relationship with PatientsBioData. It includes various constraints, such as a check constraint to validate email addresses upon insertion. Additionally, unique constraints are enforced on Username, PatientID, and InsuranceNumber to prevent duplicates, ensuring each patient has a unique InsuranceNumber, PatientID, and username for their portal account.

Unique constraints are not applied to emails and phone numbers to accommodate for scenarios where multiple patients might share the same contact information. For example, a mother and her baby registered together in a hospital might share contact details.

The "HashedPasswords" attribute is designated for storing encrypted passwords to maintain data privacy. During authentication, passwords are validated by hashing the provided password with the Salt attribute. If the resulting hash matches the stored HashedPassword attribute, the password is authenticated; otherwise, it is rejected as incorrect.

Appointments Table

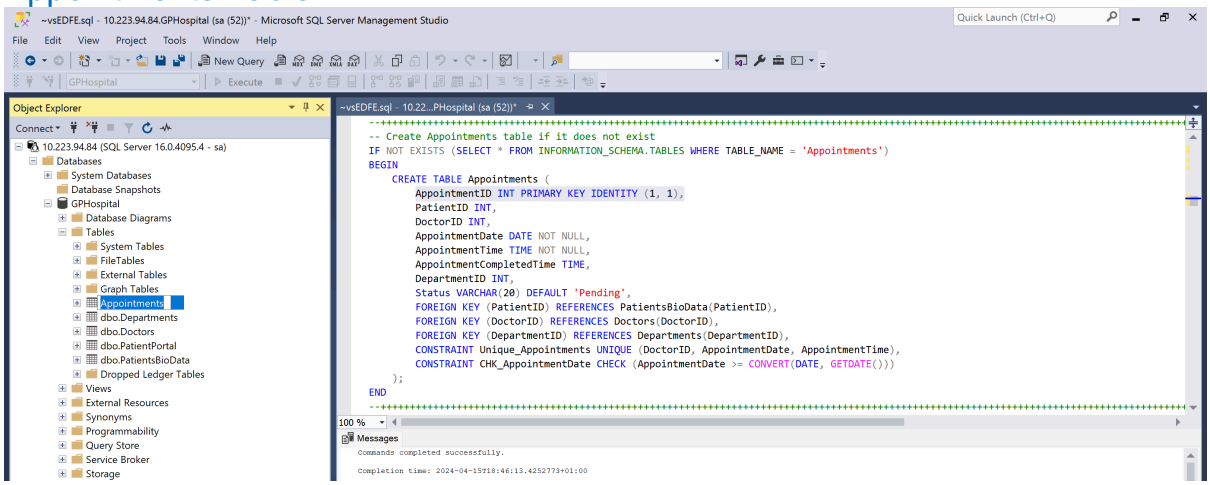


Figure 9

The Appointment table has 8 attributes, with AppointmentID as the primary key. It also has foreign keys for PatientsBioData, Doctors, and DepartmentID. A unique constraint uses DoctorID, AppointmentDate, and AppointmentTime to create unique records, preventing clashing appointments with a specific doctor.

The CHECK constraint **CHK_AppointmentDate** was also applied to the table to ensure that past dates and time as **AppointmentDate** cannot be inserted into the table.

PastAppointments Table

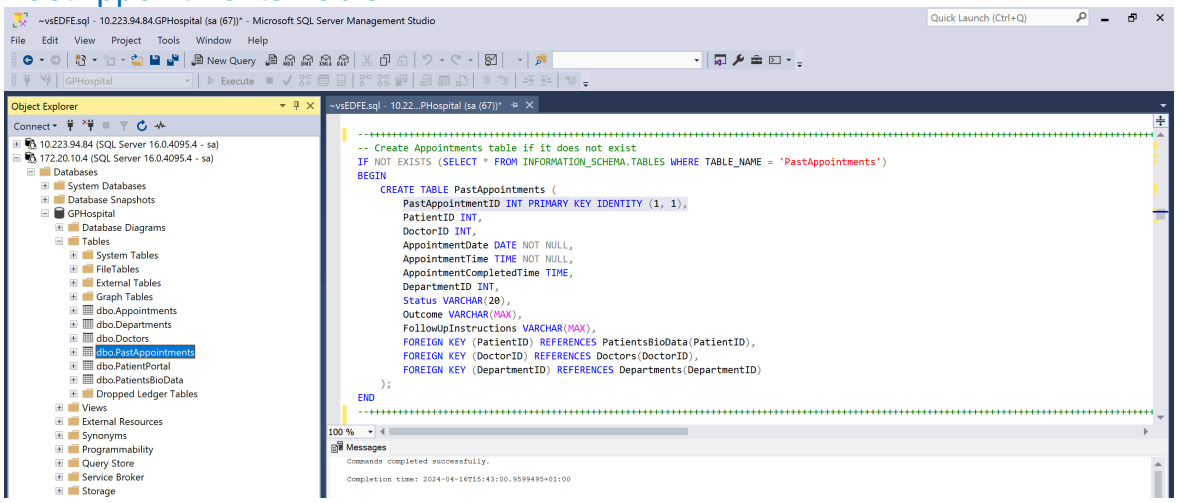


Figure 10

This Table is closely related to the Appointment table. All appointments which have been completed will be moved to this table as part of the medical records. It has **PastAppointmentID** as primary key, and has **PatientID**, **DoctorID** and **DepartmentID** as foreign keys.

Diagnosis Table

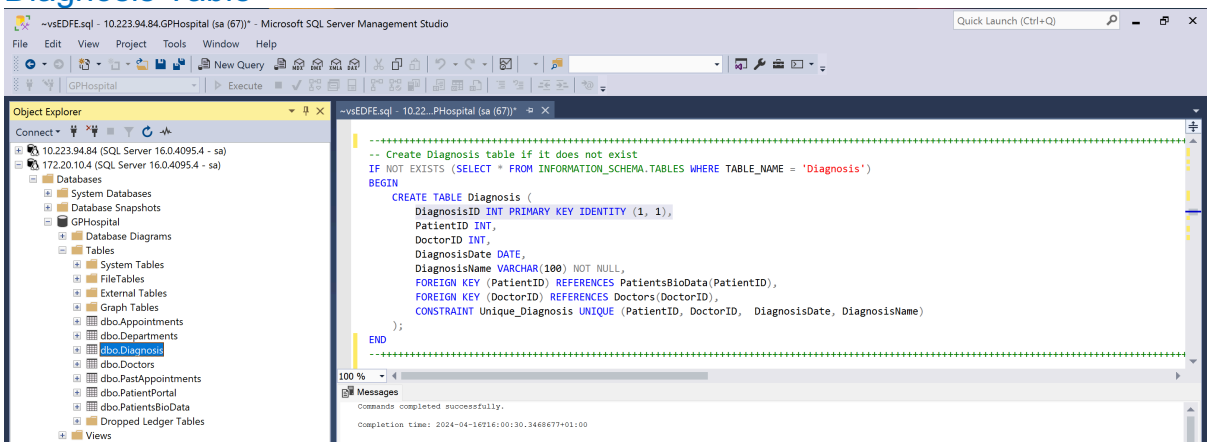


Figure 11

The Diagnosis table stores patient diagnosis information post-appointment, with DiagnosisID as primary key. PatientID and DoctorID reference PatientBioData and Doctors tables respectively. A unique constraint 'Unique_Diagnosis' prevents duplicate records for data integrity.

Medicines Table

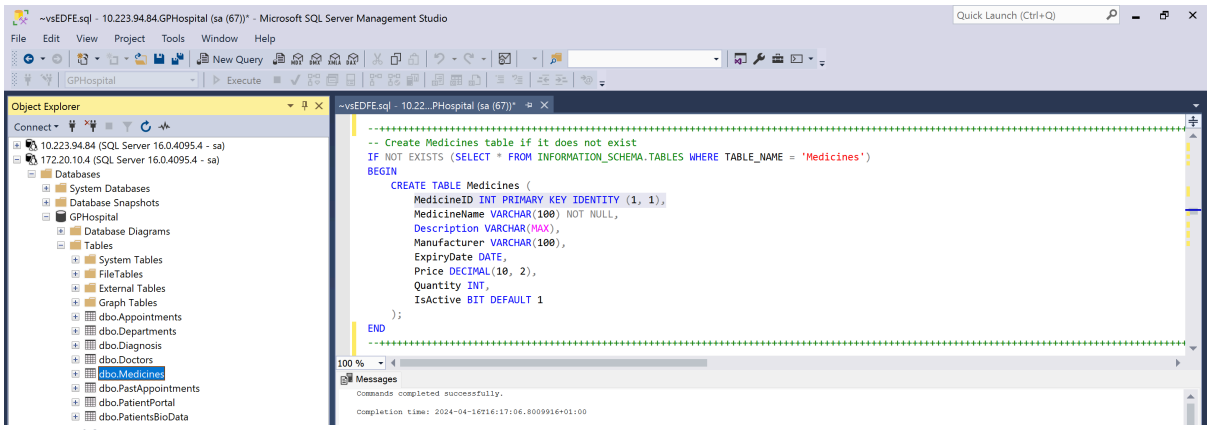
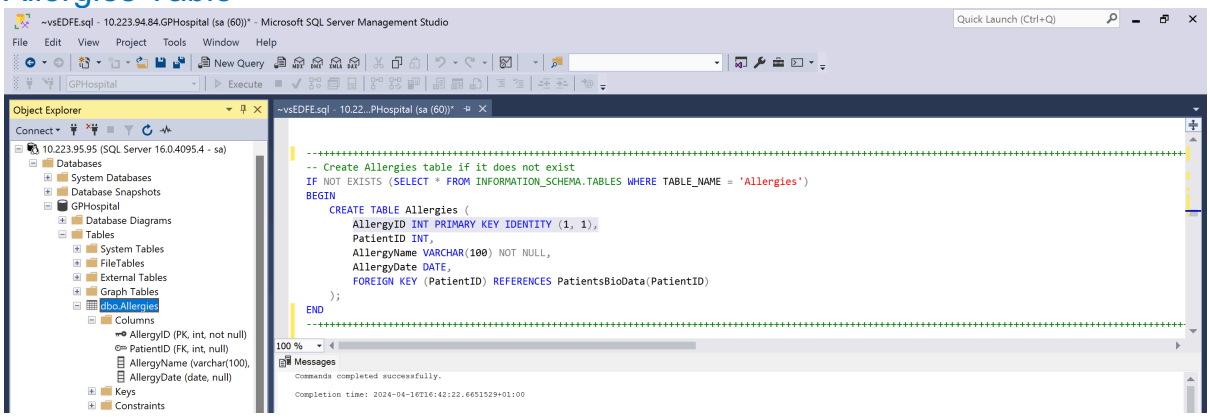


Figure 12

The Medicines table contains records of medicines available in the hospitals inventory. It has the **MedicineID** attribute as the primary key which auto increments by 1 on new insertions.

Allergies Table



Allergies table holds records for the allergies which will be recorded after an appointment between the Patient and the doctor. The PatientID attribute referenced the PatientsBioData table with a many-to-one relationship because a patient can have several allergies.

MedicinesPrescribed Table

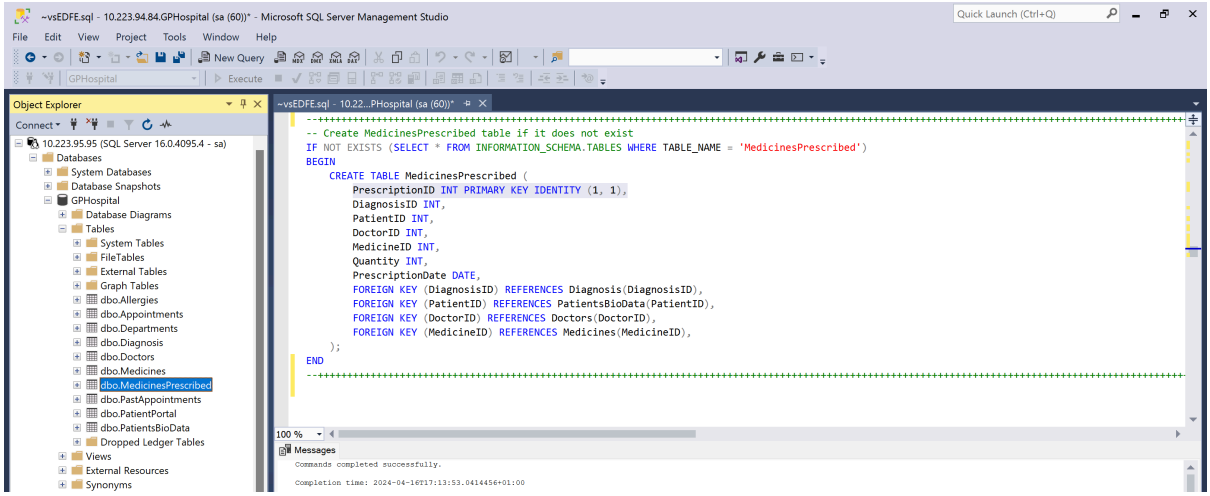


Figure 13

MedicinesPrescribed table holds records for medicines which were prescribed by a doctor to a patient during an appointment. It has the PrescriptionID as the primary key, DiagnosisID attribute references Diagnosis table with a many-to-one(N:1) relationship with the table, PatientID attribute references PatientBioData table with a many-to-one(N:1) relationship with the table, DoctorID attribute references Doctors table with a many-to-one(N:1) relationship with the table.

Reviews Table

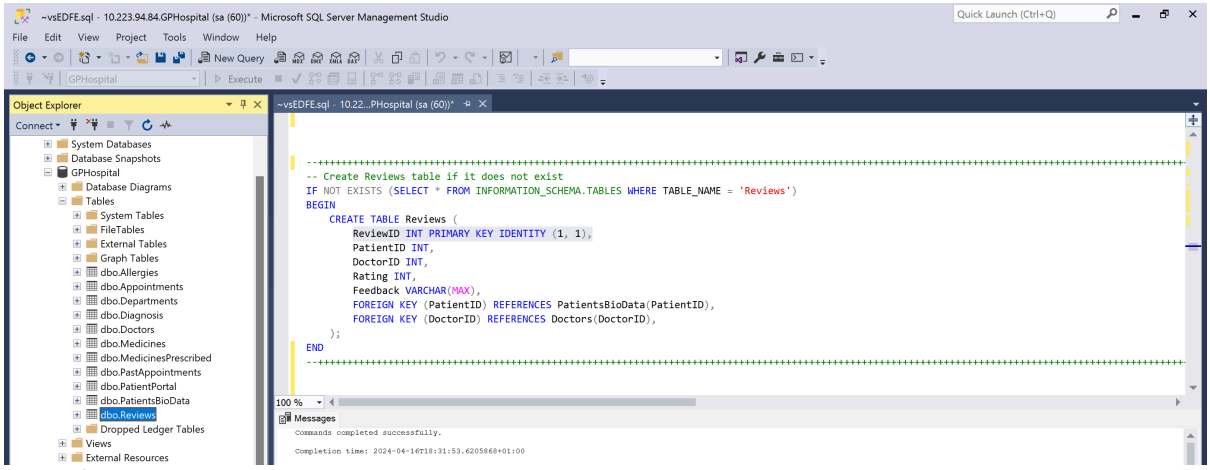


Figure 14

Reviews table keeps records for the reviews made by a patient to a doctor. The ReviewID is the auto incrementing primary key, PatientID attribute references the PatientBioData table with a many-to-one (N:1) relationship, because some patients can have many reviews. The DoctorID references the Doctors table with a many-to-one(N:1) relationship as well because a many reviews can be made for one doctor.

1.3 CREATING USER AND GRANTING PRIVILEGES

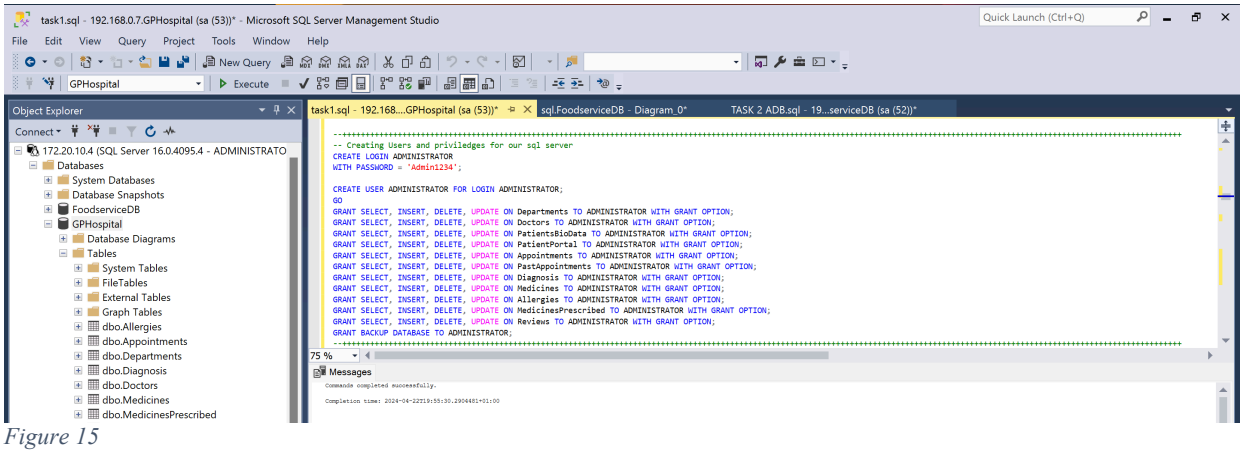


Figure 15

For security reasons, I have created a database user and granted some privileges to database access with an option to grant other users those privileges. This is to ensure that only the database administrator has complete access to the database and will be able to grant access to other trusted parties.

1.4 Database Diagram

The database design involves creating entities, attributes, relationships, keys, and constraints, resulting in an Entity Relationship (ER) Diagram. This diagram provides a clear visual representation of the database's appearance and its relationships, as illustrated in the previous section.

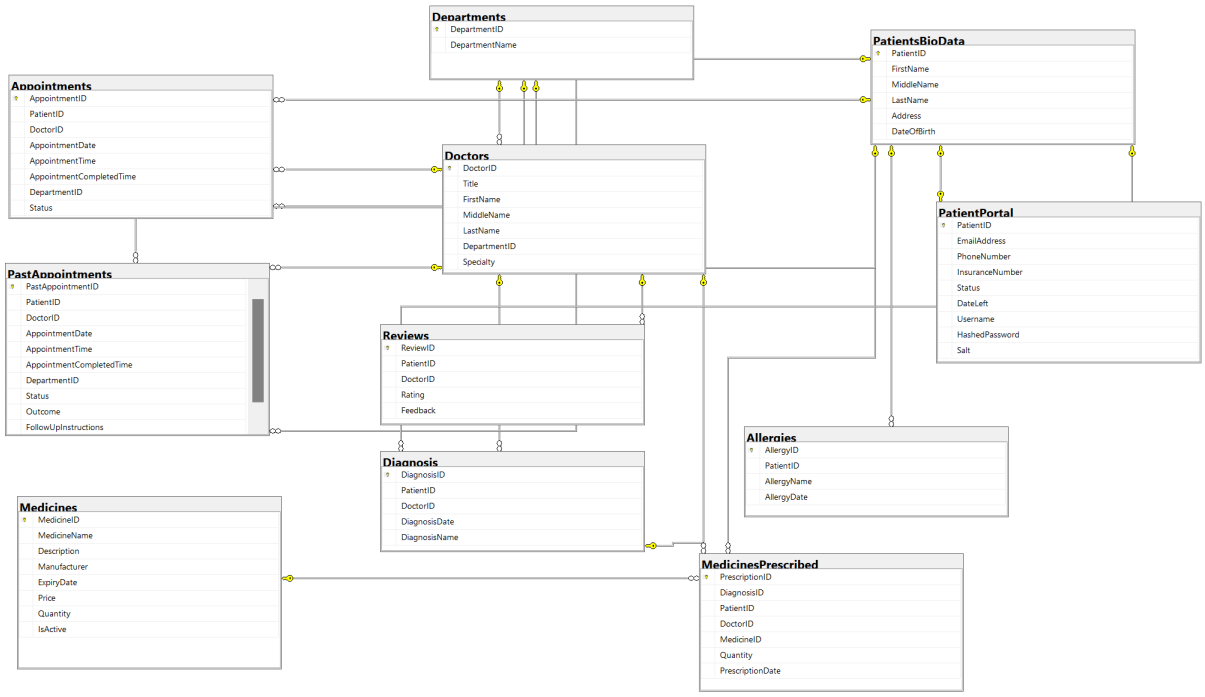


Figure 16

This diagram was generated from the Microsoft SQL server management studio directly from our tables, using its attributes, primary keys and foreign keys to show us what our actual database

1.5 Database Objects

The GPHospital database has been enhanced with the creation of several database objects, including stored procedures, triggers, and user-defined functions.

RegisterPatient (stored procedure)

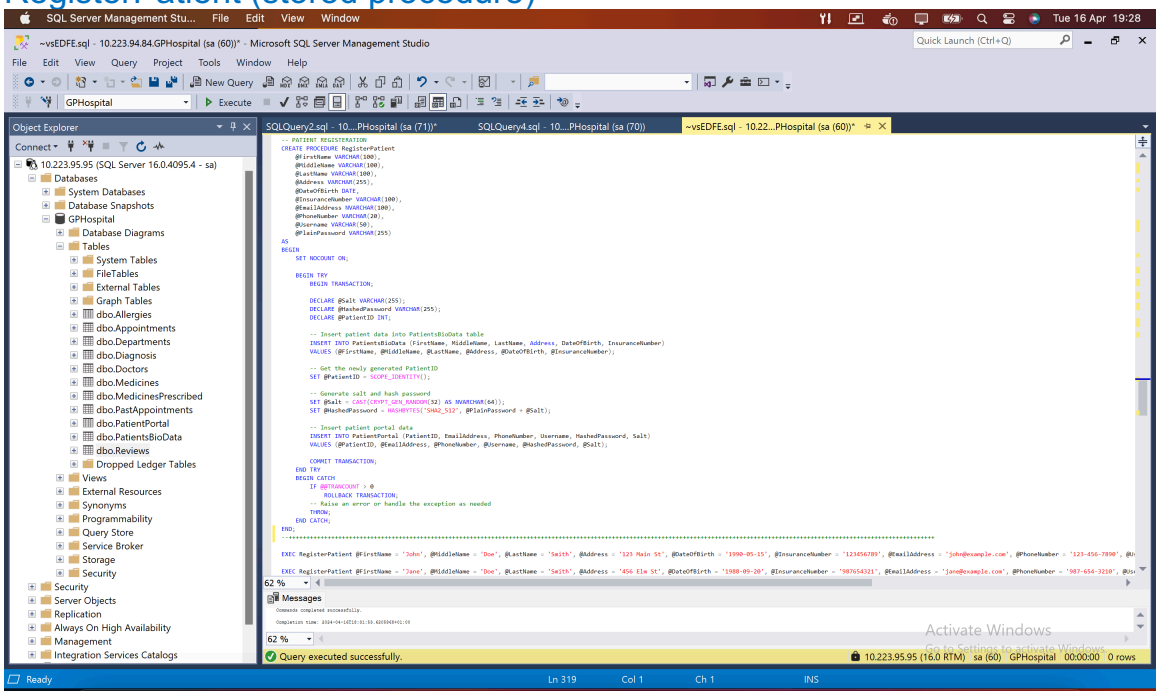


Figure 17

The RegisterPatient stored procedure is a database object that enables patient registration by inserting information into two tables: PatientBioData and PatientPortal. It accepts seven parameters, including biodata, contact details, and password. The procedure inserts data into the PatientsBioData table, retrieves the inserted PatientID, and uses SHA-512 encryption to enhance data privacy and database security.

Finally, the transaction is committed into the table. If there are any errors during insertion of the new records, the transaction will be rolled back and any records inserted during this event will be deleted. This feature is implemented to maintain data integrity in the database.

Figure 18 also show that two patients were registered using the stored procedure. We will view both tables to see that the records were properly inserted.

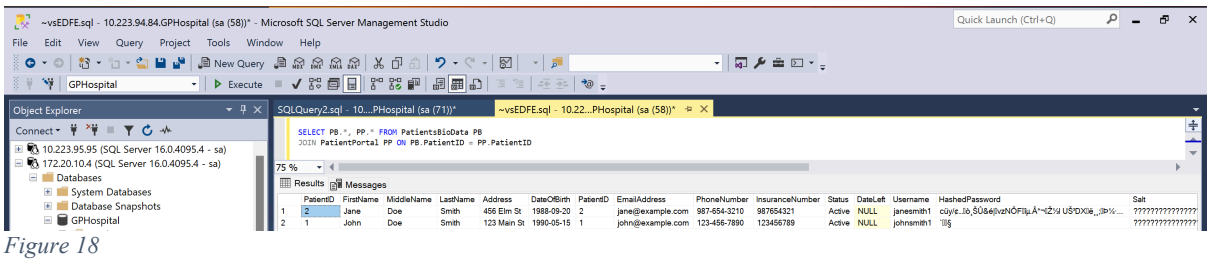


Figure 18

The Image shows the records which were inserted using the **RegisterPatient** stored procedure.

BookAppointment (stored procedure)

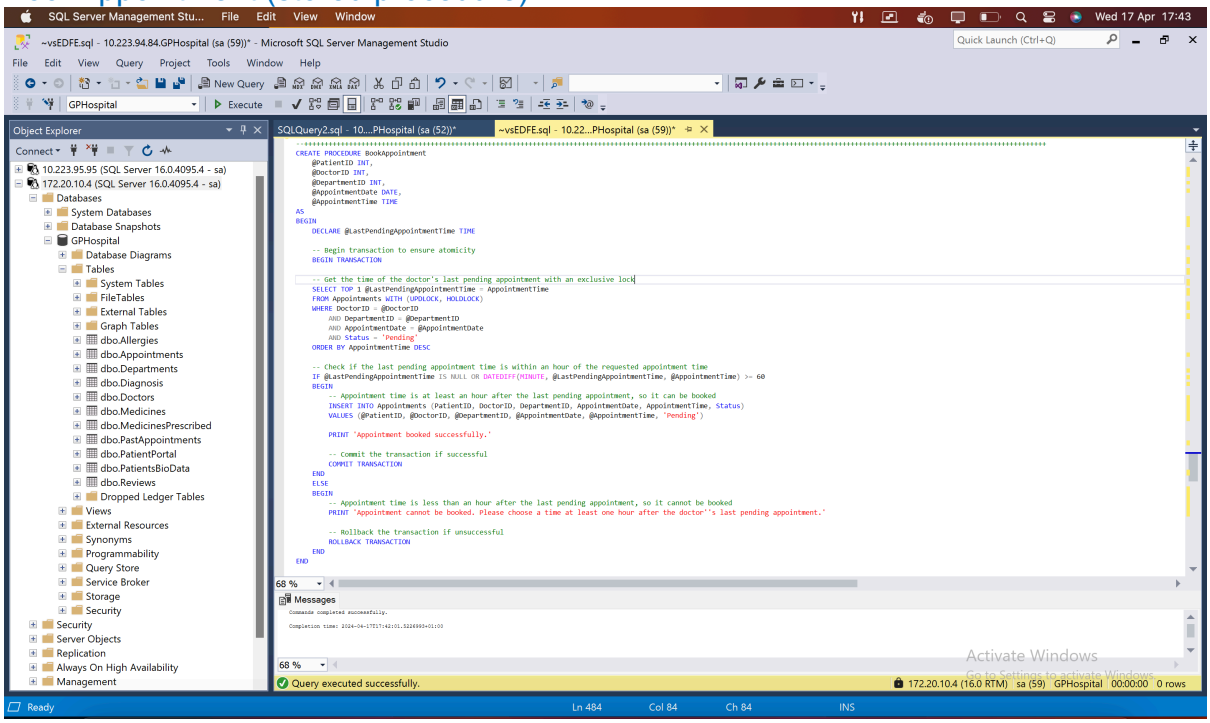


Figure 19

The `BookAppointment` stored procedure manages appointment booking in a GPHospital database. It initiates a transaction to ensure atomicity, selects the latest pending appointment time for a doctor, department, and date, and checks availability. If available, inserts a new record or rolls back. The procedure enforces a minimum one-hour gap between consecutive appointments for a doctor within a department on a specific day, maintaining the appointment scheduling system's integrity.

In figure 20, some Departments and Doctors were added into the database before using the BookAppointment stored procedure to book two appointments.

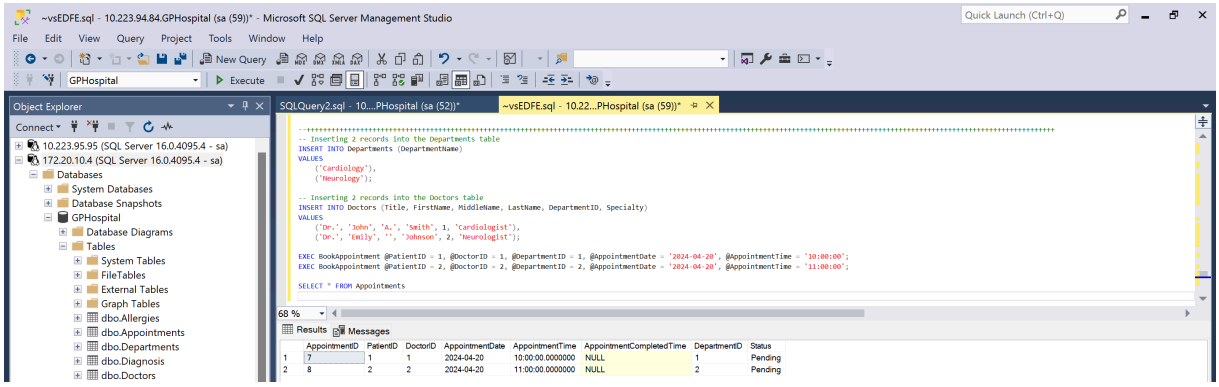


Figure 20

CancelAppointment (stored procedure)

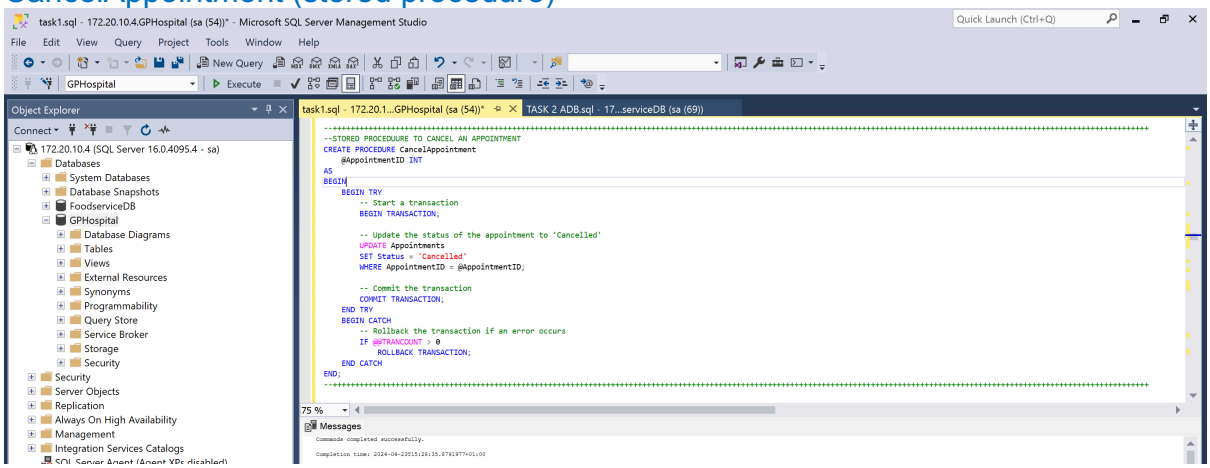


Figure 21

The Cancel appointment stored procedure will cancel an appointment which has already been booked by updating the Status attribute of that record in the Appointment table from `Pending` to `Cancelled`. It initiates a transaction and rolls back if there are any errors to maintain database integrity

RebookAppointment (stored procedure)

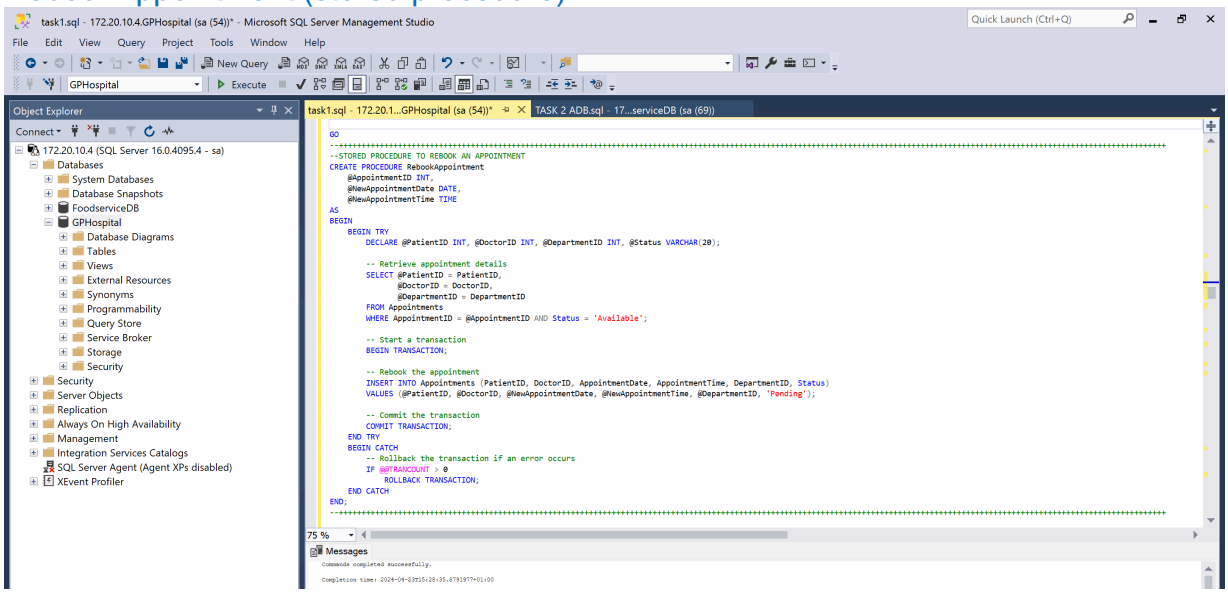


Figure 22

The RebookAppointment stored procedure rebooks an appointment which has been cancelled initially. It takes three parameters, the AppointmentID, New appointment Date and new appointment Time. The stored procedure uses the appointment ID to look for the record in the appointment table and also checks if the Status attribute is `Available` (Because of a trigger in Part 2 question 6). When it finds this record, it uses the values of this record to book a new appointment with the new date and time.

Figure 23 shows the result of the Appointment table after some appointments has been cancelled and rebooked.

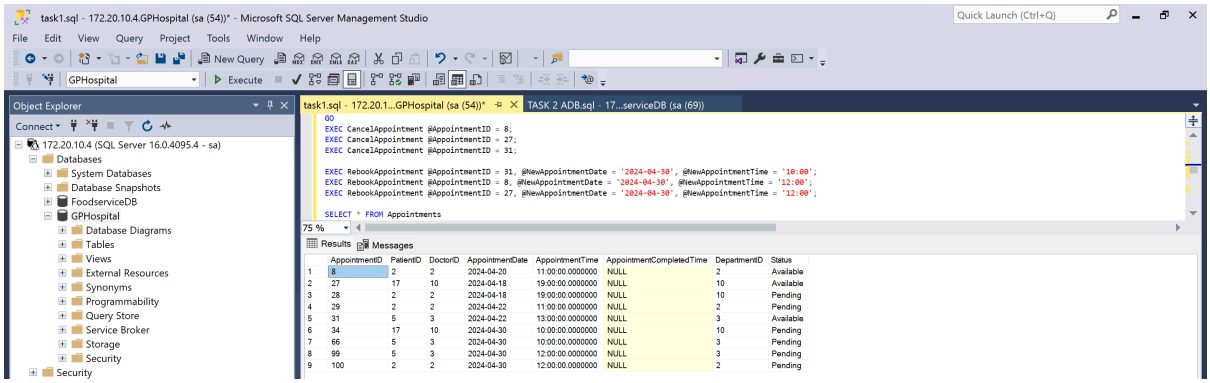


Figure 23

PendingAppointmentsView (View)

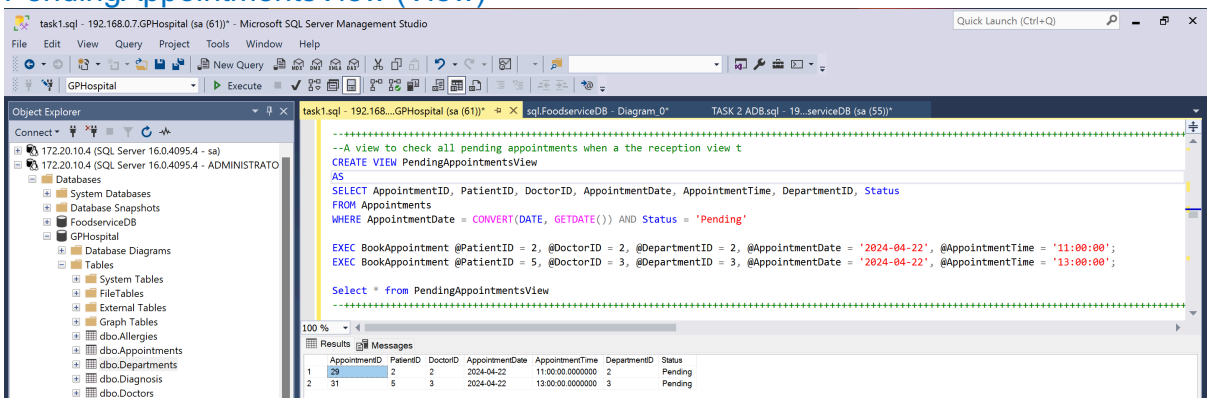


Figure 24

This is a view to check all the pending appointments today. The view uses a select statement on the Appointments table with a WHERE clause to filter appointments with status as pending and date equal to system date.

UpdateMedicalRecord (stored procedure)

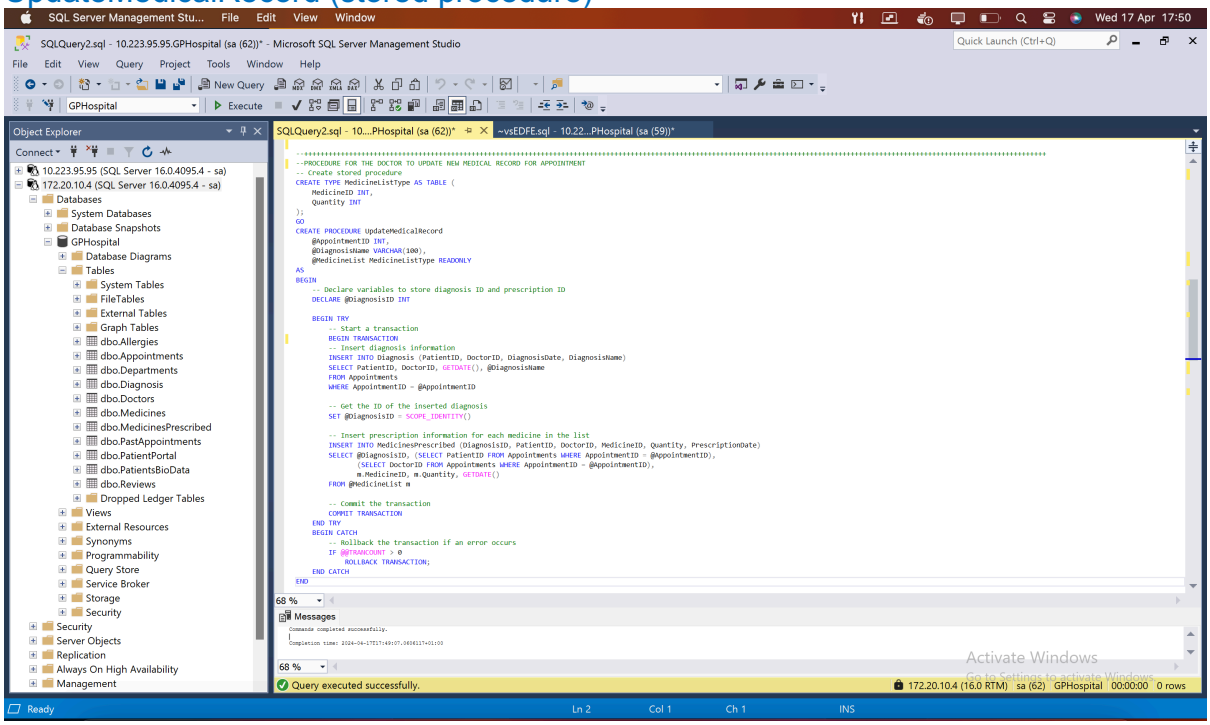


Figure 25

The UpdateMedicalRecord stored procedure updates a patient's medical record by modifying the Diagnosis and MedicinesPrescribed tables. It takes three inputs: AppointmentID, Diagnosis, and Medicine. The procedure initiates a transaction, updates the diagnosis table with provided data, and commits the transaction. If errors occur, the transaction rolls back to delete inserted data.

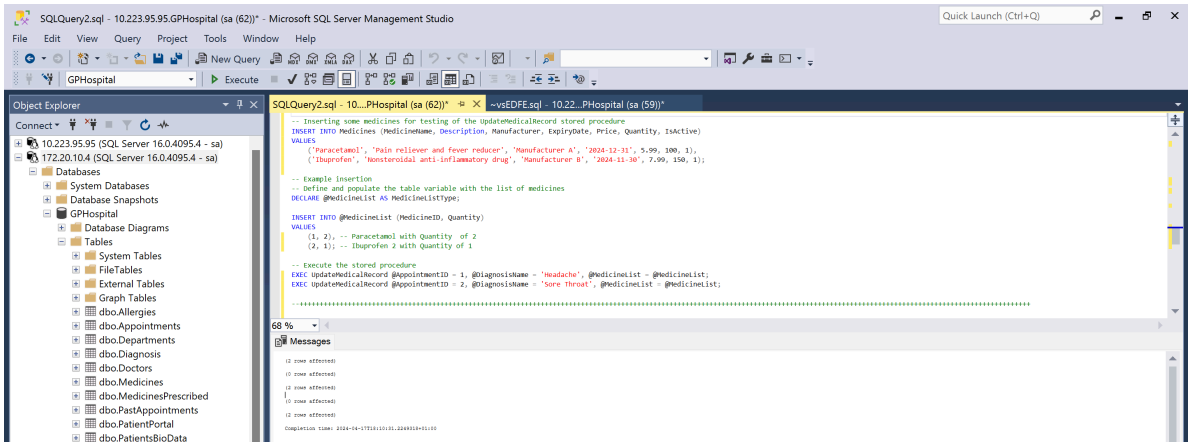


Figure 26

Figure 26 shows some medical records being updated. Some medicines were inserted into the Medicines table in order to properly test our Stored procedure because some MedicineID are needed for our new records for MedicinesPrescribed.

CompleteAppointmentAndLeaveReview (user defined function)

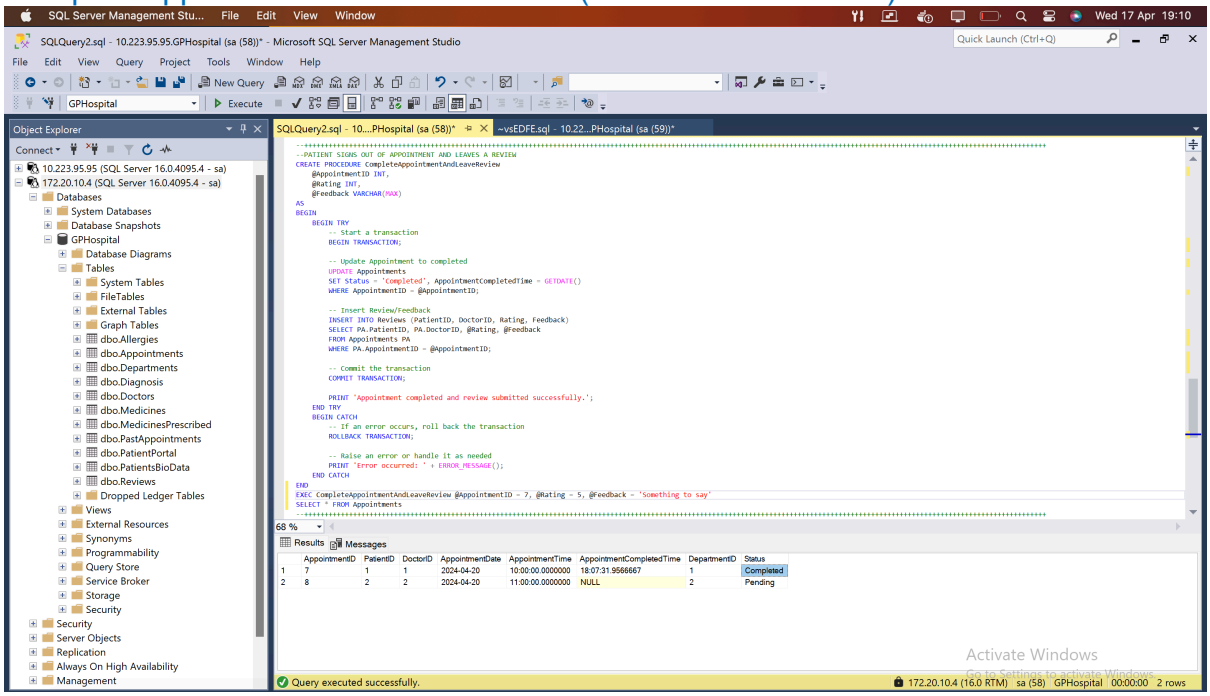


Figure 27

This stored procedure updates the Status attribute of a completed appointment in the completed table to 'Completed' and enables the patient to leave a review by updating the Reviews table. It accepts three inputs: AppointmentID, Rating, and Feedback to update the Appointment and Review tables. During execution, it initiates a

transaction to ensure atomicity and rolls back the entire process if any errors are encountered, thereby maintaining data integrity.

GetPatientMedicalRecord (user defined function)

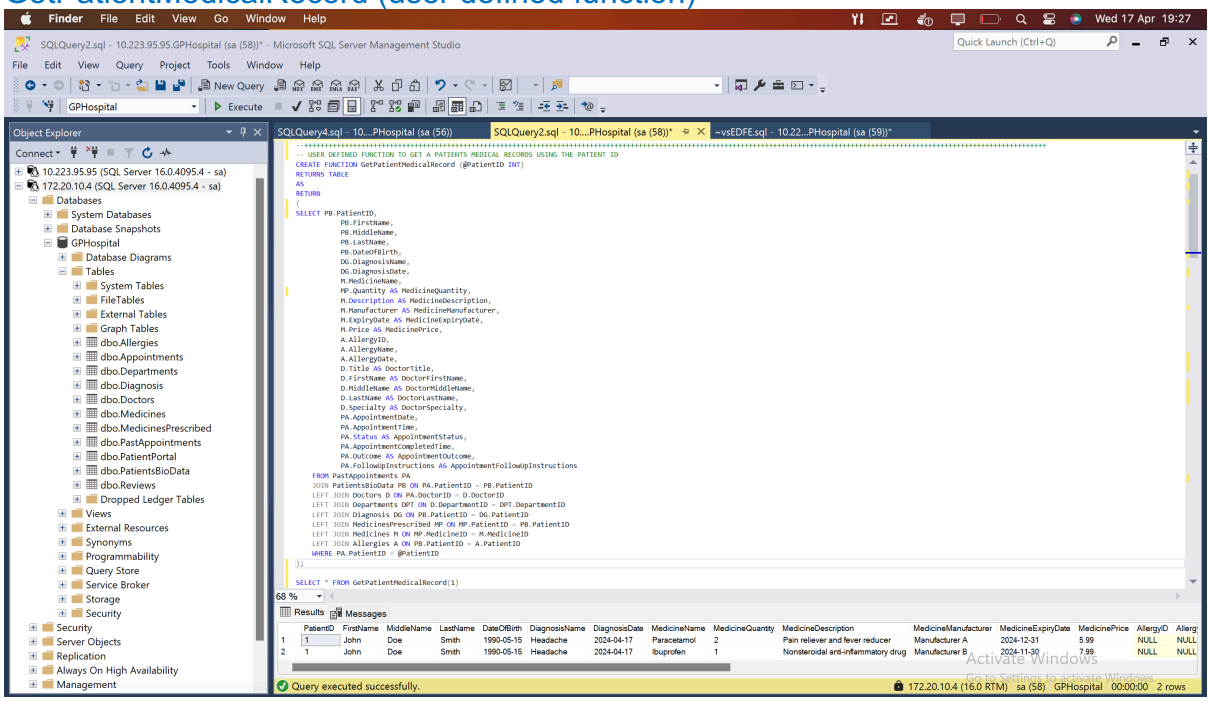


Figure 28

The GetPatientMedicalRecord function retrieves a patient's medical records during an appointment using a select query and joining multiple tables. It accepts the PatientID and returns a table with past medical records, including diagnoses, prescriptions, and allergies. The function displays the patient's medical history.

DeregisterPatient (stored procedure)

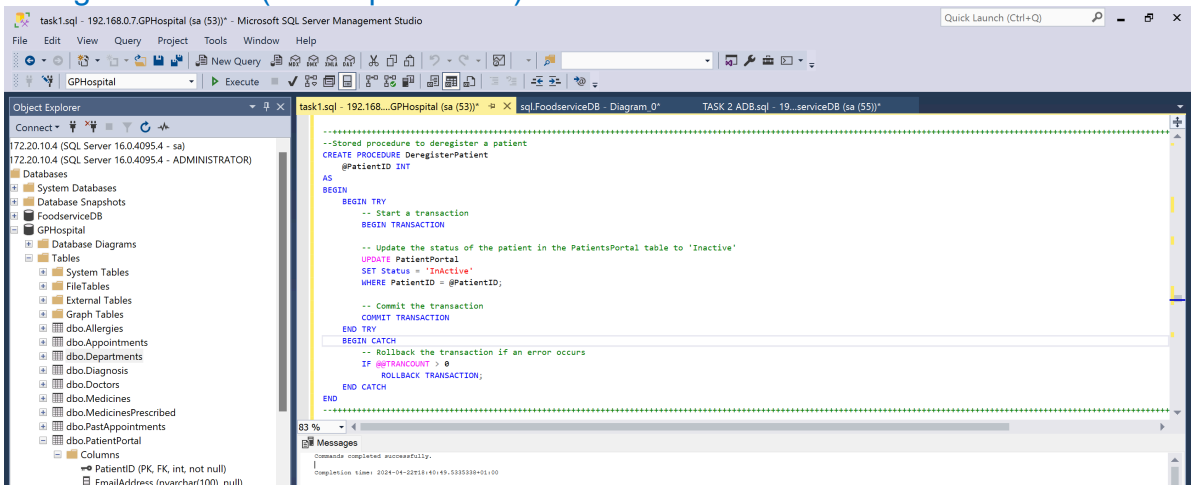


Figure 29

The `DeregisterPatient` stored procedure deregisters a patient by updating the Status attribute of the specified patient to `Inactive`. It takes the parameter

`PatientID` to identify the patient to deregister. This procedure also initiates transaction and rolls back in case of errors to ensure atomicity.

UpdateDateLeftOnInactive (trigger)

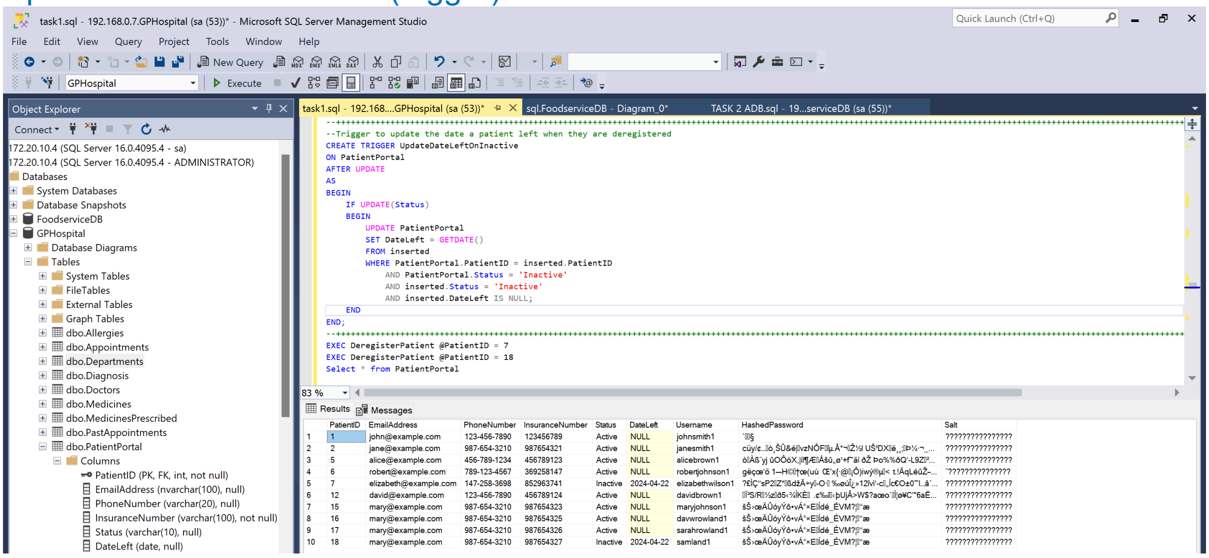


Figure 30

The `UpdateDateLeftOnInactive` trigger is used to record the date a patient left the hospital system, checking if the Status and DateLeft attributes are Inactive or NULL. This trigger is used after two patients were deregistered.

1.6 Inserting data

The following images shows an insertion of multiple records into our database in order to test our database effectively

```
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 9 records into the  Departments table
INSERT INTO Departments (DepartmentName)
VALUES
    ('Orthopedics'),
    ('Pediatrics'),
    ('Gastroenterology'),
    ('Oncology'),
    ('Gynecology'),
    ('Dermatology'),
    ('Urology'),
    ('ENT (Ear, Nose, Throat)'),
    ('Psychiatry');

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 15 records into the Doctors table
INSERT INTO Doctors (Title, FirstName, MiddleName, LastName, DepartmentID, Specialty)
VALUES
    ('Dr.', 'Michael', 'B.', 'Williams', 3, 'Orthopedist'),
    ('Dr.', 'Jessica', '', 'Davis', 4, 'Pediatrician'),
    ('Dr.', 'Robert', 'C.', 'Wilson', 5, 'Gastroenterologist'),
    ('Dr.', 'Sarah', '', 'Martinez', 5, 'Gastroenterologist'),
    ('Dr.', 'David', 'E.', 'Taylor', 5, 'Gastroenterologist'),
    ('Dr.', 'Jennifer', 'F.', 'Anderson', 5, 'Gastroenterologist'),
    ('Dr.', 'Daniel', '', 'Brown', 6, 'Oncologist'),
    ('Dr.', 'Lisa', '', 'Garcia', 7, 'Gynecologist'),
    ('Dr.', 'Kevin', 'G.', 'Lee', 8, 'Dermatologist'),
    ('Dr.', 'Michelle', '', 'Clark', 9, 'Urologist'),
    ('Dr.', 'Brian', 'H.', 'Moore', 10, 'ENT Specialist'),
    ('Dr.', 'Amanda', '', 'Perez', 11, 'Psychiatrist'),
    ('Dr.', 'James', 'I.', 'King', 11, 'Psychiatrist');
```

```sql
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Insert 20 medicine entries
INSERT INTO Medicines (MedicineName, Description, Manufacturer, ExpiryDate, Price, Quantity, IsActive)
VALUES
    ('Amoxicillin', 'Antibiotic', 'Manufacturer C', '2025-03-31', 9.99, 80, 1),
    ('Aspirin', 'Pain reliever and blood thinner', 'Manufacturer D', '2024-10-31', 4.99, 120, 1),
    ('Lisinopril', 'ACE inhibitor for high blood pressure', 'Manufacturer E', '2025-05-31', 11.99, 90, 1),
    ('Atorvastatin', 'Statins for lowering cholesterol', 'Manufacturer F', '2024-09-30', 15.99, 70, 1),
    ('Metformin', 'Antidiabetic medication', 'Manufacturer G', '2025-02-28', 8.99, 110, 1),
    ('Omeprazole', 'Proton-pump inhibitor for acid reflux', 'Manufacturer H', '2024-08-31', 12.99, 100, 1),
    ('Losartan', 'ARB for high blood pressure', 'Manufacturer I', '2024-07-31', 10.99, 130, 1),
    ('Simvastatin', 'Statins for lowering cholesterol', 'Manufacturer J', '2024-06-30', 13.99, 100, 1),
    ('Ciprofloxacin', 'Antibiotic', 'Manufacturer K', '2025-01-31', 14.99, 90, 1),
    ('Hydrochlorothiazide', 'Diuretic for high blood pressure', 'Manufacturer L', '2024-05-31', 9.99, 120, 1),
    ('Metoprolol', 'Beta blocker for high blood pressure', 'Manufacturer M', '2024-04-30', 10.99, 100, 1),
    ('Amlodipine', 'Calcium channel blocker for high blood pressure', 'Manufacturer N', '2025-06-30', 12.99, 80, 1),
    ('Metronidazole', 'Antibiotic and antiprotozoal medication', 'Manufacturer O', '2024-03-31', 8.99, 110, 1),
    ('Albuterol', 'Bronchodilator for asthma', 'Manufacturer P', '2024-02-29', 6.99, 150, 1),
    ('Fluoxetine', 'SSRI antidepressant', 'Manufacturer Q', '2025-07-31', 14.99, 70, 1),
    ('Warfarin', 'Anticoagulant', 'Manufacturer R', '2024-01-31', 7.99, 100, 1),
    ('Doxycycline', 'Antibiotic', 'Manufacturer S', '2023-12-31', 11.99, 90, 1),
    ('Prednisone', 'Corticosteroid', 'Manufacturer T', '2023-11-30', 9.99, 120, 1);


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 8 records into the  PatientBioData and PatientPortal tables
EXEC RegisterPatient @FirstName = 'Alice', @MiddleName = 'Maria', @LastName = 'Brown', @Address = '789 Oak St', @DateOfBirth = '1975-03-28', @InsuranceNumber = '456789123', @EmailAddress = 'al
EXEC RegisterPatient @FirstName = 'Robert', @MiddleName = 'William', @LastName = 'Johnson', @Address = '101 Pine St', @DateOfBirth = '1970-07-12', @InsuranceNumber = '369258147', @EmailAddress
EXEC RegisterPatient @FirstName = 'Elizabeth', @MiddleName = 'Grace', @LastName = 'Wilson', @Address = '246 Maple St', @DateOfBirth = '1965-11-03', @InsuranceNumber = '852963741', @EmailAddres
EXEC RegisterPatient @FirstName = 'David', @MiddleName = 'Robert', @LastName = 'Brown', @Address = '246 Maple St', @DateOfBirth = '1985-05-15', @InsuranceNumber = '456789124', @EmailAddress =
EXEC RegisterPatient @FirstName = 'Mary', @MiddleName = 'Ann', @LastName = 'Johnson', @Address = '789 Oak St', @DateOfBirth = '1990-09-20', @InsuranceNumber = '987654323', @EmailAddress = 'mar
EXEC RegisterPatient @FirstName = 'Dave', @MiddleName = 'Rowland', @LastName = 'Johnson', @Address = '789 Oak St', @DateOfBirth = '2015-09-20', @InsuranceNumber = '987654325', @EmailAddress =
EXEC RegisterPatient @FirstName = 'Sarah', @MiddleName = 'banks', @LastName = 'Johnson', @Address = '789 Oak St', @DateOfBirth = '1950-09-20', @InsuranceNumber = '987654326', @EmailAddress = '
EXEC RegisterPatient @FirstName = 'Sam', @MiddleName = 'John', @LastName = 'Johnson', @Address = '789 Oak St', @DateOfBirth = '1955-09-20', @InsuranceNumber = '987654327', @EmailAddress = 'mar

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 9 records into the  Appointment table
EXEC BookAppointment @PatientID = 3, @DoctorID = 3, @DepartmentID = 3, @AppointmentDate = '2024-10-10', @AppointmentTime = '12:00:00';
EXEC BookAppointment @PatientID = 4, @DoctorID = 4, @DepartmentID = 4, @AppointmentDate = '2024-10-10', @AppointmentTime = '13:00:00';
EXEC BookAppointment @PatientID = 5, @DoctorID = 5, @DepartmentID = 5, @AppointmentDate = '2024-08-10', @AppointmentTime = '14:00:00';
EXEC BookAppointment @PatientID = 6, @DoctorID = 6, @DepartmentID = 6, @AppointmentDate = '2024-08-10', @AppointmentTime = '15:00:00';
EXEC BookAppointment @PatientID = 7, @DoctorID = 7, @DepartmentID = 7, @AppointmentDate = '2024-07-10', @AppointmentTime = '16:00:00';
EXEC BookAppointment @PatientID = 12, @DoctorID = 8, @DepartmentID = 8, @AppointmentDate = '2024-08-10', @AppointmentTime = '17:00:00';
EXEC BookAppointment @PatientID = 15, @DoctorID = 9, @DepartmentID = 9, @AppointmentDate = '2024-09-10', @AppointmentTime = '18:00:00';
EXEC BookAppointment @PatientID = 17, @DoctorID = 10, @DepartmentID = 10, @AppointmentDate = '2024-09-10', @AppointmentTime = '19:00:00';
DECLARE @today DATE;
SET @today = GETDATE();
EXEC BookAppointment @PatientID = 2, @DoctorID = 2, @DepartmentID = 10, @AppointmentDate =  @today, @AppointmentTime = '19:00:00';


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 7 records into the  PatientBioData and PatientPortal tables
DECLARE @MedicineList AS MedicineListType;
INSERT INTO @MedicineList (MedicineID, Quantity)
VALUES
    (1, 2), (3, 2) ;
EXEC UpdateMedicalRecord @AppointmentID = 8, @DiagnosisName = 'Viral Pharyngitis', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 21, @DiagnosisName = 'Breast Cancer', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 22, @DiagnosisName = 'Strep Throat', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 23, @DiagnosisName = 'Skin Cancer', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 24, @DiagnosisName = 'Acute Bronchitis', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 25, @DiagnosisName = 'Gastroesophageal Reflux Disease (GERD)', @MedicineList = @MedicineList;
EXEC UpdateMedicalRecord @AppointmentID = 26, @DiagnosisName = 'Brain Cancer', @MedicineList = @MedicineList;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 8 records into the  PatientBioData and PatientPortal tables
EXEC CompleteAppointmentAndLeaveReview @AppointmentID = 21, @Rating = 5, @Feedback = 'Something to say'
EXEC CompleteAppointmentAndLeaveReview @AppointmentID = 22, @Rating = 5, @Feedback = 'Something to say'
EXEC CompleteAppointmentAndLeaveReview @AppointmentID = 26, @Rating = 5, @Feedback = 'Something to say'


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-- Inserting 8 records into the  PatientBioData and PatientPortal tables
INSERT INTO Allergies (PatientID, AllergyName, AllergyDate) VALUES
(1, 'Peanuts', '2024-04-17'),
(2, 'Shellfish', '2024-04-16'),
(2, 'Chocolate', '2024-04-16'),
(3, 'Pollen', '2024-04-15'),
(4, 'Dust', '2024-04-14'),
(5, 'Milk', '2024-04-13'),
(6, 'Eggs', '2024-04-12'),
(2, 'Penicillin', '2024-04-18'),
(7, 'Penicillin', '2024-04-11');
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 31

Part 2

Q.2: Add the constraint to check that the appointment date is not in the past.

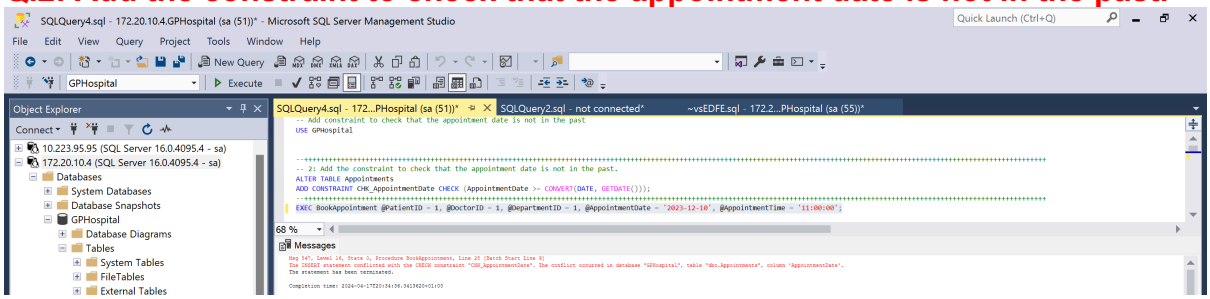


Figure 32

To add a constraint to an already existing table, you can use an ALTER TABLE statement to add a constraint. The CHK_Appointment constraint was added as a CHECK constraint which checks that the new insertion is greater or equal to the current system date. In figure 32 above, an error was thrown when attempting to insert an appointment record with the date in the past.

Q.3: List all the patients with older than 40 and have Cancer in diagnosis.

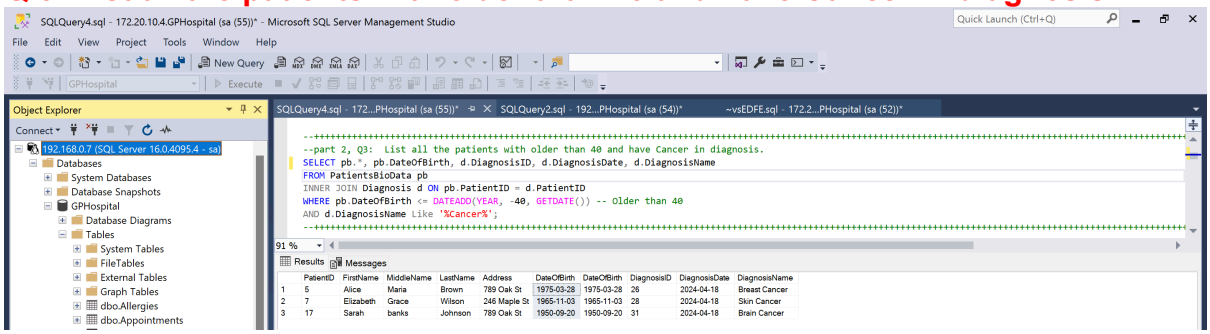


Figure 33

The query extracts attributes from PatientsBioData and Diagnosis tables, based on PatientID. The output is filtered using a WHERE clause, combining two conditions: DateOfBirth less than today's date minus 40 years and DiagnosisName containing 'Cancer' (%Cancer%: starts with character, 'Cancer' in middle, ends with any character).

Q.4A: Search the database of the hospital for matching character strings by name of medicine. Results should be sorted with most recent medicine prescribed date first.

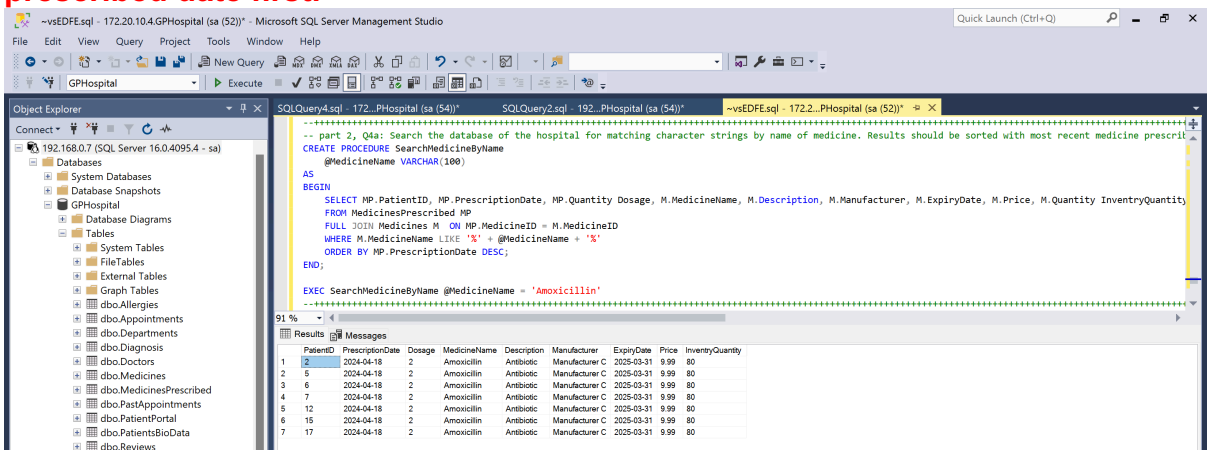


Figure 34

This stored procedure retrieves a table of prescribed medicines from the hospital's database. It utilizes a full join between the Medicine and MedicinePrescribed tables. The procedure accepts a single input representing the medicine name and filters results using a WHERE clause. The final output is sorted in descending order by prescription date, ensuring the most recent prescription appears first.

Q.4B: Return a full list of diagnosis and allergies for a specific patient who has an appointment today.

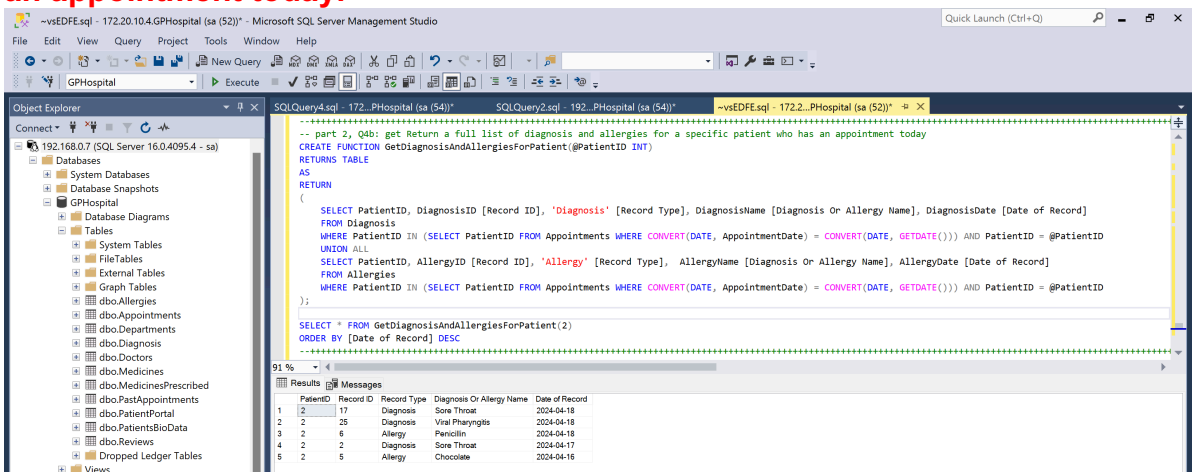


Figure 35

A user-defined function was used to select records from a UNION of the diagnosis table and the allergies table based on a PatientID matching the Appointment table IDs for the current day, resulting in a table ordered by Record Date.

Q.4C: Update the details for an existing doctor.

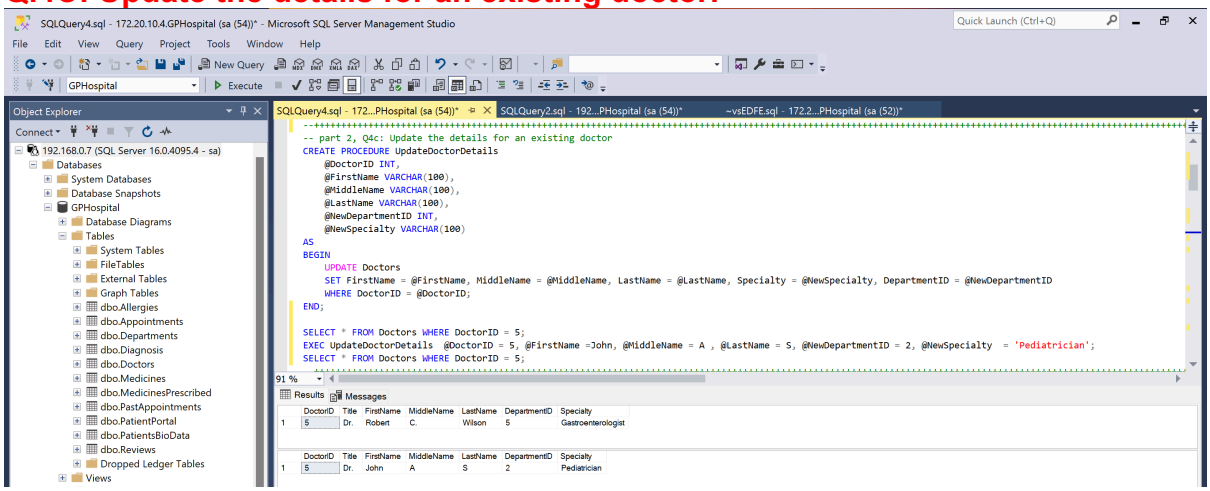


Figure 36

For this question, the stored procedure takes all the Attributes of the Doctors table as input and updates the Doctor table. The screenshot shows the records of a particular doctor before and after the stored procedure was used. The attributes were updated with the new values appropriately.

Q.4D: Delete the appointment who status is already completed.

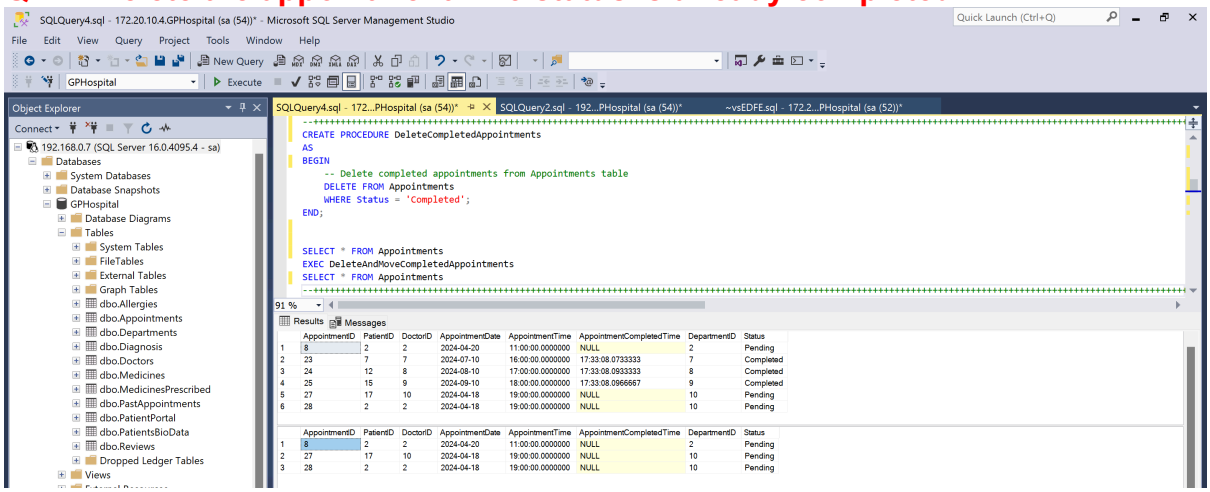


Figure 37

The DeleteCompletedAppointment Stored procedure uses the DML ‘DELETE’ and there WHERE clause to delete a record where the Status attribute is ‘Completed’. The results show the tables before and after the stored procedure was executed.

Q.5: The hospital wants to view the appointment date and time, showing all previous and current appointments for all doctors, and including details of the department (the doctor is associated with), doctor's specialty and any associate review/feedback given for a doctor. You should create a view containing all the required information.

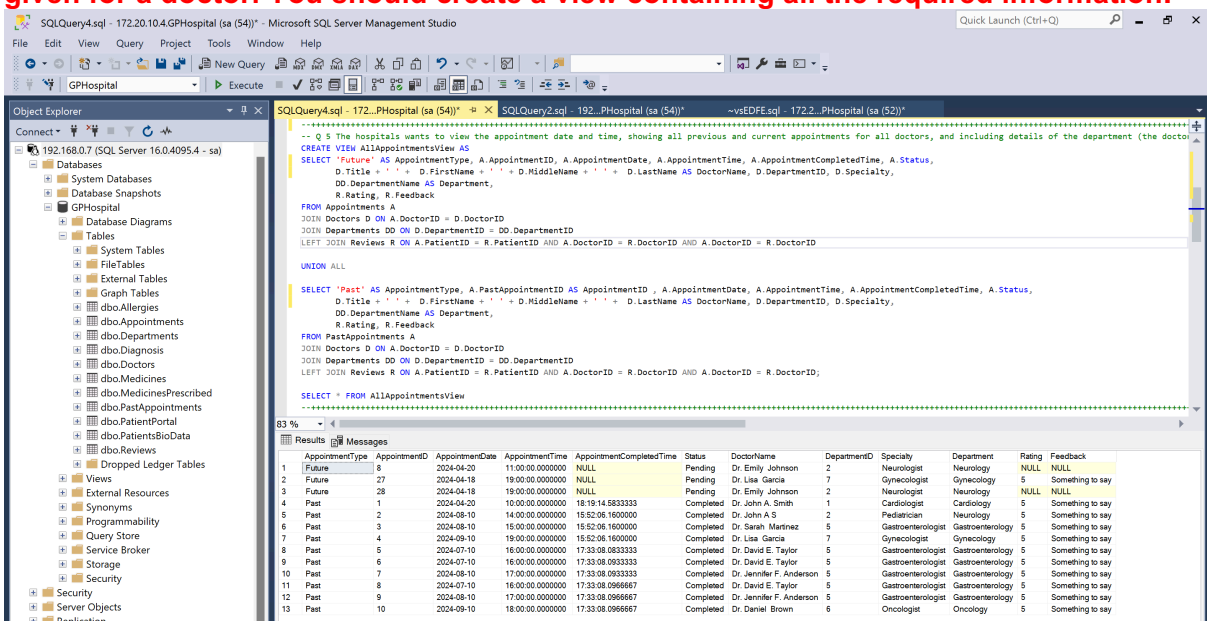


Figure 38

AllAppointmentsView returns all past and future appointments with all associated doctors, departments and ratings. It makes use of the UNION ALL to join results from two different queries: The first query joins the Appointments, Doctors, Departments and Reviews using their different IDs as the joining conditions, the second query joins the PastAppointments, Doctors, Departments and Reviews using their different IDs as the joining conditions.

Q.6: Create a trigger so that the current state of an appointment can be changed to available when it is cancelled.

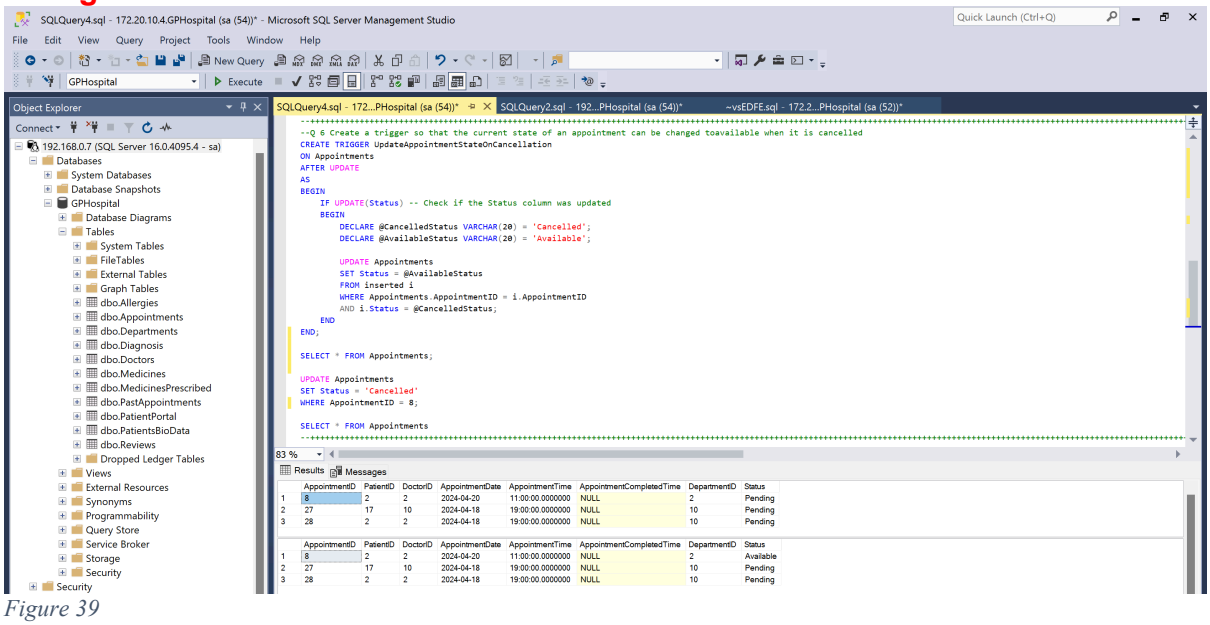


Figure 39

The UpdateAppointmentStateOnCancellation Trigger is set to change an appointment status to Available when it is Cancelled. The trigger is attached to the Appointments table. When an update is made to the Appointments table, the trigger is called. The trigger checks if the Status attribute has been altered. If it has, It updates rows in the appointment table where the Status is equal to ‘Cancelled’.

Q.7: Write a select query which allows the hospital to identify the number of completed appointments with the specialty of doctors as ‘Gastroenterologists’.

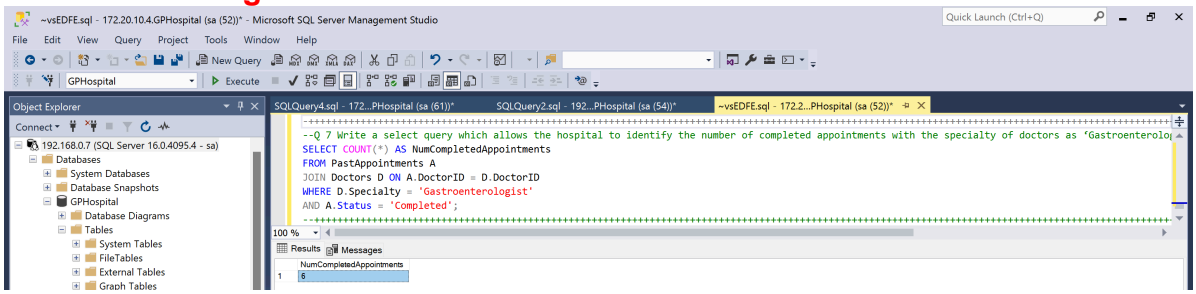


Figure 40

This select query uses a combination of the PastAppointments table and the Doctors table to extract the number records about appointments completed by a Gastroenterologist. It uses a JOIN statement to join the two tables with the condition of the DoctorID being equal, a WHERE clause to filter out the rows where the Doctors specialty is recorded as a ` Gastroenterologist`. The result shows that 6 appointments have been completed by Gastroenterologists.

QESTION 8

Data integrity and concurrency

The GPHospital database was developed with constraints to ensure data integrity. Unique constraints were applied to tables like PatientsBioData, PatientsPortal, Departments, Doctors, Appointments, and Diagnosis to prevent duplicate records and a CHECK constraint on the Appointment table to prevent past appointment dates.

Transactions within stored procedures ensure data consistency and atomicity, while exclusive locking in the Appointment booking stored procedure prevents concurrent transactions from booking conflicting appointments, ensuring system reliability.

Database security

After creation of the database and the tables, I created a user called ADMINISTRATOR with password as ‘Admin1234’ as seen in section **1.3**. This user has full access to all the database tables and will be able to grant other users access to some or all of the tables in the database.

To maintain optimal database security, I advise that this administrator user should only grant access to only trusted parties. The access granted to these new users should not supersede the privileges of the ADMINISTRATOR. Only limited access should be granted to tables or objects in the database which are only required to complete tasks of that new user.

Database backup and recovery

The database should be backed up regularly to minimize data loss in case of emergencies.
Note that we already Granted the administrator permission to back-up our database in section **1.3**, Therefore we can back up the GPHospital database.

The following are the steps you should take for the database backup in Microsoft SQL management studio.

i. Navigate to Tasks → Back Up

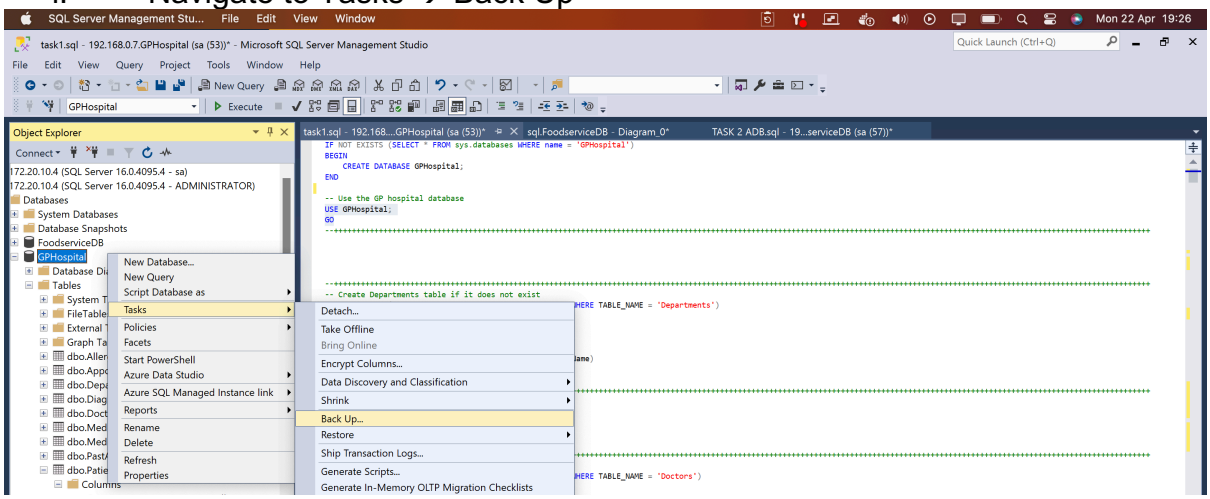


Figure 41

ii. Select Database: GPHospital, Backup type: Full.

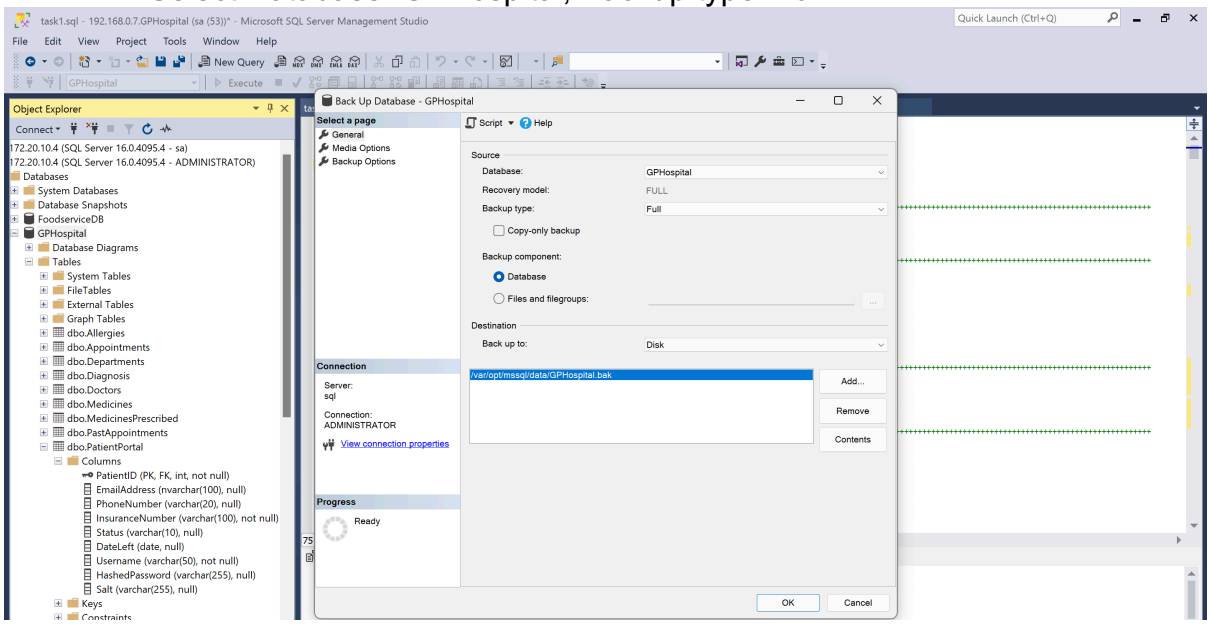


Figure 42

Take note of the directory or change to directory of your choice

iii. Navigate to Media Options and select `Overwrite all existing backup sets`

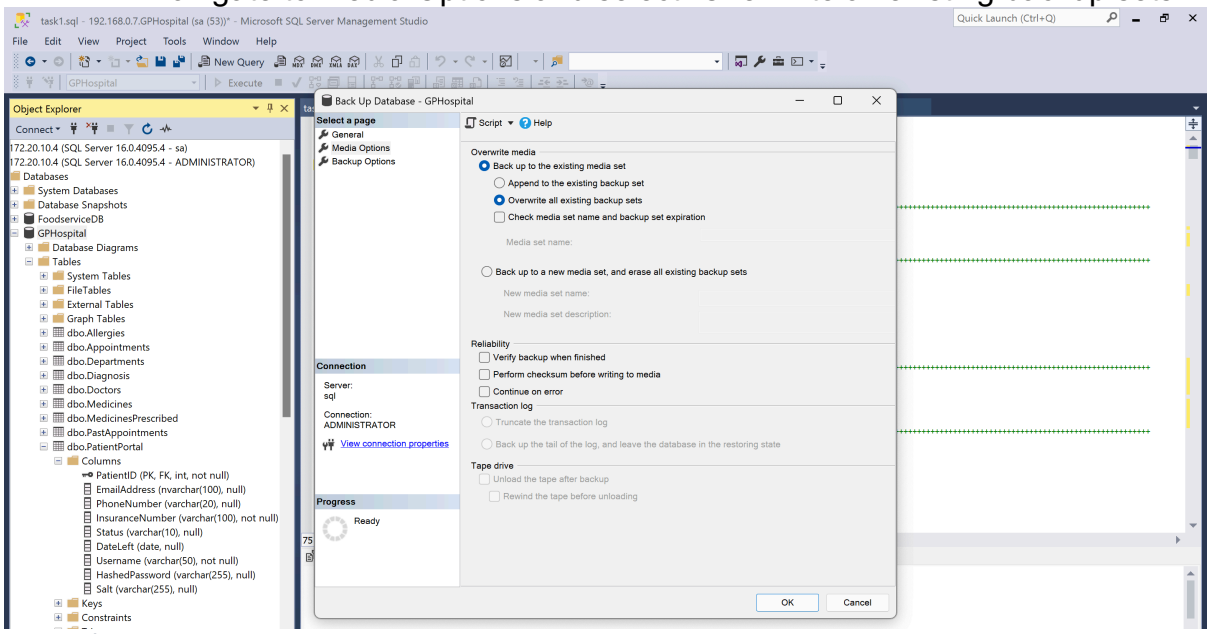


Figure 43

iv. Navigate to Backup Options and change the `Compression` section to `Compress backup` as shown below.

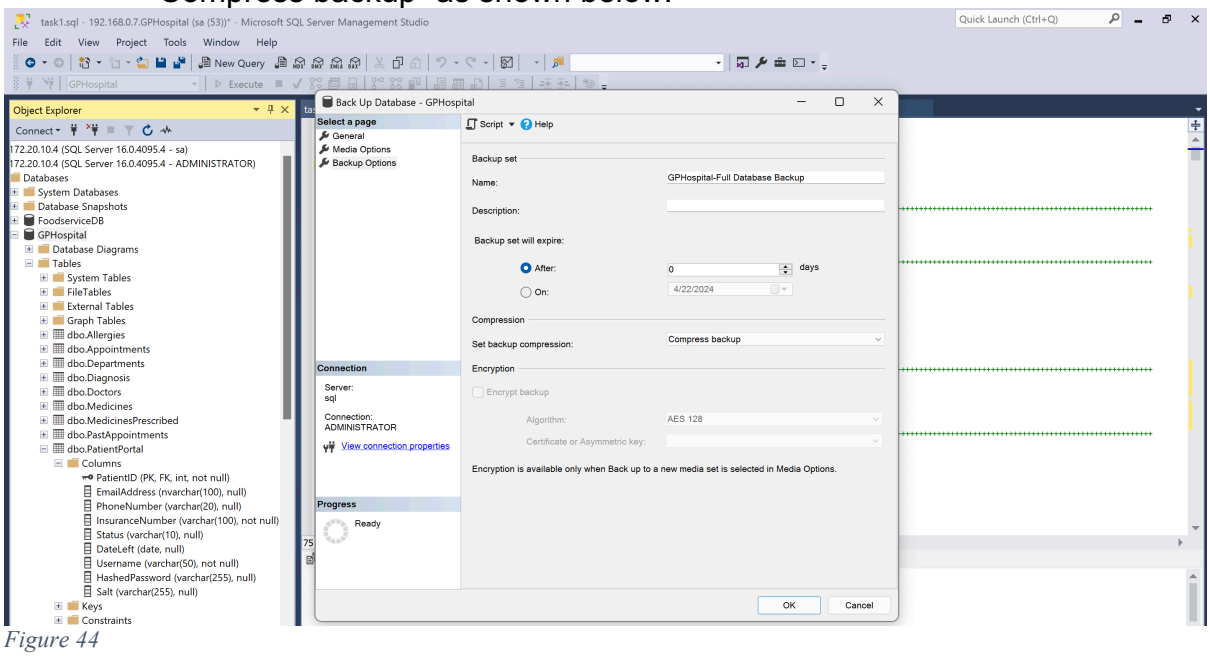


Figure 44

v. Click the `OK` button to back up and complete the process.

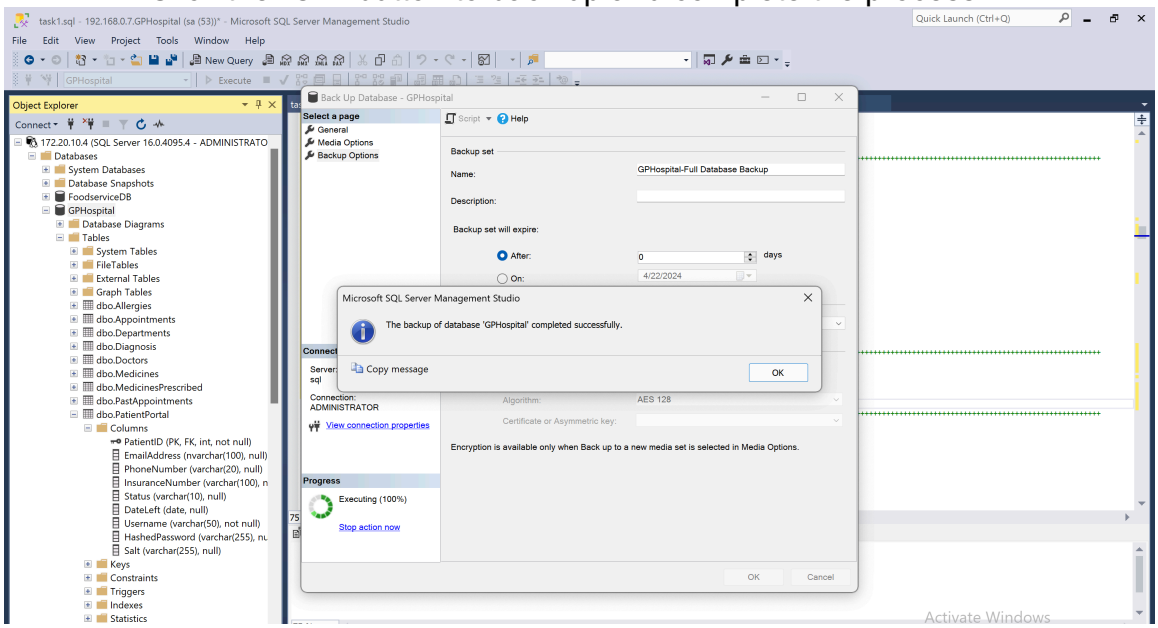


Figure 45

The message in the figure shows that our backup was successful. Now we can use this `.bak` file restore our database with the following steps.

i. Navigate to Tasks → Restore → Database

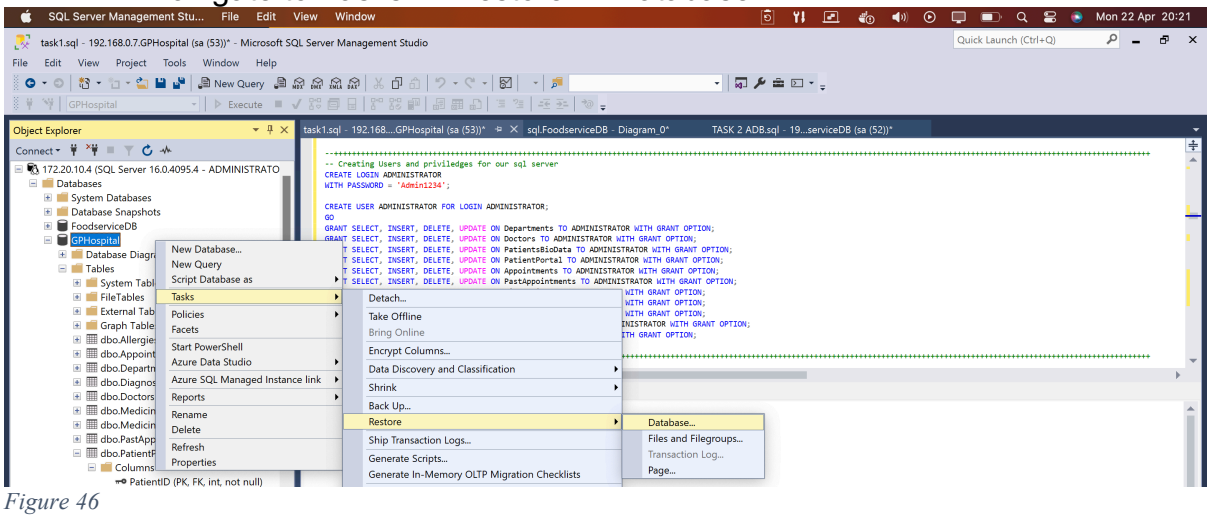


Figure 46

ii. Select Source: Device and Destination: GPHospital, also select the `…` button to show the dialog box in the next figure

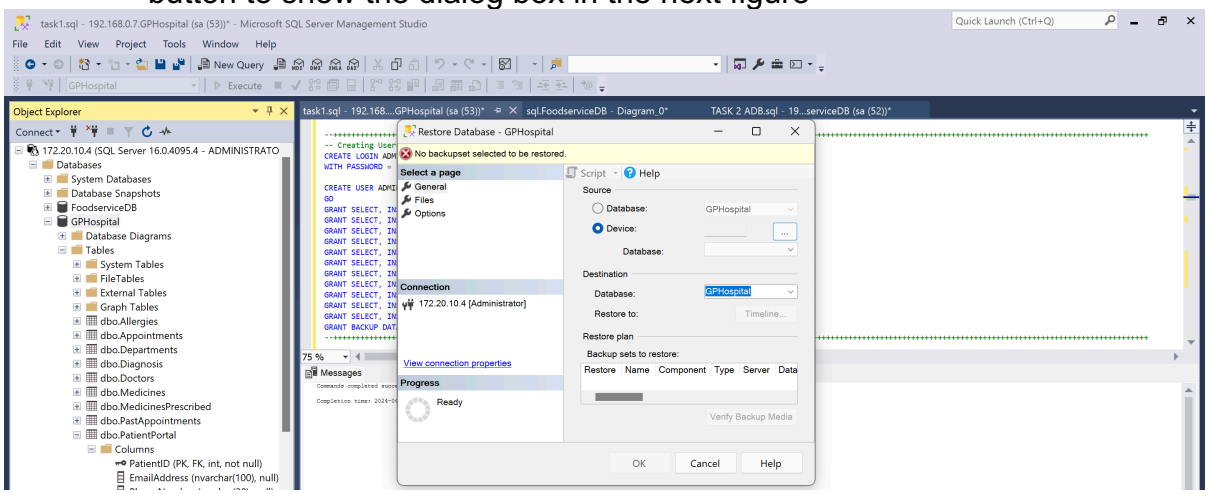


Figure 47

iii. Select Source: Device and Destination: GPHospital, also select the `…` button to show the dialog box in the next figure

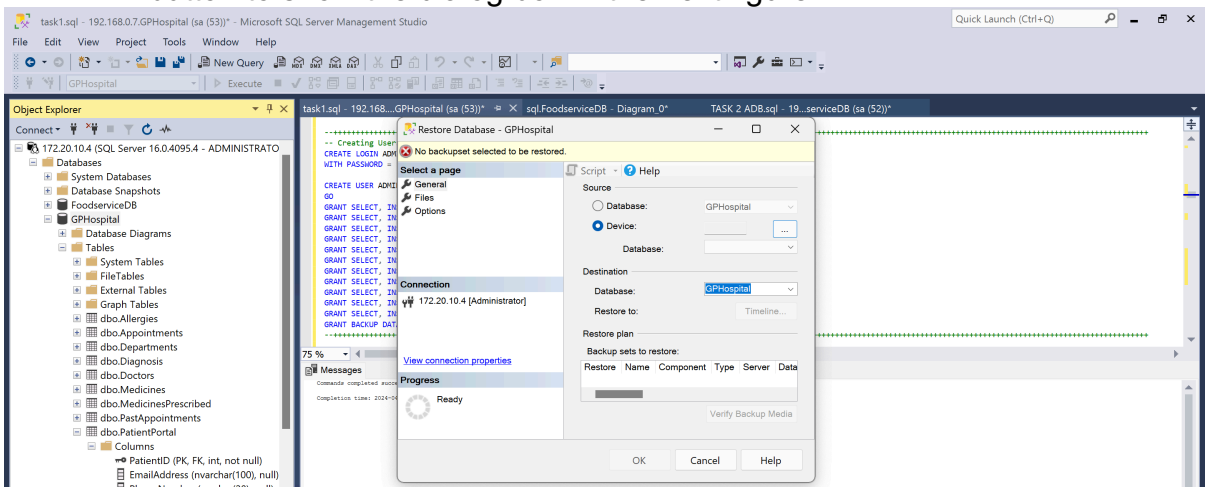


Figure 48

iv. Select Backup media type as File and click the `Add` button to show another dialog box as shown below, select GPHospital.bak and click OK.

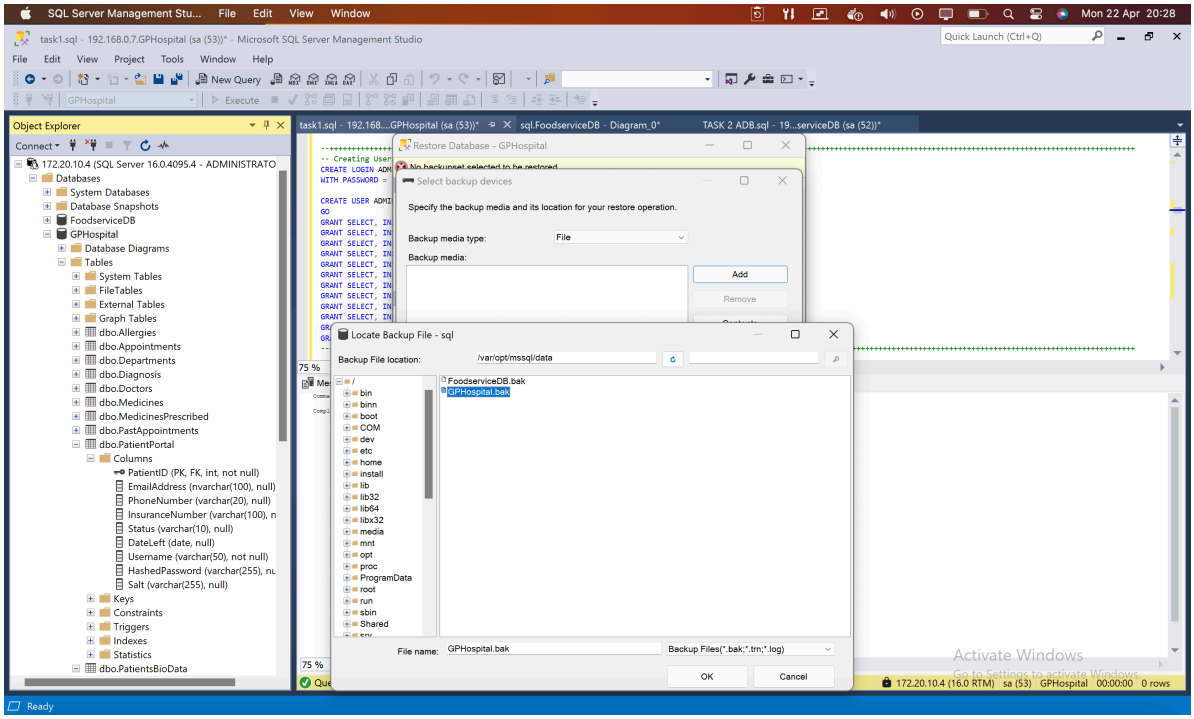


Figure 49

v. Navigate to `Options` and select Recovery state as RESTORE WITH RECOVERY, and check the `Close existing connections to destination database`, then click OK to complete the process.

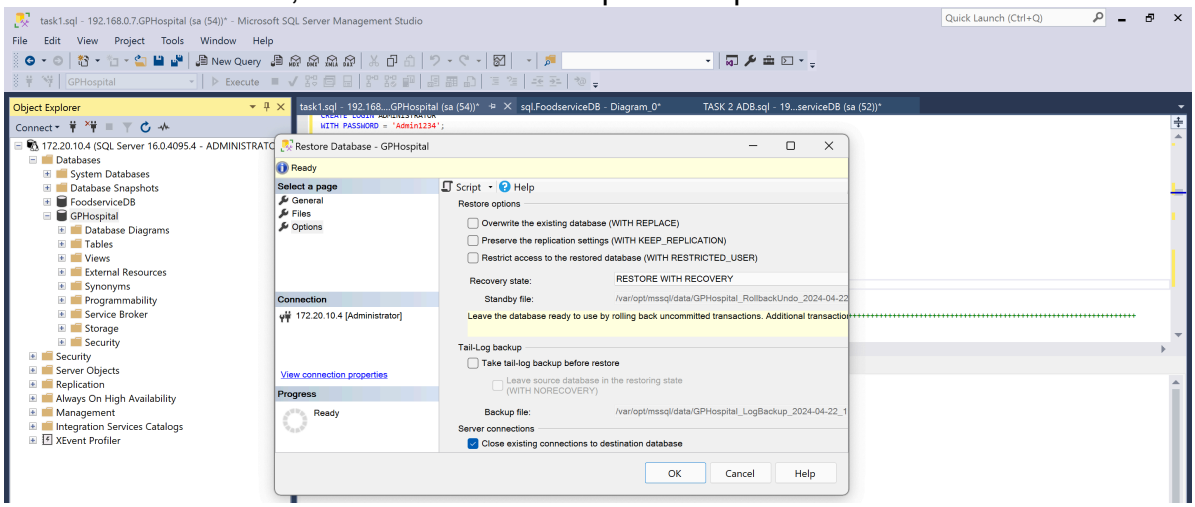


Figure 50

i. The database has been recovered successfully in the figure below.

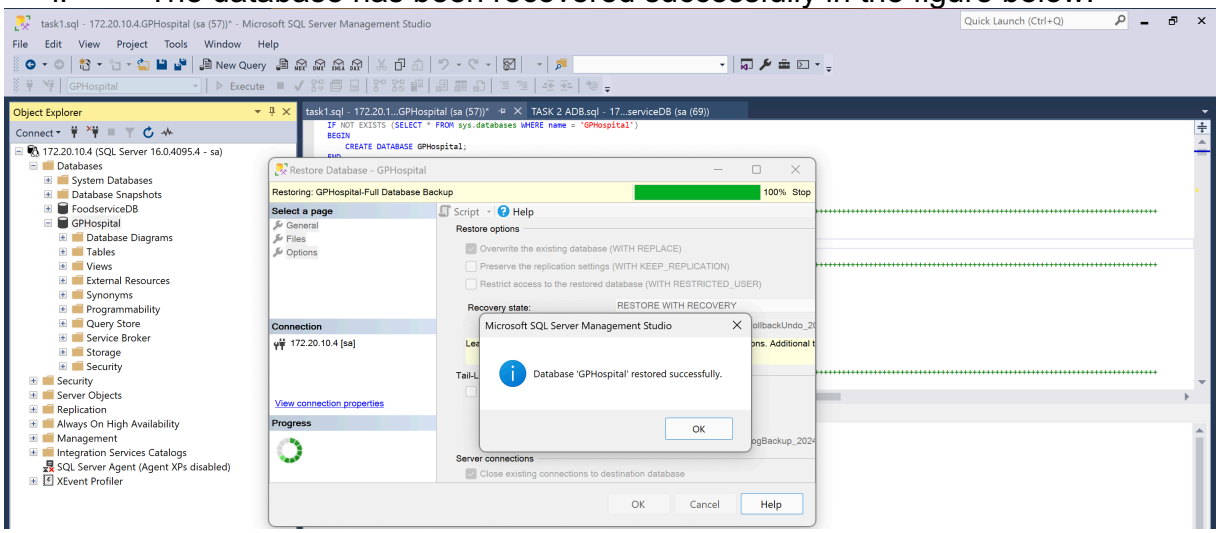


Figure 51

Conclusions

In this task, I was required to design and develop a database for a Hospital which will meet all daily operational needs of the hospital.

I designed and created a database called GPHospital and several tables, normalized in 3NF to fit the requirements of the Hospital.

I added primary and foreign key constraints to the tables to form entity relationships between the tables. I also enhanced the functionality of the database by adding various database objects such as user defined functions, stored procedures, triggers and views to meet some functional requirements of the daily operations of the hospital.

Finally, we inserted some test data into our database and performed some operations and analysis on this database such as registering patients, booking appointments and viewing medical records of patients.

This task evaluated and improved my knowledge and expertise in Advanced database design and development.