CSE 581 Introduction to Database Management Systems (Spring 2021)

Project 2: University Medical Center Database

ABSTRACT

The objective of this project is to design, implement and test a Database Management System in order to maintain the data of University Medical Center. This database will also investigate further about the typical operations in a hospital setting and accordingly consider several base tables such as for medical staff as well as linking tables such as for patient appointments, physician schedules, nurse schedules, surgery room schedules, inpatient room schedules, and equipment reservations.

NAME: Vijayalakshmi Girijala

SUID: 454260110

DATE: 20th May, 2021

INTRODUCTION

This project is divided into three steps, that is Design, Implementation & Testing. Design includes the different Entity Relationships and the relevant tables. We will also discuss possible business logics and transactions to consider, and how to incorporate them into the design. Implementation involves execution of codes that create tables, views, triggers, procedures etc. And finally, the testing phase includes executing various scenarios that test the complexity of the database.

DESIGN CONSIDERATIONS

Table 1: Entity – Attributes

Table No.	Entity	Attributes
1	**Facility**	**FacilityID [PK]**,FacilityName, Location, Hours, RoomCapacity, DepartmentID, CareCapacityLevel, MedicalEquipment
2	**Department**	**DepartmentID [FK]**, DepartmentName
3	**CareCapacity**	**CareCapacityLevel [FK],** CareCapacityName
4	**MedicalStaff**	**StaffID [PK]**, **FacilityID [FK]**, StaffType, Position, Status
5	**StaffPersonalInfo**	**StaffID [FK],** StaffName, StaffAddress, StaffPhone, StaffEmail
6	**Employees**	**EmployeeID [PK]**, **StaffID [FK]**, EmployeeSalary, Benefits, ContractID, EmployeeReviewID, **DepartmentID [FK]**, OfficeRoom, OfficePhone
7	**EmployeePersonalInfo**	**EmployeeID [FK]**, EmployeeName, EmployeeAddress, EmployeePhone, EmployeeEmail
8	**ContractDetails**	**ContractID [FK]**, ContractType, ContractTerm
9	**EmployeeReviewDetails**	**EmployeeReviewID [FK],** EmployeeReview, ReviewerName
10	**Patient**	**PatientID [PK]**, InsuranceID, PrimaryCareDoctor, BillingID
11	**PatientPersonalInfo**	**PatientID [FK]**, PatientName, PatientDOB, PatientAddress, PatientPhone
12	**PatientMedicalRecords**	**PatientRecordNumber [PK]**, **PatientID [FK]**, Weight, Height, Vitals, CheckIn, CheckOut, Symptoms, Diagnose, **TestID [FK]**, ProcedureCode, AttendingPhysician, ReferralDoctor, MedicationID
13	**Lab**	**TestID [PK]**, Results, MandatedReportCounty, MandatedReportState
14	**Tests**	**TestID [FK]**, TestName
15	**Imaging**	**TestID [FK]**, ImagingLocation
16	**Medication**	**MedicationID [FK]**, MedicineNumber
17	**ProcedureInfo**	**ProcedureCode [FK]**, ProcedureName, ProcedureCost

18	**Pharmacy**	**MedicineNumber [FK]**, MedicineName, MedicationCost
19	**Billing**	**BillingID [FK]**, NumberOfVisits, MedicalBillingCode, Payer, PaymentMethod, **TotalCost [FK]**
20	**Cost**	**MedicationCost [FK]**, **ProcedureCost[FK]**, TotalCost
21	**InsuranceCoverage**	**InsuranceID[FK],** InsuranceCompany, InsurancePhone, InsuranceCoverage
22	**Visitors**	**PatientID[FK]**, **VisitorID [PK]**, VisitorName, Relation, entryTime, ExitTime

**PK – Primary Key, *FK – Foreign Key*

ENTITY RELATIONSHIP

This relationship as shown in the figure below is incorporated in the 3NF. There are 20+ tables shown here.

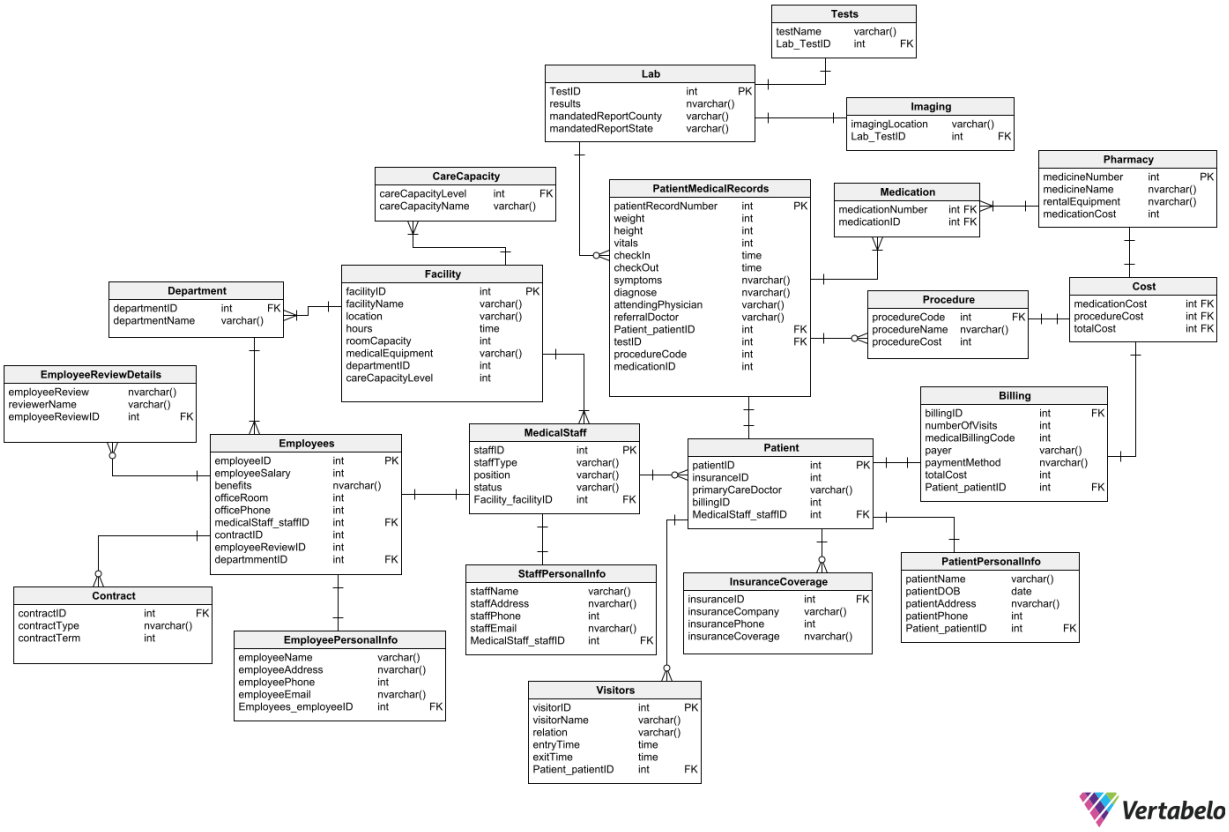


Figure 2: Entity Relationship Diagram

Choices

The choices of the entity and attributes made are as shown in the Entity Relationship Diagram. Considering the choices made, the relationships between tables is also shown. There are more than 20 tables for my design implementation. All tables are normalized in 3NF.

The University Medical Center database consists of 5 main tables and all the other tables are referenced with them.

The first table – **Facility** is one of the essential tables in the UMC Database as it contains details of each facility and the departments under it with care capacity levels. Each facility is a center for different medical areas and each facility consists of specialized departments. FacilityID is the primary Key here.

The second table – **Employees** has details of every employee working at the hospital be it a doctor or a clerical worker. Many tables like EmployeePersonalInfo and ContractDetails are referenced from employeeID which is the primary key of Employees. It is also used to derive other employee information.

The third table – **MedicalStaff** contains the details of the medical staff at the hospital like doctors and nurses. Also references staff information like salary, position, status. It also references patient tables as medical staff act as attending physicians or referral doctors.

The fourth table – **PatientMedicalRecords** has all the data important for patients from their symptoms, vitals to their medication. Tables like PatientInfo, and Lab tests contain patientID as foreign key to find test results and other important information. patientID is the primary key for Patient. Visitors table also references from patientID.

The fifth table – **Billing** contains the accounting part of the database. Consists of data dealing with procedure and medication costs. Also deals with InsuranceCoverage data and payment methods.

To understand **business reliability and logic**, let's look at some scenarios.

All IDs for any table need to be unique as they act as either the primary keys or foreign keys. So using UNIQUE **constraint** is important. Another constraint to be considered is NOT NULL. empty values have a negative effect on the database, hence, NOT NULL is also an important constraint.

We will be using **views** to handle scenarios like the total cost billed to each patient, how many days each patient has been checked into the hospital or to find out the top 5 employees at UMC. Data might also be as full name and would be required to be split into first name and last name for the administration. We will be using views to achieve these.

We might need to check if any patients are at the hospital or are checked in during a given time period or may need to find staff within a desired salary range. For these purposes, we will be using **stored procedures**. These procedures can be used for business analysis. We will also create user-defined **functions** to perform various operations on the database.

To deal with correction during insertion of values into tables or to handle missing or empty values, we will use **triggers**. Any time such an insert query is written it will fire a trigger to handle these errors/issues.

Suppose we want to make sure that values from a table are not deleted accidentally, we will use **transactions** to CATCH the query that deletes the rows and ROLLBACK ensuring it doesn't happen. This maintains the integrity of the tables.

We will also use **scripts** to grant different permissions to insert, update etc to certain users with given roles only. Thus creating different security levels. Scripts will also include queries to password protect the database with login information.

These are some of the logics and scenarios behind our business decisions for the design implementation.

IMPLEMENTATION

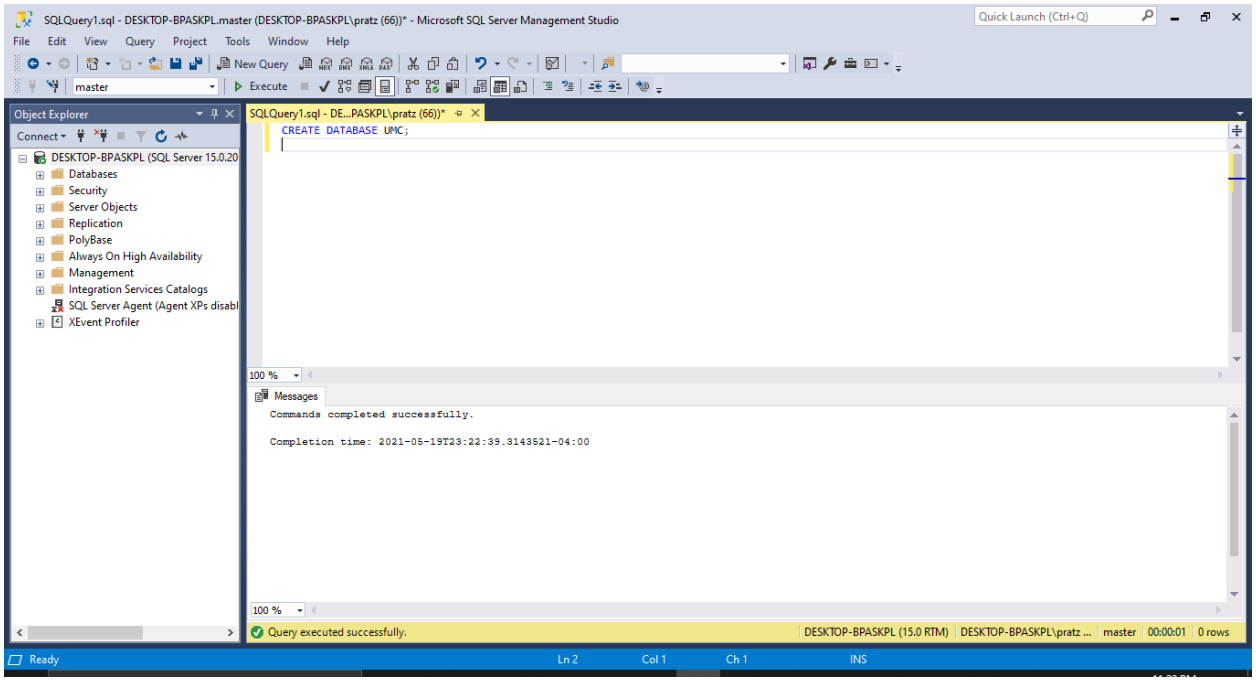
In this phase, the following is **achieved**:

- The tables, columns, primary keys, data types, nullabilities, and relationships are determined
- The design is normalized into its 3rd Normal Form.
- And, potential integrity and security issues are also addressed

//Creating a Database//

Source Code:

```
CREATE DATABASE UMC;
```



Source Code:

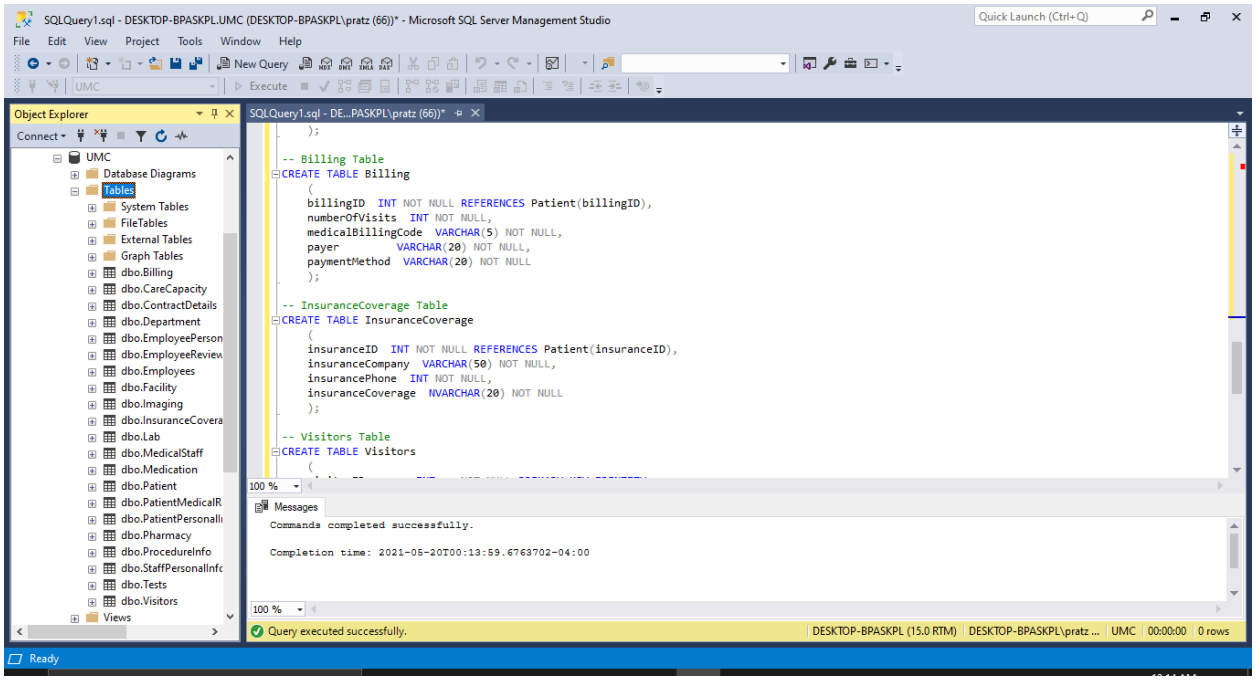
```
/*************************************************************
 This script creates table for database named UMC
*************************************************************
USE UMC;

-- Creates tables for database UMC

-- Facility Table
CREATE TABLE Facility
       (
```

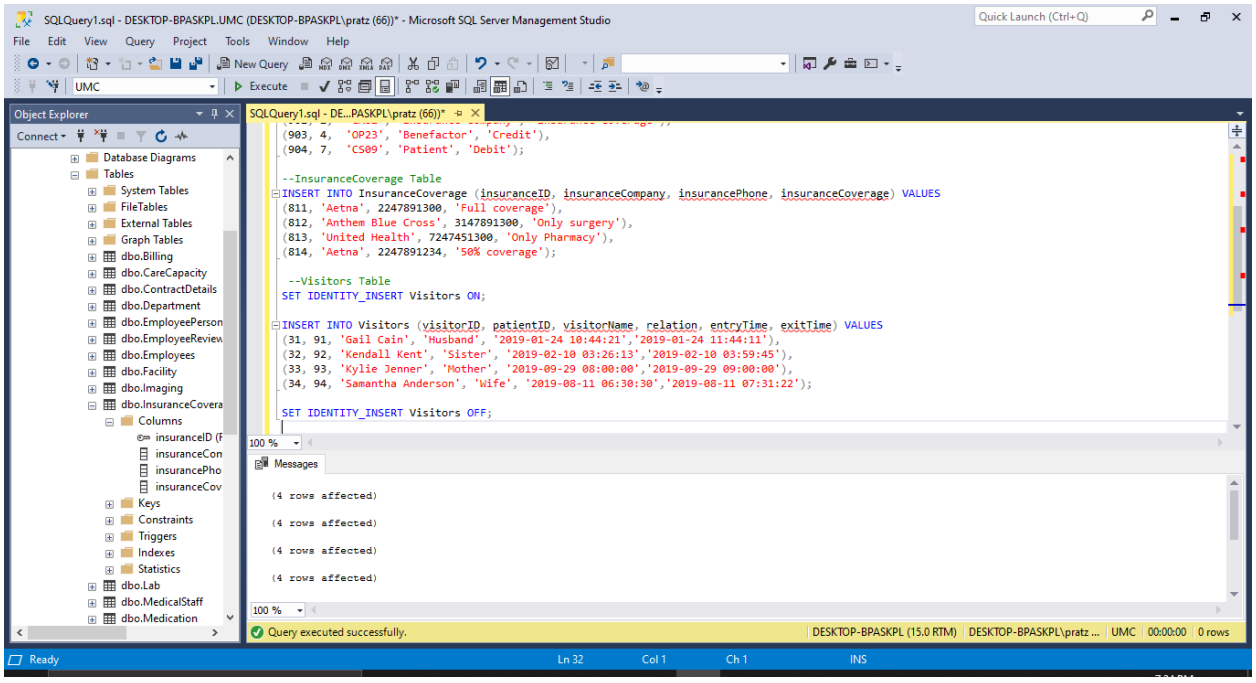
```
	facilityID		INT	NOT NULL PRIMARY KEY,
	facilityName		VARCHAR(50)	NOT NULL,
	location		TEXT NOT NULL,
	hours			VARCHAR(50)	NOT NULL,
	roomCapacity		INT	NOT NULL,
	departmentID		INT	NOT NULL UNIQUE,
	careCapacityLevel	INT	NOT NULL UNIQUE,
	medicalEquipment	TEXT NOT NULL
	);

-- Department Table
CREATE TABLE Department
	(
	departmentID		INT	NOT NULL REFERENCES Facility(departmentID),
	departmentName		VARCHAR(50)	NOT NULL
	);

-- CareCapacity Table
CREATE TABLE CareCapacity
	(
	careCapacityLevel	INT	NOT NULL REFERENCES Facility(careCapacityLevel),
	careCapacityName	VARCHAR(50)		NOT NULL
	);

-- MedicalStaff Table
CREATE TABLE MedicalStaff
	(
	staffID			INT		NOT NULL PRIMARY KEY IDENTITY,
	facilityID		INT		NOT NULL REFERENCES Facility(facilityID),
	staffType		VARCHAR(50) NOT NULL,
	position		VARCHAR(20) NOT NULL,
	status			VARCHAR(20) NOT NULL
	);

-- StaffPersonalInfo Table
CREATE TABLE StaffPersonalInfo
	(
	staffID			INT		NOT NULL REFERENCES MedicalStaff(staffID),
	staffName		VARCHAR(50) NOT NULL,
	staffAddress		NVARCHAR(50) NOT NULL,
	staffPhone		INT		NOT NULL,
	staffEmail		NVARCHAR(50) NOT NULL
	);

-- Employees Table
CREATE TABLE Employees
	(
	employeeID		INT		NOT NULL PRIMARY KEY IDENTITY,
	staffID			INT		NOT NULL REFERENCES MedicalStaff(staffID),
	employeeSalary		MONEY		NOT NULL,
	benefits		VARCHAR(50)	NOT NULL,
	contractID		INT		NOT NULL UNIQUE,
	employeeReviewID	INT		NOT NULL UNIQUE,
	departmentID		INT		NOT NULL REFERENCES Facility(departmentID),
	officeRoom		INT NOT NULL,
	officePhone		INT NOT NULL
	);
```

```
-- EmployeePersonalInfo Table
CREATE TABLE EmployeePersonalInfo
       (
       employeeID        INT       NOT NULL REFERENCES Employees(employeeID),
       employeeName      VARCHAR(50) NOT NULL,
       employeeAddress   NVARCHAR(50) NOT NULL,
       employeePhone       INT     NOT NULL,
       employeeEmail      NVARCHAR(50) NOT NULL
       );

-- ContactDetails Table
CREATE TABLE ContractDetails
       (
       contractID          INT    NOT NULL REFERENCES Employees(contractID),
       contractType      VARCHAR(50)     NOT NULL,
       contractTerm        INT    NOT NULL
       );

-- EmployeeReviewDetails Table
CREATE TABLE EmployeeReviewDetails
       (
       employeeReviewID    INT    NOT NULL REFERENCES Employees(employeeReviewID),
       employeeReview      VARCHAR(70)   NOT NULL,
       reviewerName      VARCHAR(20)     NOT NULL
       );

-- Patient Table
CREATE TABLE Patient
       (
       patientID        INT        NOT NULL PRIMARY KEY IDENTITY,
       insuranceID      INT        NOT NULL UNIQUE,
       primaryCareDoctor VARCHAR(50) NOT NULL,
       staffID           INT       NOT NULL REFERENCES MedicalStaff(staffID),
       billingID           INT     NOT NULL UNIQUE
       );

-- PatientPersonalInfo Table
CREATE TABLE PatientPersonalInfo
       (
       patientID           INT    NOT NULL REFERENCES Patient(patientID),
       patientName      VARCHAR(50) NOT NULL,
       patientAddress   NVARCHAR(50) NOT NULL,
       patientPhone        BIGINT         NOT NULL,
       patientDOB        DATE NOT NULL
       );

-- PatientMedicalRecords Table
CREATE TABLE PatientMedicalRecords
       (
       patientRecordNumber BIGINT NOT NULL PRIMARY KEY IDENTITY,
       patientID           INT    NOT NULL REFERENCES Patient(patientID),
       testID        INT    NOT NULL REFERENCES Lab(testID),
       weight        INT NOT NULL,
       height        INT NOT NULL,
       vitals        INT NOT NULL,
       checkIn       SMALLDATETIME     NOT NULL,
```

```
        checkOut      SMALLDATETIME     NOT NULL,
        symptoms      NVARCHAR(100) NOT NULL,
        diagnose      NVARCHAR(50)   NOT NULL,
        procedureCode    INT NOT NULL UNIQUE,
        attendingPhysician  VARHCAR(20)  NOT NULL,
        referralDoctor  VARHCAR(20)  NOT NULL,
        medicationID INT NOT NULL UNIQUE
        );

-- Lab Table
CREATE TABLE Lab
        (
        testID        INT    NOT NULL PRIMARY KEY IDENTITY,
        results      VARCHAR(50) NOT NULL,
        mandatedReportCounty   VARCHAR(20) NOT NULL,
        mandatedReportState    VARCHAR(20)       NOT NULL
        );

-- Tests Table
CREATE TABLE Tests
        (
        testID          INT NOT NULL REFERENCES Lab(testID),
        testName        VARCHAR(50) NOT NULL
        );

-- Imaging Table
CREATE TABLE Imaging
        (
        testID          INT NOT NULL REFERENCES Lab(testID),
        imagingLocation  NVARCHAR(50) NOT NULL
        );

-- Medication Table
CREATE TABLE Medication
        (
        medicationID     INT NOT NULL REFERENCES PatientMedicalRecords(medicationID),
        medicineNumber   INT NOT NULL UNIQUE
        );

-- Procedure Table
CREATE TABLE ProcedureInfo
        (
        procedureCode  INT NOT NULL REFERENCES PatientMedicalRecords(procedureCode),
        procedureName   VARCHAR(50) NOT NULL,
        procedureCost   MONEY NOT NULL
        );
-- Pharmacy Table
CREATE TABLE Pharmacy
        (
        medicineNumber  INT NOT NULL REFERENCES Medication(medicineNumber),
        medicineName   VARCHAR(50) NOT NULL,
        medicationCost   MONEY NOT NULL
        );

-- Billing Table
CREATE TABLE Billing
        (
        billingID  INT NOT NULL REFERENCES Patient(billingID),
```

```
        numberOfVisits  INT NOT NULL,
        medicalBillingCode  VARCHAR(5) NOT NULL,
        payer           VARCHAR(20) NOT NULL,
        paymentMethod  VARCHAR(20) NOT NULL
        );

-- InsuranceCoverage Table
CREATE TABLE InsuranceCoverage
        (
        insuranceID  INT NOT NULL REFERENCES Patient(insuranceID),
        insuranceCompany  VARCHAR(50) NOT NULL,
        insurancePhone  BIGINT NOT NULL,
        insuranceCoverage  NVARCHAR(20) NOT NULL
        );

-- Visitors Table
CREATE TABLE Visitors
        (
        visitorID        INT        NOT NULL PRIMARY KEY IDENTITY,
        patientID        INT        NOT NULL REFERENCES Patient(patientID),
        visitorName      VARCHAR(50) NOT NULL,
        relation   NVARCHAR(50) NOT NULL,
        entryTime      TIMESTAMP    NOT NULL,
        exitTime       TIMESTAMP NOT NULL
        );
```



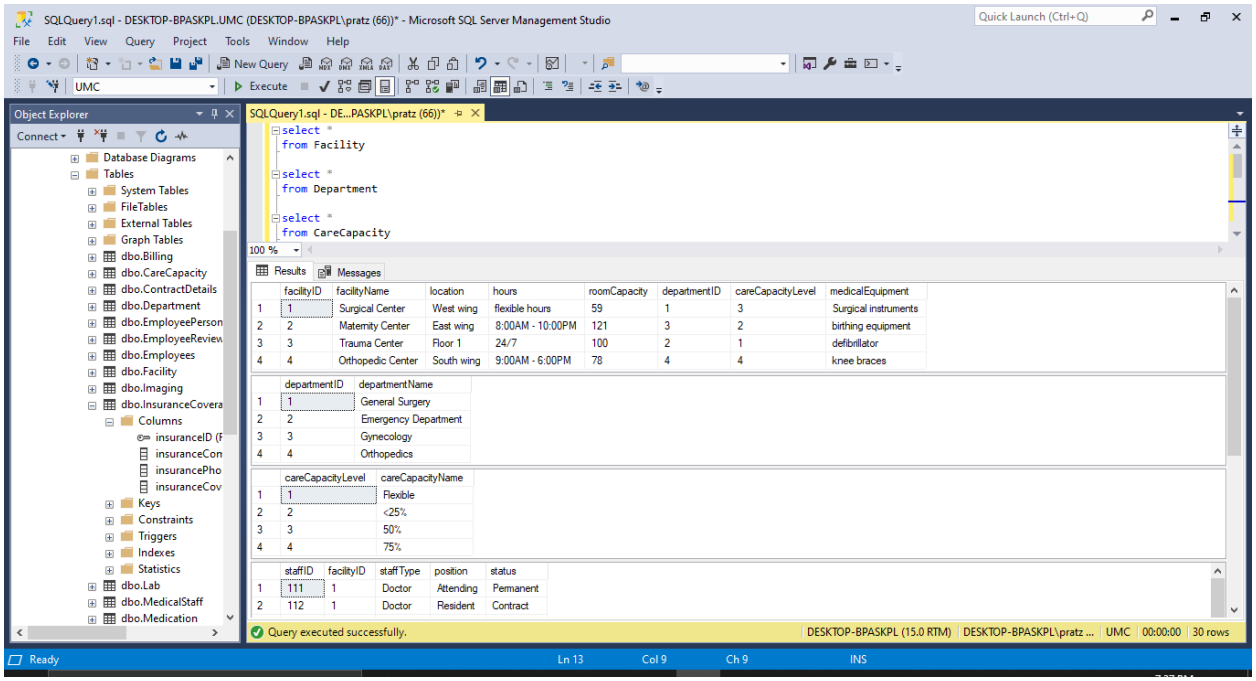


Source Code:

//The script shows the source code for the insertion of values in the tables. //

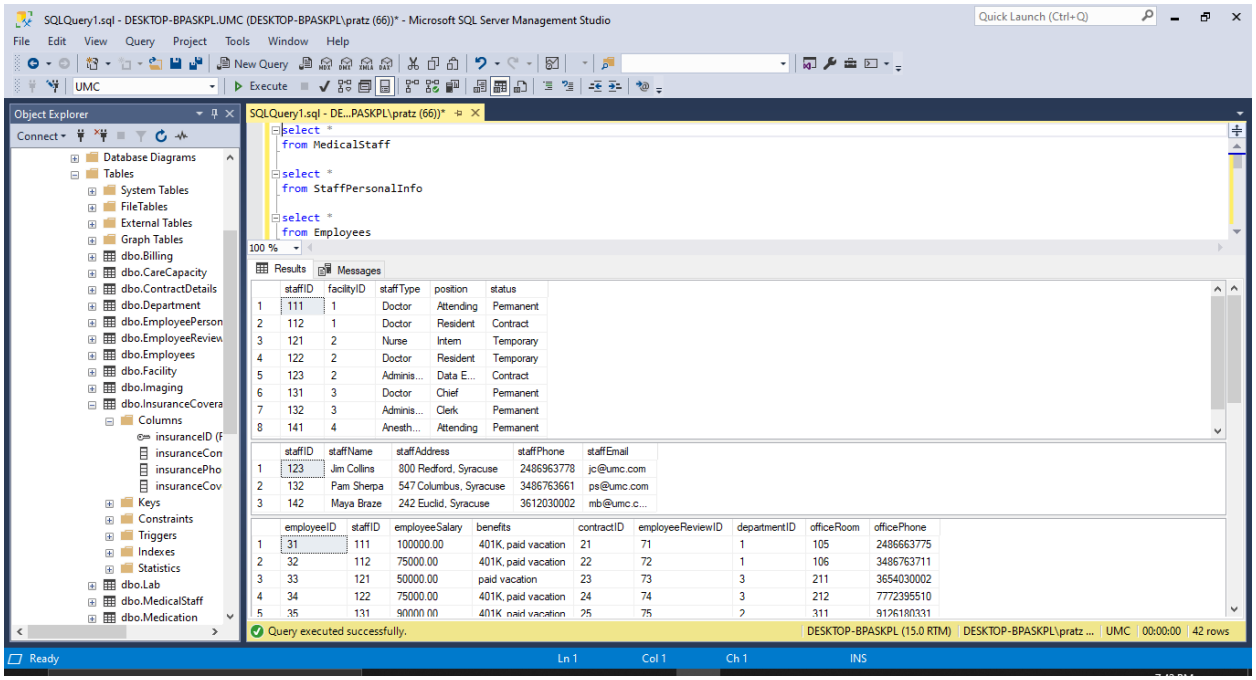
```
-- Insert data into the tables

--Facility Table
SET IDENTITY_INSERT Facility ON;
```

```
INSERT INTO Facility (facilityID, facilityName, location, hours, roomCapacity,
departmentID, careCapacityLevel, medicalEquipment) VALUES
(1, 'Surgical Center', 'West wing', 'flexible hours', 59, 01, 3, 'Surgical instruments'),
(2, 'Maternity Center', 'East wing', '8:00AM - 10:00PM', 121, 03, 2, 'birthing
equipment'),
(3, 'Trauma Center', 'Floor 1', '24/7', 100, 02, 1, 'defibrillator'),
(4, 'Orthopedic Center', 'South wing', '9:00AM - 6:00PM', 78, 04, 2, 'knee braces');

SET IDENTITY_INSERT Facility OFF;

--Department Table
SET IDENTITY_INSERT Department ON;

INSERT INTO Department (departmentID, departmentName) VALUES
(01, 'General Surgery'),
(02, 'Emergency Department'),
(03, 'Gynecology'),
(04, 'Orthopedics');

SET IDENTITY_INSERT Department OFF;

--CareCapacity Table
SET IDENTITY_INSERT CareCapacity ON;

INSERT INTO CareCapacity (careCapacityLevel, careCapacityName) VALUES
(1, 'Flexible'),
(2, '<25%'),
(3, '50%'),
(4, '75%'),
(5, 'Maxed out');

SET IDENTITY_INSERT CareCapacity OFF;
```

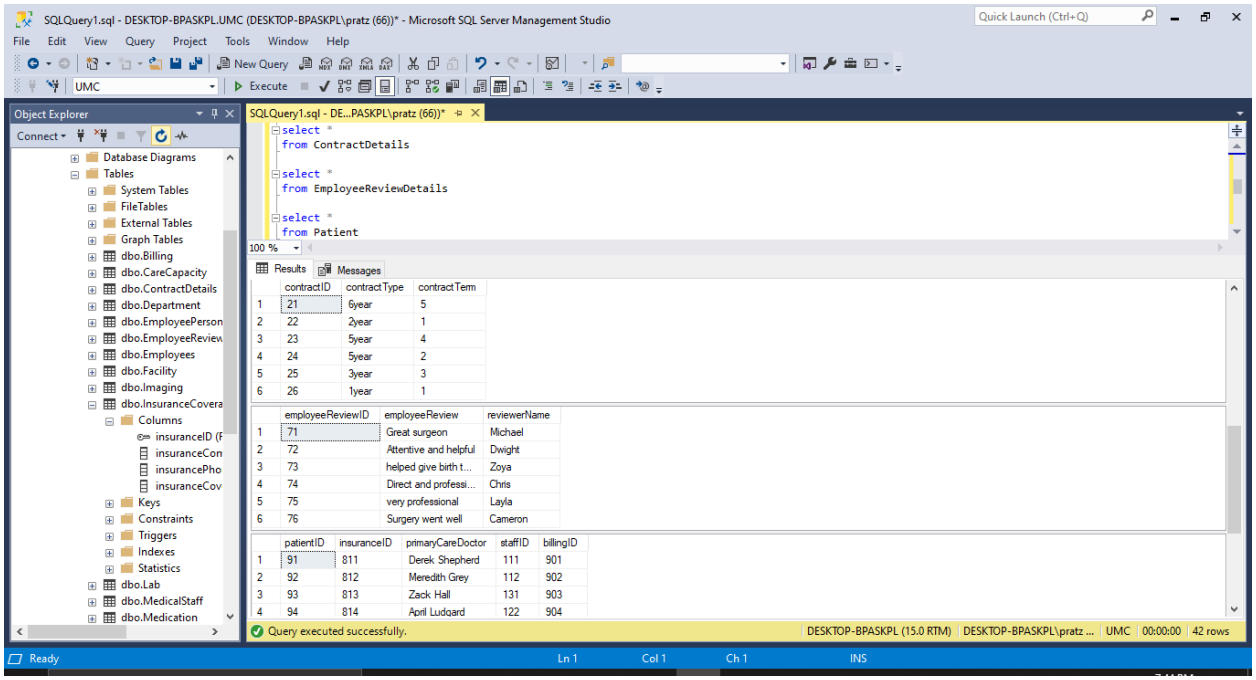



```
--MedicalStaff Table
SET IDENTITY_INSERT MedicalStaff ON;
```

```
INSERT INTO MedicalStaff (staffID, facilityID, staffType, position, status) VALUES
(111, 1, 'Doctor', 'Attending', 'Permanent'),
(112, 1, 'Doctor', 'Resident', 'Contract'),
(121, 2, 'Nurse', 'Intern', 'Temporary'),
(122, 2, 'Doctor', 'Resident', 'Temporary'),
(123, 2, 'Administration', 'Data Entry Operator', 'Contract'),
(131, 3, 'Doctor', 'Chief', 'Permanent'),
(132, 3, ‘Administration’, 'Clerk', 'Permanent'),
(141, 4, 'Anesthesiologist', 'Attending', 'Permanent'),
(142, 4, 'Administration', 'Receptionist', 'Temporary');

SET IDENTITY_INSERT MedicalStaff OFF;

--StaffPersonalInfo Table
SET IDENTITY_INSERT StaffPersonalInfo ON;

INSERT INTO StaffPersonalInfo (staffID, staffName, staffAddress, staffPhone, staffEmail)
VALUES
(123, 'Jim Collins', '800 Redford, Syracuse', ‘2486963775’, 'jc@umc.com'),
(132, 'Pam Sherpa', '547 Columbus, Syracuse', ‘3486763661’, 'ps@umc.com'),
(142, 'Maya Braze', '242 Euclid, Syracuse', ‘3612030002’, 'mb@umc.com');


SET IDENTITY_INSERT StaffPersonalInfo OFF;

--Employees Table
SET IDENTITY_INSERT Employees ON;

INSERT INTO Employees (employeeID, staffID, employeeSalary, benefits, contractID,
employeeReviewID, departmentID, officeRoom, officePhone) VALUES
(31, 111, '100,000.00', '401K, paid vacation', 21, 71, 01, 105, ‘2486663775’),
(32, 112, '75,000.00', '401K, paid vacation', 22, 72, 01, 106, ‘3486763711’),
(33, 121, '50,000.00', 'paid vacation', 23, 73, 03, 211, ‘3654030002’),
(34, 122, '75,000.00', '401K, paid vacation', 24, 74, 03, 212, ‘7772395510’),
(35, 131, '90,000.00', '401K, paid vacation', 25, 75, 02, 311, ‘9126180331’),
(36, 141, '60,000.00', '401K', 26, 76, 04, 412, ‘2478440012’);

SET IDENTITY_INSERT Employees OFF;
```

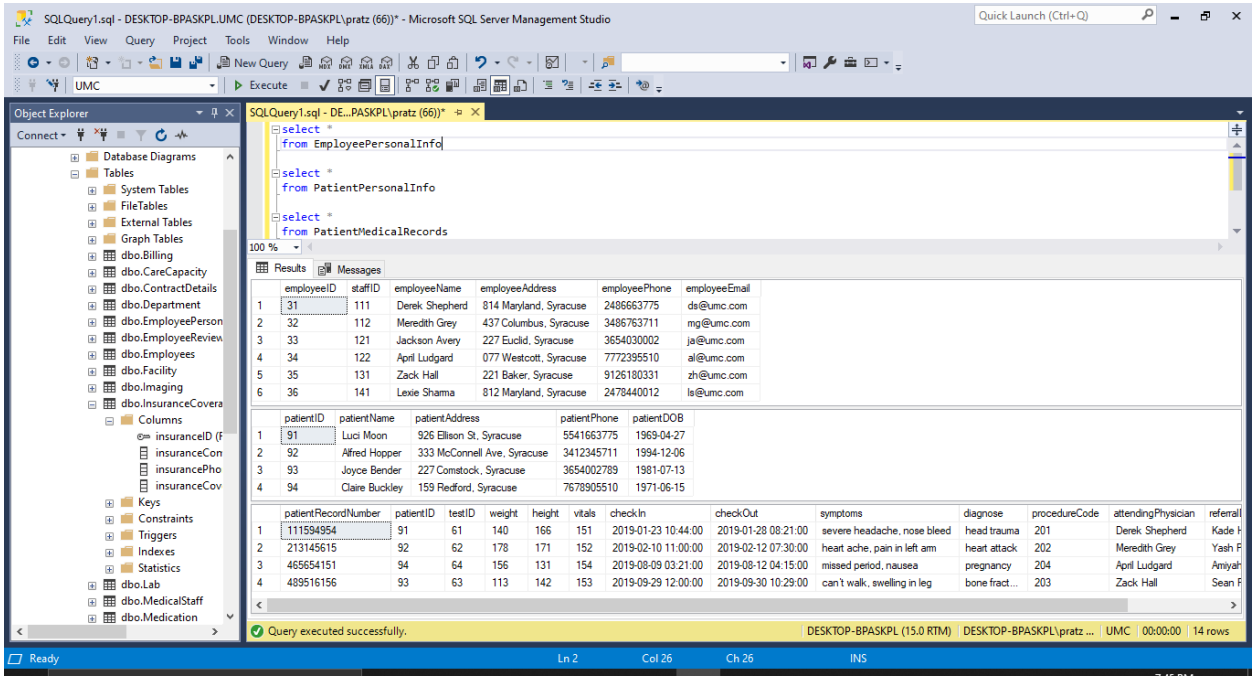



```
--EmployeePersonalInfo Table
SET IDENTITY_INSERT EmployeePersonalInfo ON;

INSERT INTO EmployeePersonalInfo (employeeID, staffID, employeeName, employeeAddress,
employeePhone, employeeEmail) VALUES
(31, 111, 'Derek Shepherd', '814 Maryland, Syracuse', 2486663775, 'ds@umc.com'),
(32, 112, 'Meredith Grey', '437 Columbus, Syracuse', 3486763711, 'mg@umc.com'),
(33, 121, 'Jackson Avery', '227 Euclid, Syracuse', 3654030002, 'ja@umc.com'),
(34, 122, 'April Ludgard', '077 Westcott, Syracuse', 7772395510, 'al@umc.com'),
(35, 131, 'Zack Hall', '221 Baker, Syracuse', 9126180331, 'zh@umc.com'),
(36, 141, 'Lexie Sharma', '812 Maryland, Syracuse', 2478440012, 'ls@umc.com');

SET IDENTITY_INSERT EmployeePersonalInfo OFF;

--ContractDetails Table
INSERT INTO ContractDetails (contractID, contractType, contractTerm) VALUES
(21, '6year', 5),
(22, '2year', 1),
(23, '5year', 4),
(24, '5year', 2),
(25, '3year', 3),
(26, '1year', 1);


--EmployeeReviewDetails Table
INSERT INTO EmployeeReviewDetails(employeeReviewID, employeeReview, reviewerName) VALUES
(71, 'Great surgeon', 'Michael'),
(72, 'Attentive and helpful', 'Dwight'),
(73, 'helped give birth to our beautiful baby', 'Zoya'),
(74, 'Direct and professional', 'Chris'),
(75, 'very professional', 'Layla'),
(76, 'Surgery went well', 'Cameron');
```

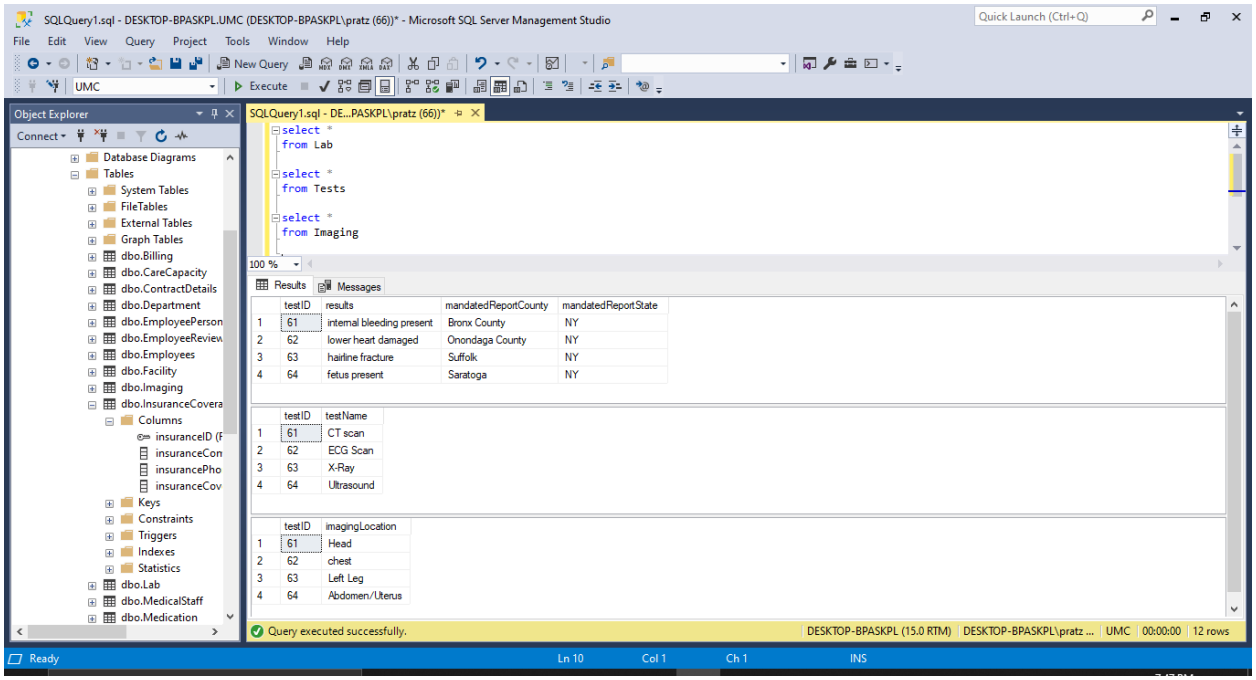



```
--Patient Table
SET IDENTITY_INSERT Patient ON;

INSERT INTO Patient (patientID, insuranceID, primaryCareDoctor, staffID, billingID)
VALUES
(91, 811, 'Derek Shepherd', 111, 901),
(92, 812, 'Meredith Grey', 112, 902),
(93, 813, 'Zack Hall', 131, 903),
(94, 814, 'April Ludgard', 122, 904);

SET IDENTITY_INSERT Patient OFF;

--PatientPersonalInfo Table

INSERT INTO PatientPersonalInfo (patientID, patientName, patientAddress, patientPhone,
patientDOB) VALUES
(91, 'Luci Moon', '926 Ellison St, Syracuse', 5541663775, '1969-04-27'),
(92, 'Alfred Hopper', '333 McConnell Ave, Syracuse', 3412345711, '1994-12-06'),
(93, 'Joyce Bender', '227 Comstock, Syracuse', 3654002789, '1981-07-13'),
(94, 'Claire Buckley', '159 Redford, Syracuse', 7678905510, '1971-06-15');



--PatientMedicalRecords Table

SET IDENTITY_INSERT PatientMedicalRecords ON;

INSERT INTO PatientMedicalRecords (patientRecordNumber, patientID, testID, weight,
height, vitals, checkIn, checkOut, symptoms, diagnose, procedureCode, attendingPhysician,
referralDoctor, medicationID)
VALUES
(111594954,91,61, 140, 166,151,'2019-01-23 10:44:21','2019-01-28 08:21:11','severe
headache, nose bleed','head trauma', 201, 'Derek Shepherd', 'Kade Haley',345),
```

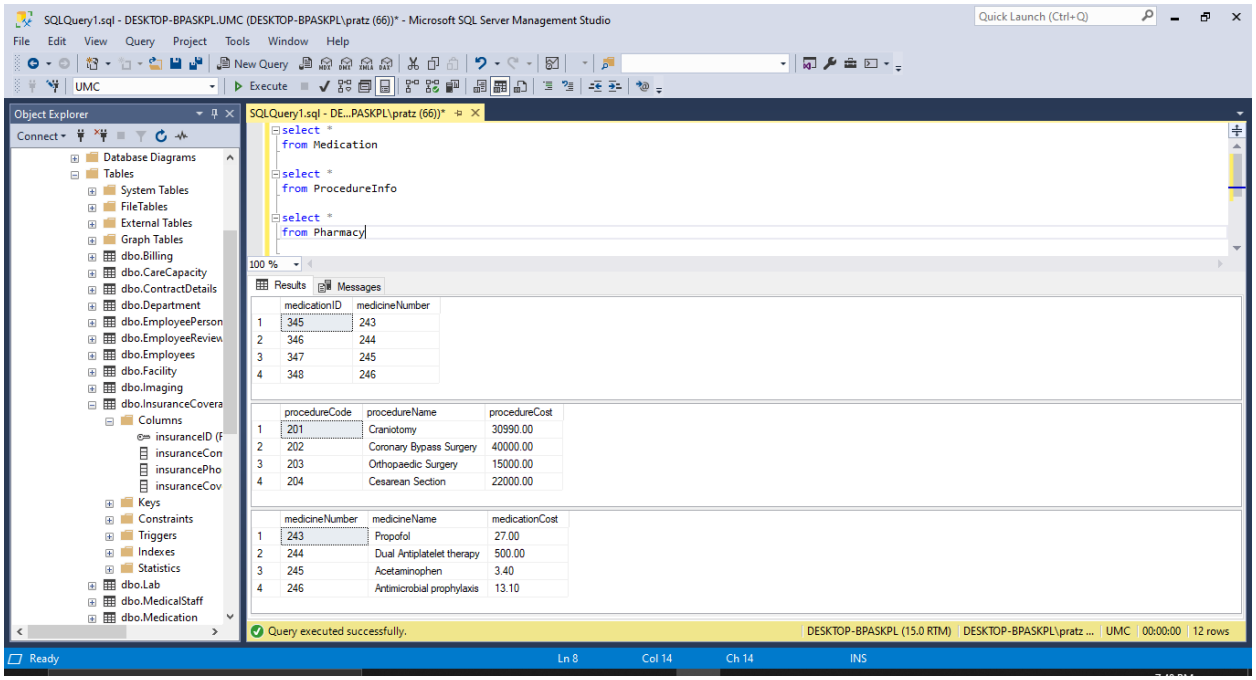
```
(213145615, 92,62, 178, 171, 152,'2019-02-10 11:00:08','2019-02-12 07:30:20','heart ache, pain in left arm','heart attack', 202, 'Meredith Grey', 'Yash Patil',346),
(489516156,93,63,113.2,142,153,'2019-09-29 12:00:00','2019-09-30 10:29:21','can't walk, swelling in leg','bone fracture', 203, 'Zack Hall', 'Sean Ryan',347),
(465654151, 94,64, 156.1,131,154,'2019-08-09 03:21:00','2019-08-12 04:15:15','missed period, nausea','pregnancy', 204, 'April Ludgard', 'Amiyah beech',348);

SET IDENTITY_INSERT PatientMedicalRecords OFF;
```



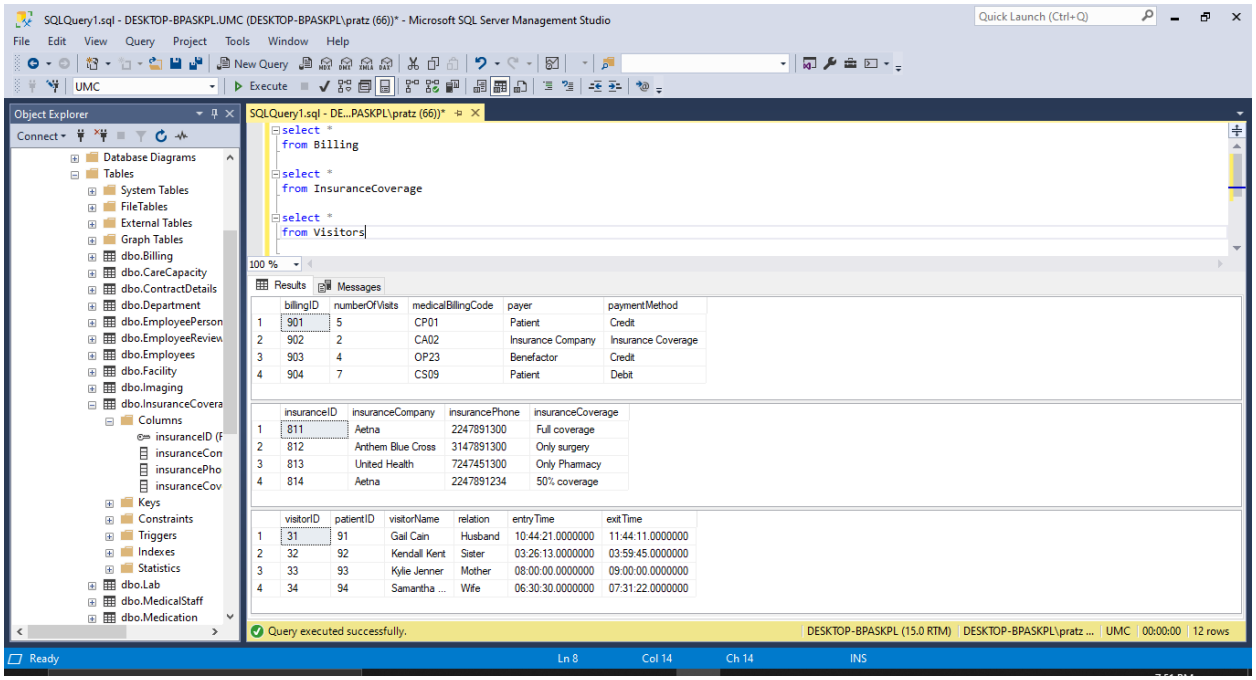

```
--Lab Table
SET IDENTITY_INSERT Lab ON;

INSERT INTO Lab (testID, results, mandatedReportCounty, mandatedReportState) VALUES
(61, 'internal bleeding present', 'Bronx County', 'NY'),
(62, 'lower heart damaged', 'Onondaga County', 'NY'),
(63, 'hairline fracture', 'Suffolk', 'NY'),
(64, 'fetus present', 'Saratoga', 'NY');

SET IDENTITY_INSERT Lab OFF;

--Tests Table
INSERT INTO Tests (testID, testName) VALUES
(61, 'CT scan'),
(62, 'ECG Scan'),
(63, 'X-Ray'),
(64, 'Ultrasound');

--Imaging Table
INSERT INTO Imaging (testID, imagingLocation) VALUES
(61, 'Head'),
(62, 'chest'),
(63, 'Left Leg'),
(64, 'Abdomen/Uterus');
```




```sql
--Medication Table
INSERT INTO Medication (medicationID, medicineNumber) VALUES
(345, 243),
(346, 244),
(347, 245),
(348, 246);

--Procedure Table
INSERT INTO ProcedureInfo (procedureCode, procedureName, procedureCost) VALUES
(201, 'Craniotomy', '30,990.00'),
(202, 'Coronary Bypass Surgery', '40,000.00'),
(203, 'Orthopaedic Surgery', '15,000.00'),
(204, 'Cesarean Section', '22,000.00');

--Pharmacy Table
INSERT INTO Pharmacy (medicineNumber, medicineName, medicationCost) VALUES
(243, 'Propofol', '27.00'),
(244, 'Dual Antiplatelet therapy', '500.00'),
(245, 'Acetaminophen', '3.40'),
(246, 'Antimicrobial prophylaxis', '13.10');
```




```
--Billing Table
INSERT INTO Billing (billingID, numberOfVisits, medicalBillingCode, payer, paymentMethod)
VALUES
(901, 5,  'CP01', 'Patient', 'Credit'),
(902, 2,  'CA02', 'Insurance Company', 'Insurance Coverage'),
(903, 4,  'OP23', 'Benefactor', 'Credit'),
(904, 7,  'CS09', 'Patient', 'Debit');

--InsuranceCoverage Table
INSERT INTO InsuranceCoverage (insuranceID, insuranceCompany, insurancePhone,
insuranceCoverage) VALUES
(811, 'Aetna', 2247891300, 'Full coverage'),
(812, 'Anthem Blue Cross', 3147891300, 'Only surgery'),
(813, 'United Health', 7247451300, 'Only Pharmacy'),
(814, 'Aetna', 2247891234, '50% coverage');

 --Visitors Table
SET IDENTITY_INSERT Visitors ON;

INSERT INTO Visitors (visitorID, patientID, visitorName, relation, entryTime, exitTime)
VALUES
(31, 91, 'Gail Cain', 'Husband', '2019-01-24 10:44:21','2019-01-24 11:44:11'),
(32, 92, 'Kendall Kent', 'Sister', '2019-02-10 03:26:13','2019-02-10 03:59:45'),
(33, 93, 'Kylie Jenner', 'Mother', '2019-09-29 08:00:00','2019-09-29 09:00:00'),
(34, 94, 'Samantha Anderson', 'Wife', '2019-08-11 06:30:30','2019-08-11 07:31:22');

SET IDENTITY_INSERT Visitors OFF;
```



TESTING

This phase includes the following:

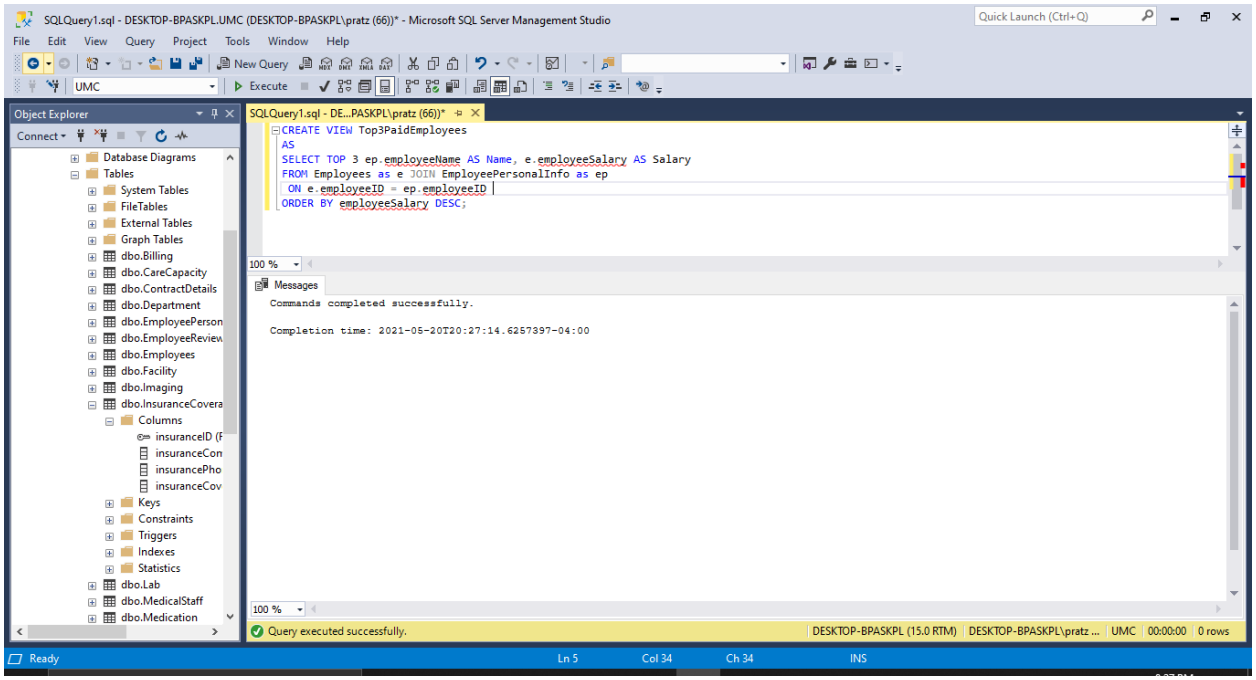
- Populating the database with test data
- Producing various reports
- Demonstrating the reliability through several scenarios
- Performing several transactions.

First View

We are going to create a view, Top3PaidEmployees, that returns the top 3 employees that get paid the highest. For each employee, the view will return employeeID, employeeName and employeeSalary. Return only 3 employees with the highest salaries. Then we are using a select statement to show the results.

CODE:

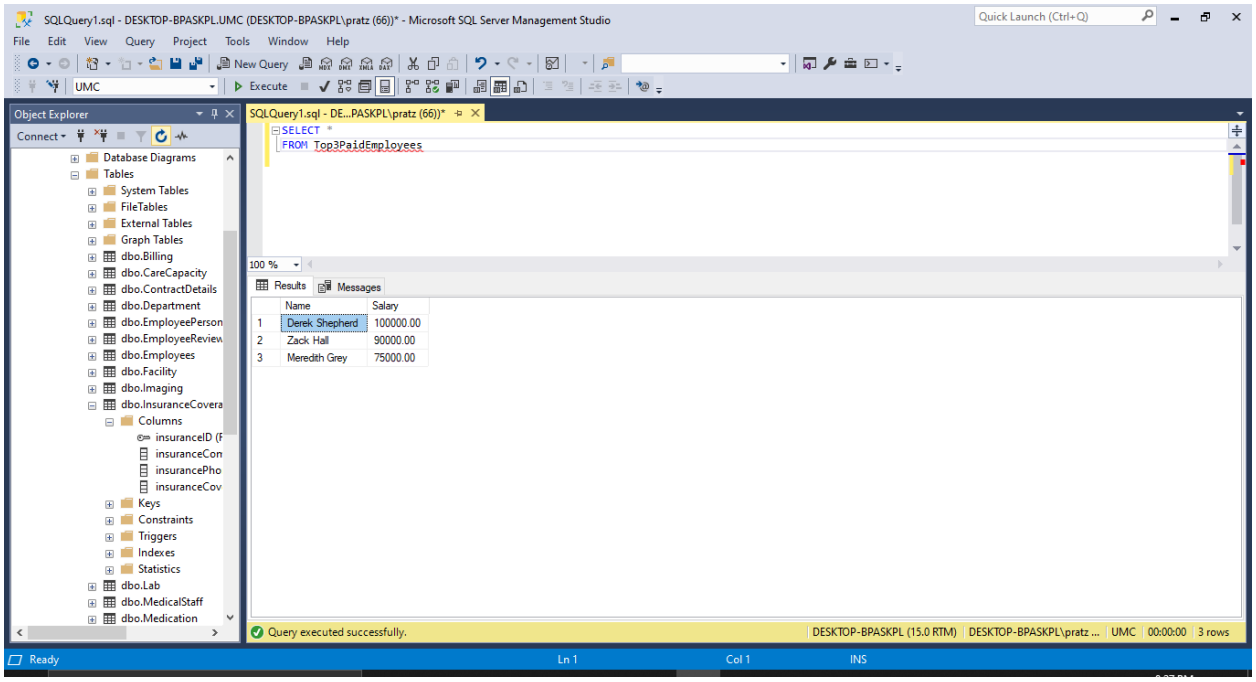
```
CREATE VIEW Top3PaidEmployees
AS
SELECT TOP 3 e.employeeName AS Name, ep.employeeSalary AS Salary
FROM Employees AS e JOIN EmployeePersonalInfo AS ep
 ON e.employeeID = ep.employeeID
ORDER BY employeeSalary DESC;
```



RESULT

SELECT *

FROM Top3PaidEmployees

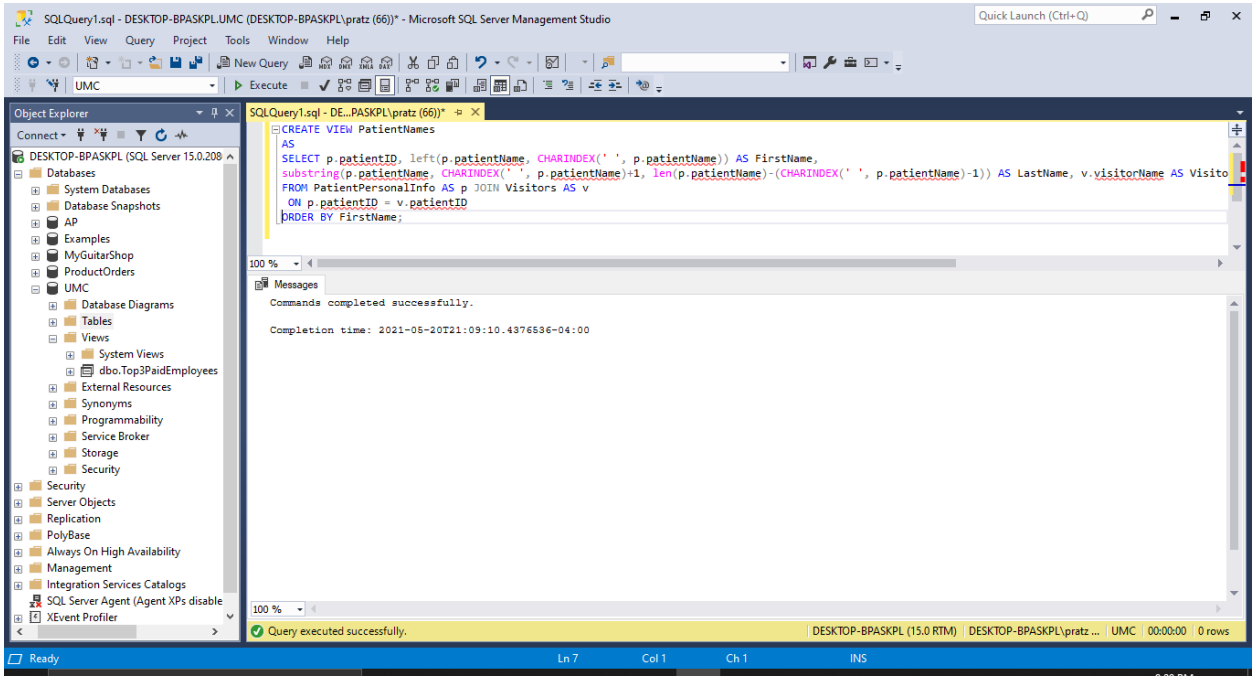


Second View

We are going to create a view, PatientNames, that returns the name of patients as their first name and last name separately. The view should also return their visitors and the relationship between the patient and the visitor. The rows should be ordered by the patient's first name alphabetically. Then we are using a select statement to show the results.

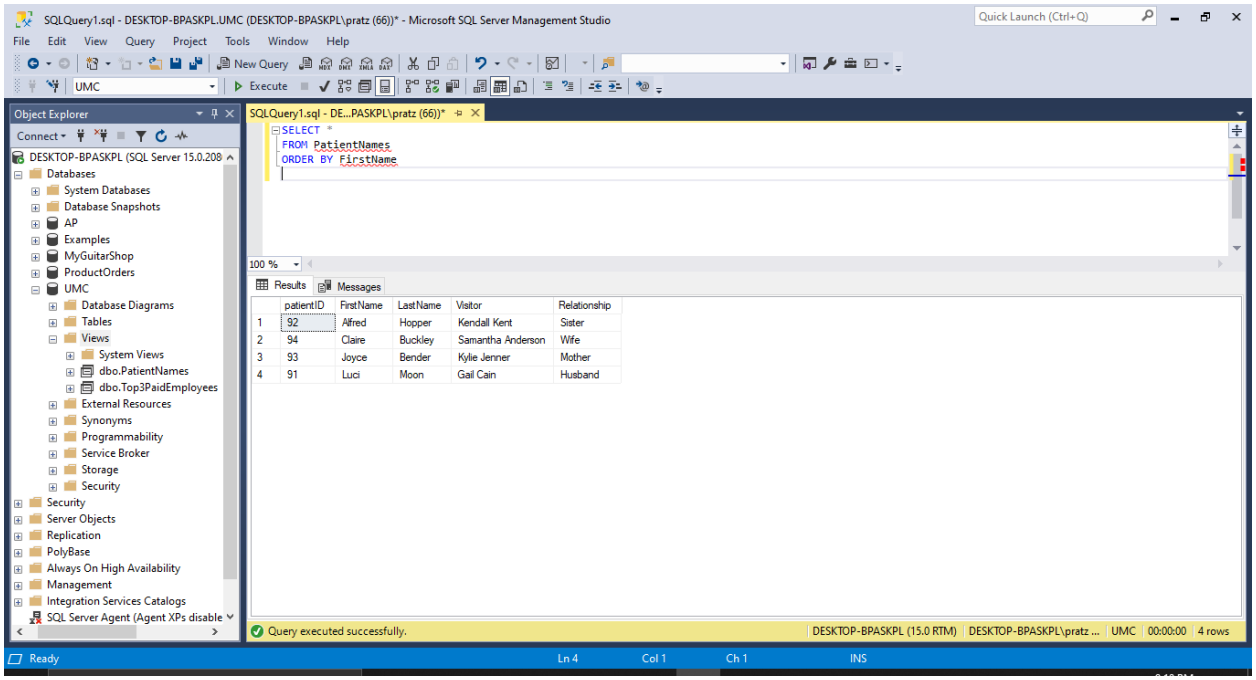
CODE:

```
CREATE VIEW PatientNames
AS
SELECT p.patientID, left(p.patientName, CHARINDEX(' ', p.patientName)) AS
FirstName,
substring(p.patientName, CHARINDEX(' ', p.patientName)+1,
len(p.patientName)-(CHARINDEX(' ', p.patientName)-1)) AS LastName,
v.visitorName AS Visitor, v.relation AS Relationship
FROM PatientPersonalInfo AS p JOIN Visitors AS v
 ON p.patientID = v.patientID
ORDER BY FirstName;
```



RESULT

SELECT *
FROM PatientNames
ORDER BY FirstName

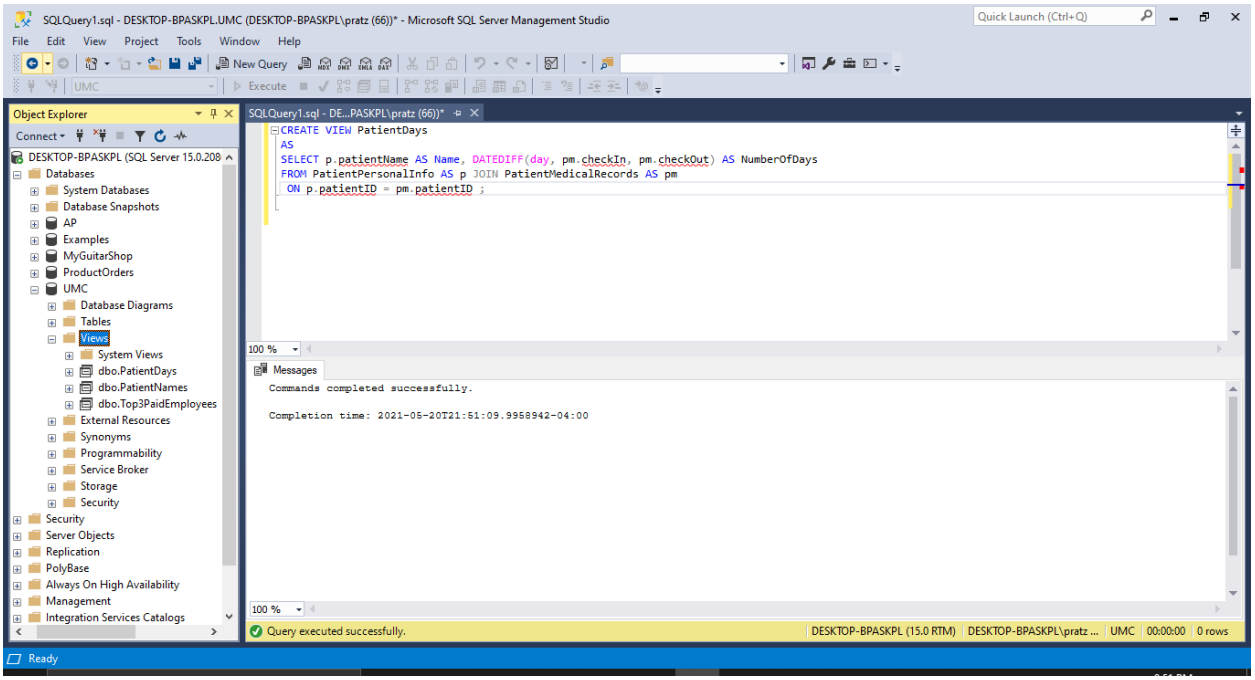


Third View

We are going to create a view that returns how many total days each patient was checked in to the hospital. The view should return two columns, the patient name and the days they were at the hospital. The rows should be ordered in with more to less number of days. Then we are using a select statement to show the results.

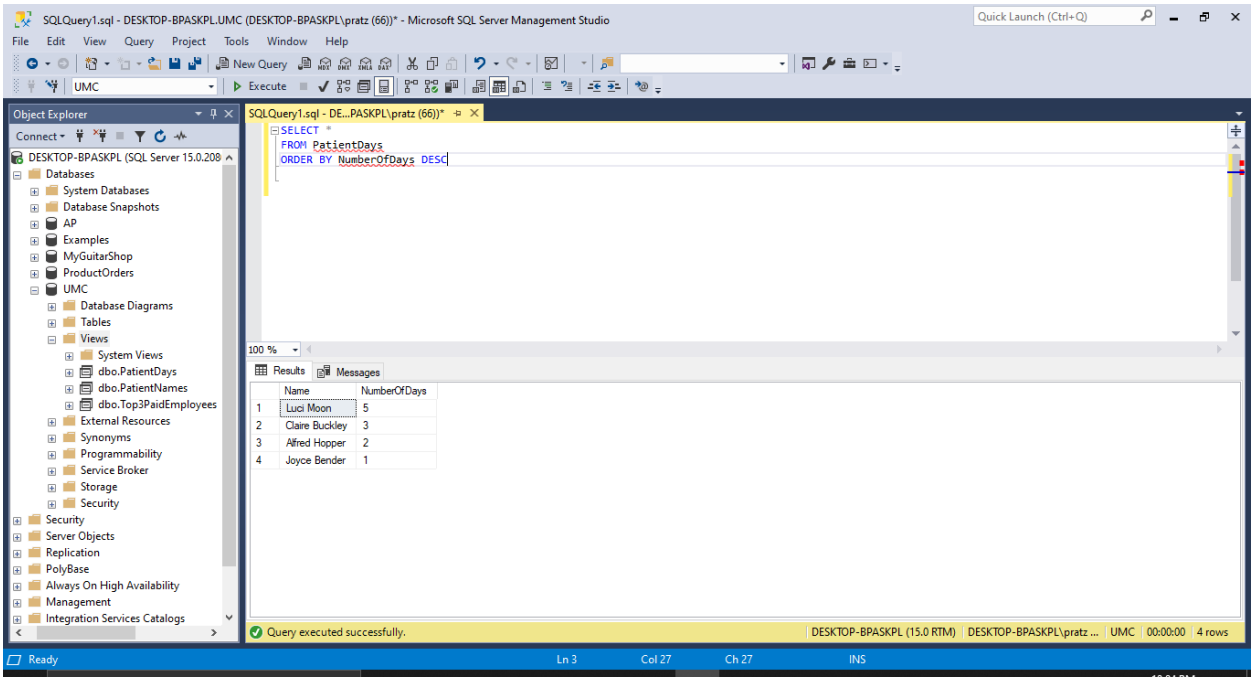
CODE:

```
CREATE VIEW PatientDays
AS
SELECT p.patientName AS Name, DATEDIFF(day, pm.checkIn, pm.checkOut) AS
NumberOfDays
FROM PatientPersonalInfo AS p JOIN PatientMedicalRecords AS pm
 ON p.patientID = pm.patientID;
```



RESULT

SELECT *
FROM PatientDays
ORDER BY NumberOfDays DESC;

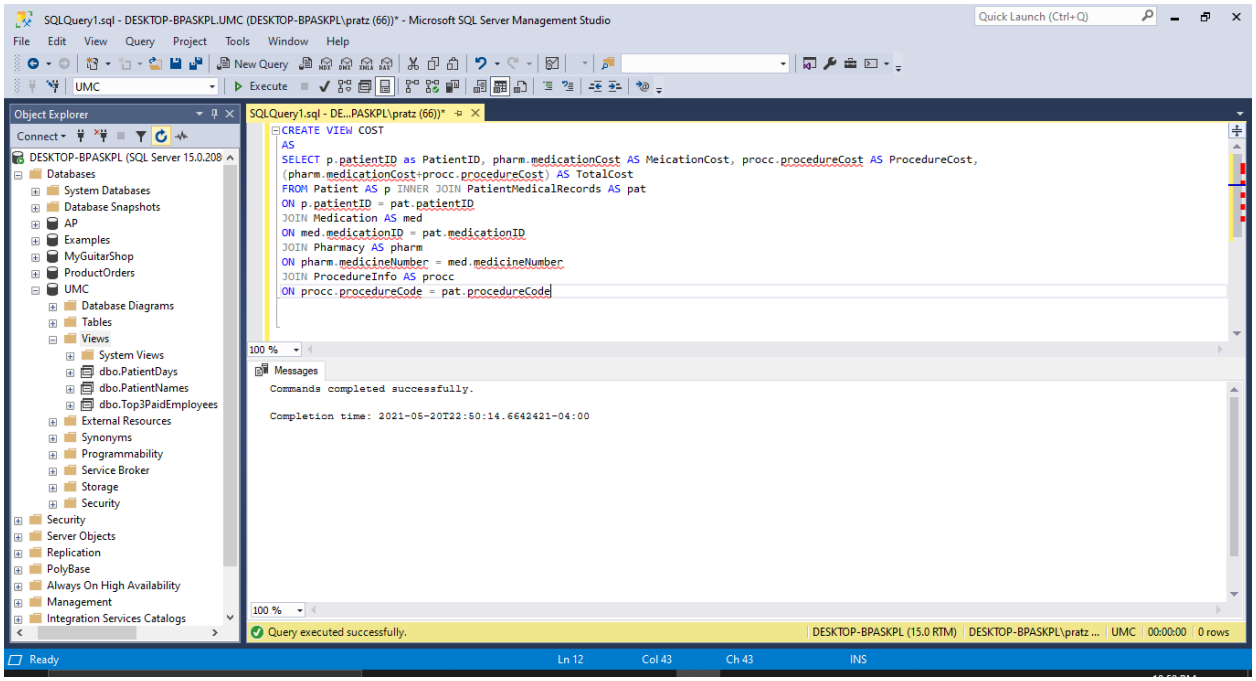


Fourth View

We will create a view that gives the total cost charged to every patient for their medications and procedures. Here, we are using 5 tables as they have been normalized to 3NF and have incorporated various JOINS. The final result gives the total cost.

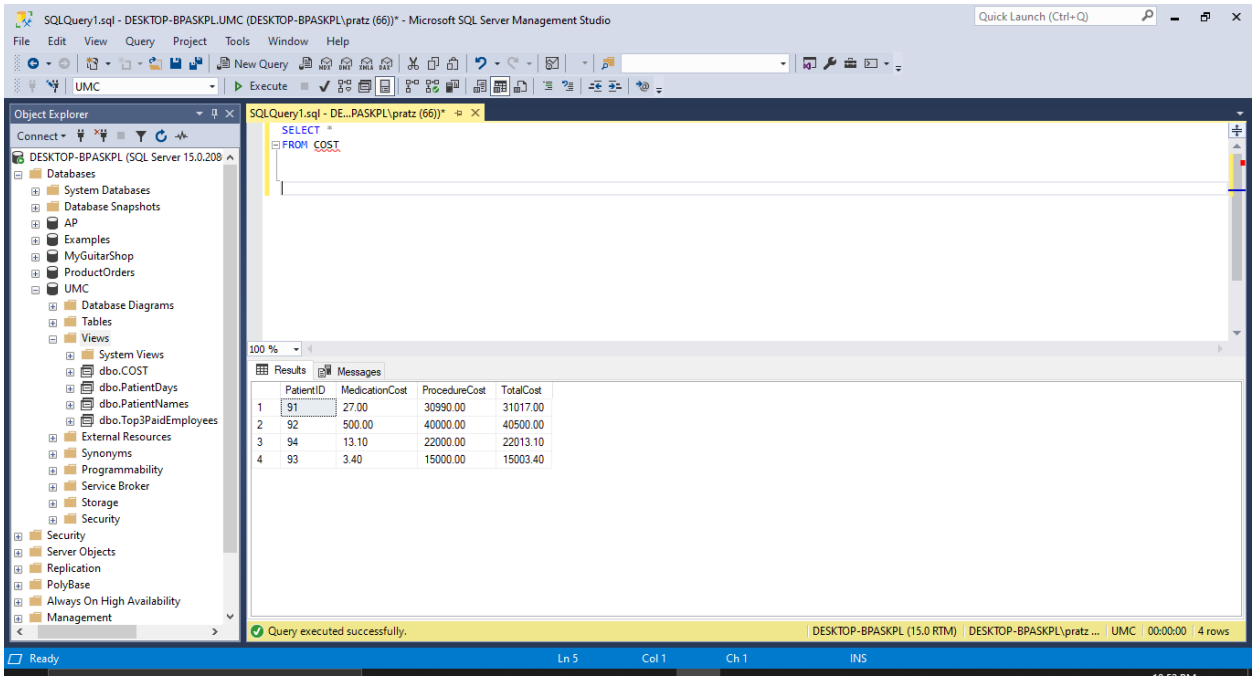
CODE:

```
CREATE VIEW COST
AS
SELECT p.patientID as PatientID, pharm.medicationCost AS MedicationCost,
procc.procedureCost AS ProcedureCost,
(pharm.medicationCost+procc.procedureCost) AS TotalCost
FROM Patient AS p INNER JOIN PatientMedicalRecords AS pat
ON p.patientID = pat.patientID
JOIN Medication AS med
ON med.medicationID = pat.medicationID
JOIN Pharmacy AS pharm
ON pharm.medicineNumber = med.medicineNumber
JOIN ProcedureInfo AS procc
ON procc.procedureCode = pat.procedureCode
```



RESULT

```
SELECT *
FROM COST
```

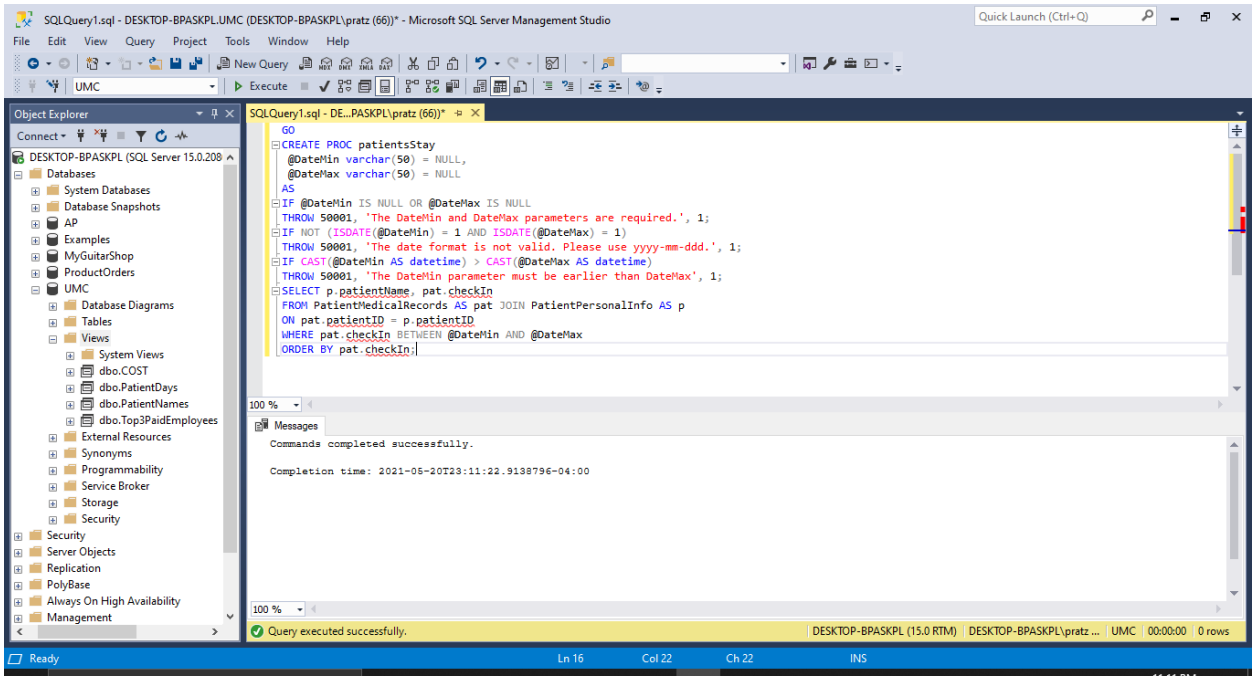


First Stored Procedure

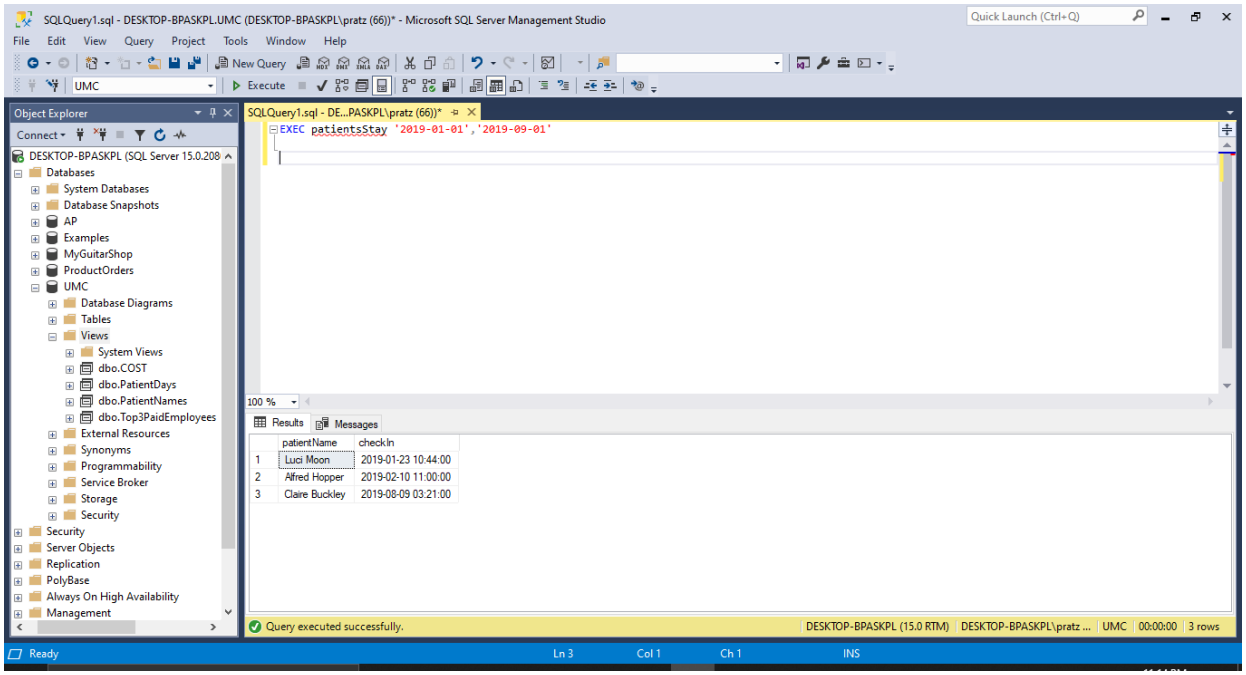
We will create a stored procedure named patientsStay that checks whether a patient is checked in at the hospital between these dates. This procedure accepts two parameters, @DateMin and @DateMax, with data type varchar and default value null. If called with no parameters or with null values, raise an error that describes the problem. If called with non-null values, validate the parameters. Test that the literal strings are valid dates and test that @DateMin is earlier than @DateMax. If the parameters are valid, return a result set that includes the patient name and the check in date.

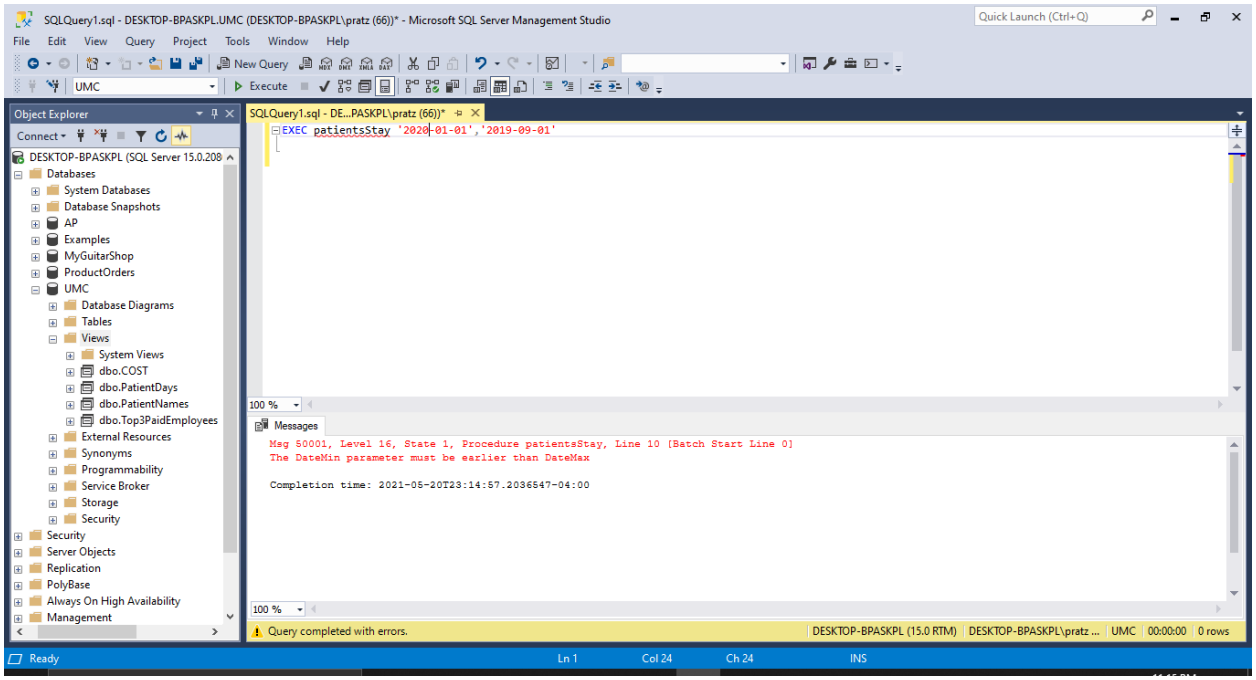
CODE:

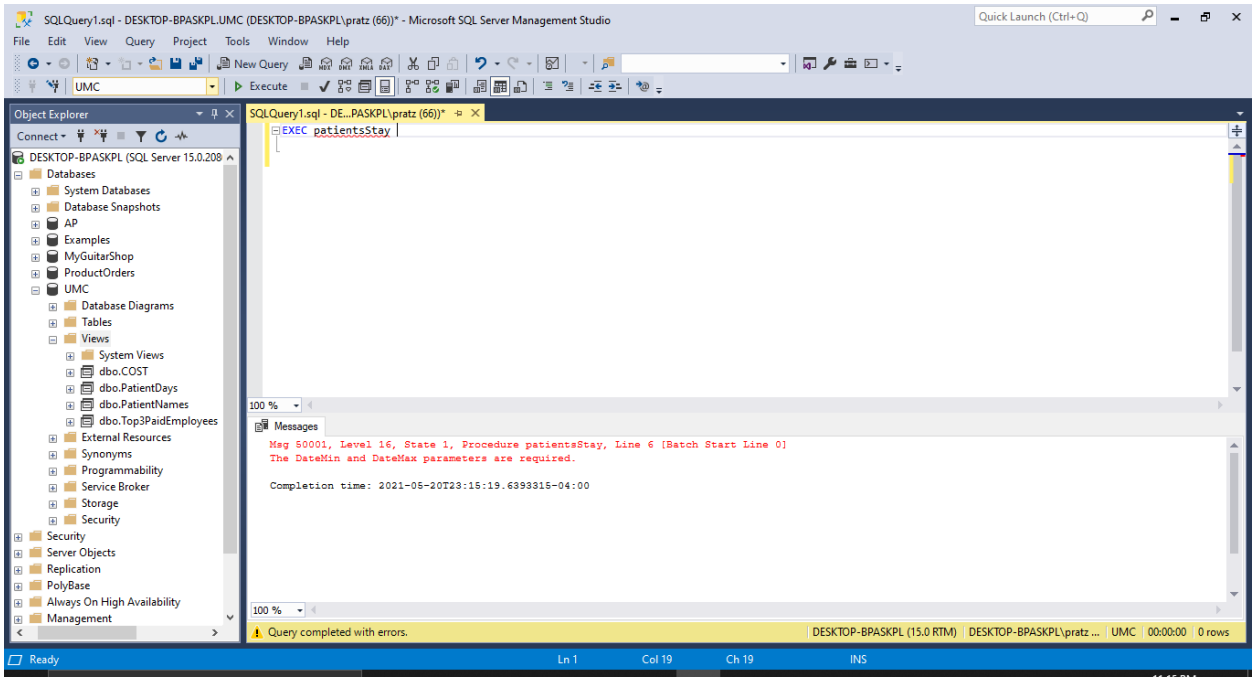
```
GO
CREATE PROC patientsStay
 @DateMin varchar(50) = NULL,
 @DateMax varchar(50) = NULL
AS
IF @DateMin IS NULL OR @DateMax IS NULL
        THROW 50001, 'The DateMin and DateMax parameters are required.', 1;
IF NOT (ISDATE(@DateMin) = 1 AND ISDATE(@DateMax) = 1)
        THROW 50001, 'The date format is not valid. Please use yyyy-mm-dd.', 1;
IF CAST(@DateMin AS datetime) > CAST(@DateMax AS datetime)
       THROW 50001, 'The DateMin parameter must be earlier than DateMax', 1;
SELECT p.patientName, pat.checkIn
FROM PatientMedicalRecords AS pat JOIN PatientPersonalInfo AS p
ON pat.patientID = p.patientID
WHERE pat.checkIn BETWEEN @DateMin AND @DateMax
ORDER BY pat.checkIn;
```


SQLQuery1.sql - DESKTOP-BPASKPL.UMC (DESKTOP-BPASKPL\pratz (66))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
Execute
UMC
Object Explorer
Connect
DESKTOP-BPASKPL (SQL Server 15.0.208
Databases
System Databases
Database Snapshots
AP
Examples
MyGuitarShop
ProductOrders
UMC
Database Diagrams
Tables
Views
System Views
dbo.COST
dbo.PatientDays
dbo.PatientNames
dbo.Top3PaidEmployees
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
SQLQuery1.sql - DE...PASKPL\pratz (66))*
GO
CREATE PROC patientsStay
@DateMin varchar(50) = NULL,
@DateMax varchar(50) = NULL
AS
IF @DateMin IS NULL OR @DateMax IS NULL
THROW 50001, 'The DateMin and DateMax parameters are required.', 1;
IF NOT (ISDATE(@DateMin) = 1 AND ISDATE(@DateMax) = 1)
THROW 50001, 'The date format is not valid. Please use yyyy-mm-ddd.', 1;
IF CAST(@DateMin AS datetime) > CAST(@DateMax AS datetime)
THROW 50001, 'The DateMin parameter must be earlier than DateMax', 1;
SELECT p.patientName, pat.checkIn
FROM PatientMedicalRecords AS pat JOIN PatientPersonalInfo AS p
ON pat.patientID = p.patientID
WHERE pat.checkIn BETWEEN @DateMin AND @DateMax
ORDER BY pat.checkIn;
100 %
Messages
Commands completed successfully.
Completion time: 2021-05-20T23:11:22.9138796-04:00
100 %
Query executed successfully.
DESKTOP-BPASKPL (15.0 RTM)
DESKTOP-BPASKPL\pratz ...
UMC
00:00:00
0 rows
Ready
Ln 16
Col 22
Ch 22
INS

RESULTS


SQLQuery1.sql - DESKTOP-BPASKPL.UMC (DESKTOP-BPASKPL\pratz (66))* - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
Execute
UMC
Object Explorer
Connect
DESKTOP-BPASKPL (SQL Server 15.0.208
Databases
System Databases
Database Snapshots
AP
Examples
MyGuitarShop
ProductOrders
UMC
Database Diagrams
Tables
Views
System Views
dbo.COST
dbo.PatientDays
dbo.PatientNames
dbo.Top3PaidEmployees
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
SQLQuery1.sql - DE...PASKPL\pratz (66))*
EXEC patientsStay '2019-01-01','2019-09-01'
100 %
Results
Messages
patientName
checkIn
1 Luci Moon 2019-01-23 10:44:00
2 Alfred Hopper 2019-02-10 11:00:00
3 Claire Buckley 2019-08-09 03:21:00
Query executed successfully.
DESKTOP-BPASKPL (15.0 RTM)
DESKTOP-BPASKPL\pratz ...
UMC
00:00:00
3 rows
Ready
Ln 3
Col 1
Ch 1
INS



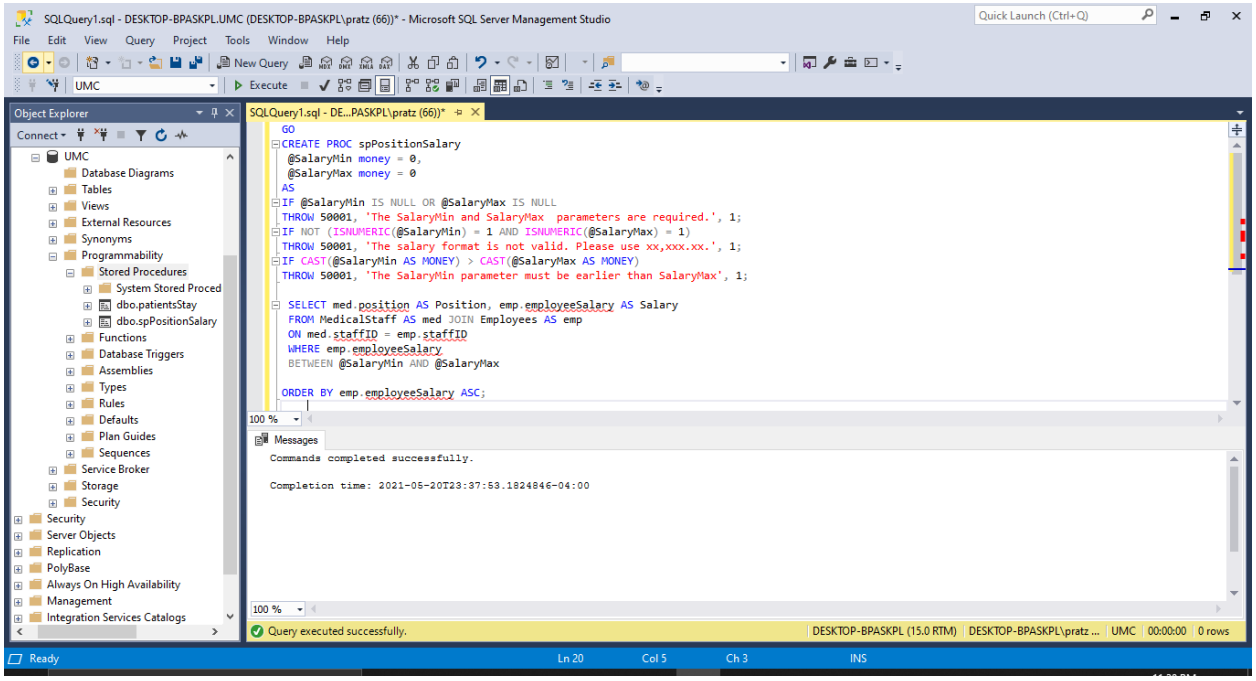


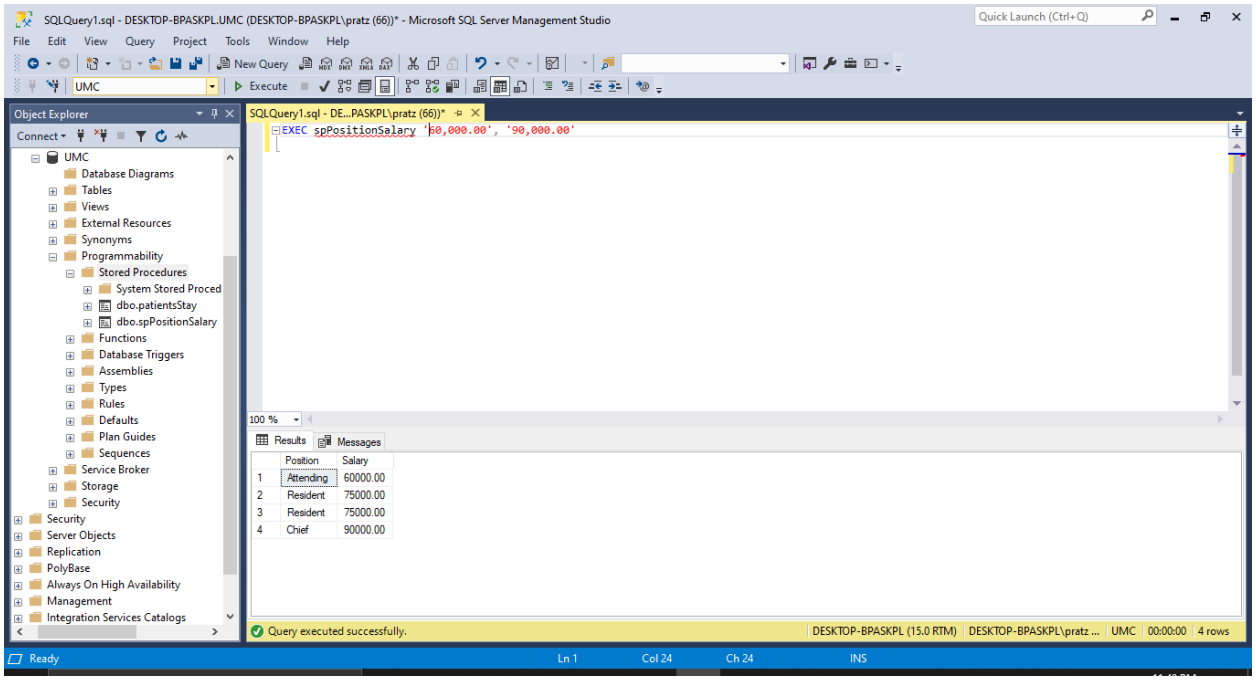
Second Stored Procedure

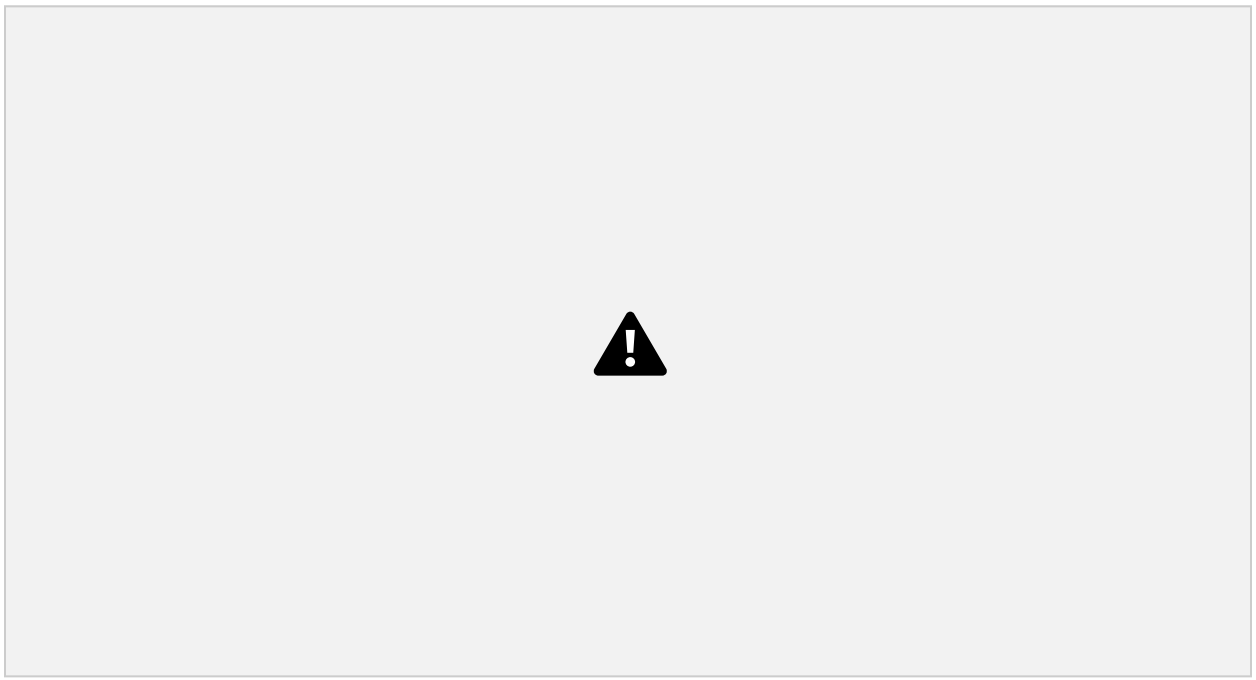
We will create a stored procedure named spPositionSalary that returns positions at the hospital within the given Salary range. This procedure accepts two parameters, @SalaryMin and @SalaryMax, with data type MONEY and default value 0. If called with no parameters or with null values, raise an error that describes the problem. If called with non-null values, validate the parameters. Test that the literal strings are valid salaries and test that @BalanceMin is earlier than @BalanceMax. If the parameters are valid, return a result set that includes the position and respective salary.

CODE:

```
GO
CREATE PROC spPositionSalary
 @SalaryMin money = 0,
 @SalaryMax money = 0
AS
IF @SalaryMin IS NULL OR @SalaryMax IS NULL
        THROW 50001, 'The SalaryMin and SalaryMax  parameters are required.', 1;
IF NOT (ISNUMERIC(@SalaryMin) = 1 AND ISNUMERIC(@SalaryMax) = 1)
          THROW 50001, 'The salary format is not valid. Please use xx,xxx.xx.', 1;
IF CAST(@SalaryMin AS MONEY) > CAST(@SalaryMax AS MONEY)
      THROW 50001, 'The SalaryMin parameter must be earlier than SalaryMax', 1;

 SELECT med.position AS Position, emp.employeeSalary AS Salary,
 FROM MedicalStaff AS med JOIN Employees AS emp
 ON med.staffID = emp.staffID
 WHERE emp.employeeSalary
 BETWEEN @SalaryMin AND @SalaryMax

ORDER BY emp.employeeSalary ASC;
END;
```







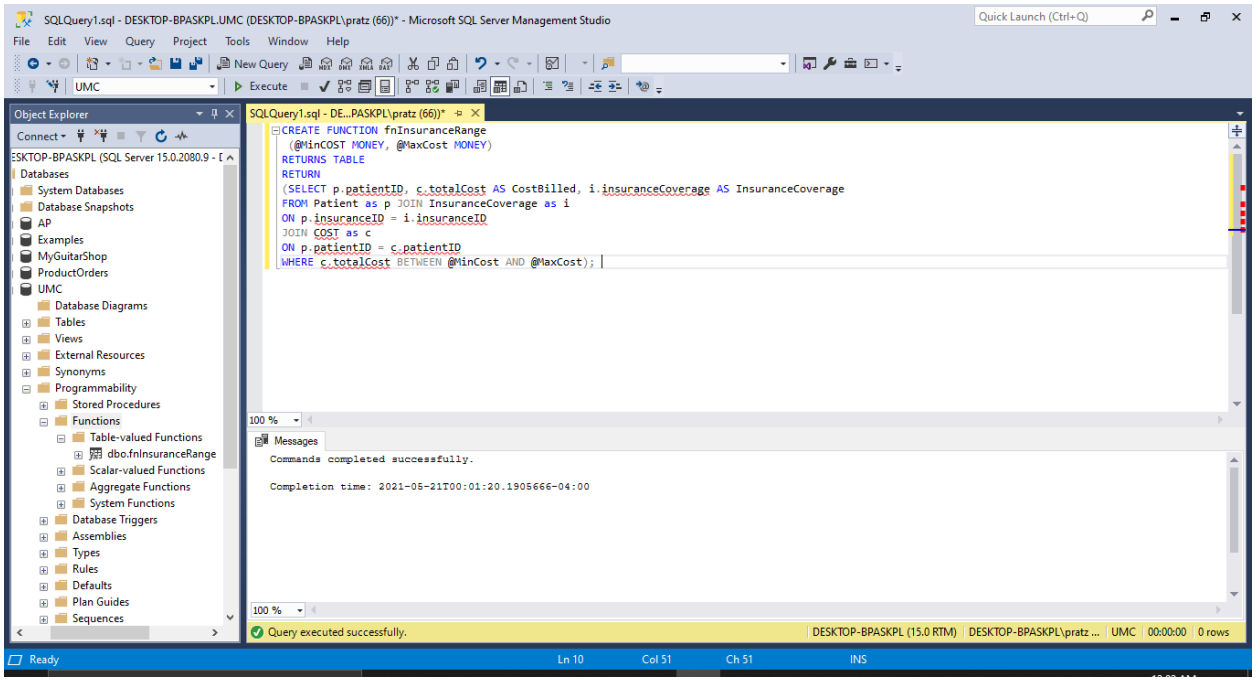
First User-defined Function

We are gonna create a table-valued function that returns the cost billed to a patient and the insurance coverage they are provided. This function takes in two parameters @MinCost and @MaxCost and returns the rows with cost billed to the patient between minimum and maximum costs defined by the user.

CODE:

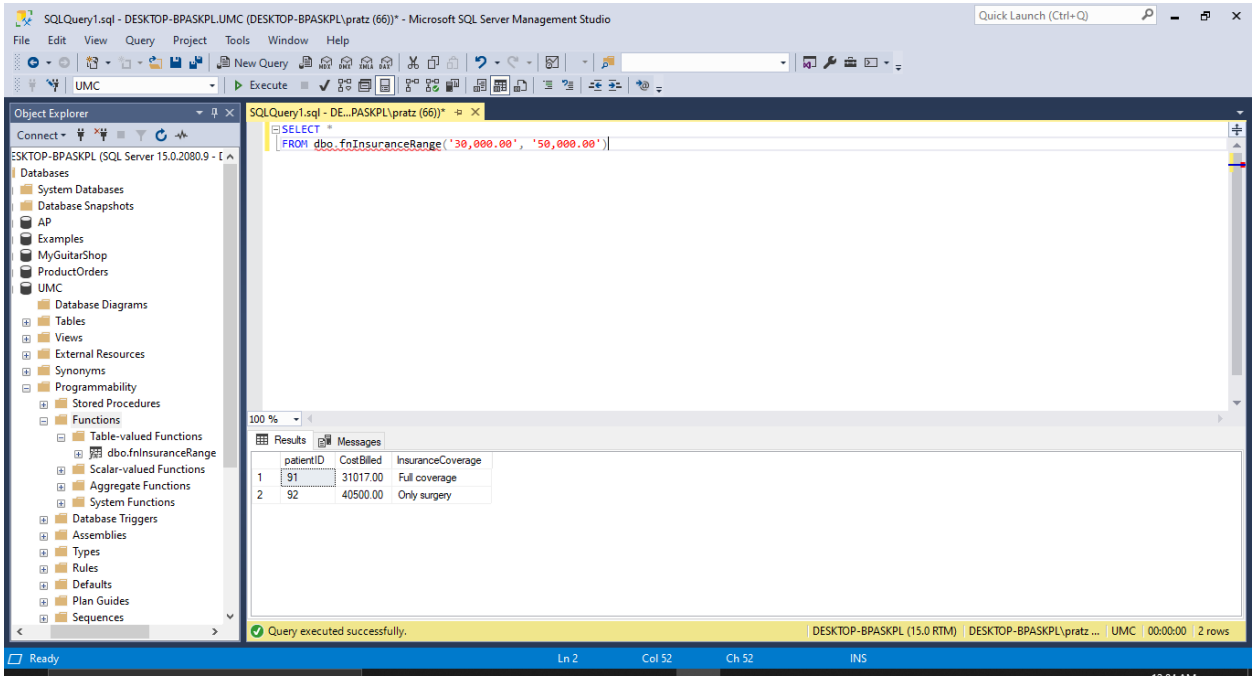
```
CREATE FUNCTION fnInsuranceRange
 (@MinCOST MONEY, @MaxCost MONEY)
RETURNS TABLE
```

```
RETURN
(SELECT p.patientID, c.totalCost AS CostBilled, i.insuranceCoverage AS
InsuranceCoverage
 FROM Patient as p JOIN InsuranceCoverage as i
 ON p.insuranceID = i.insuranceID
 JOIN COST as c
 ON p.patientID = c.patientID
 WHERE c.totalCost BETWEEN @MinCost AND @MaxCost);
```



RESULT

```
SELECT *
FROM dbo.fnInsuranceRange('30,000.00','50,000.00');
```

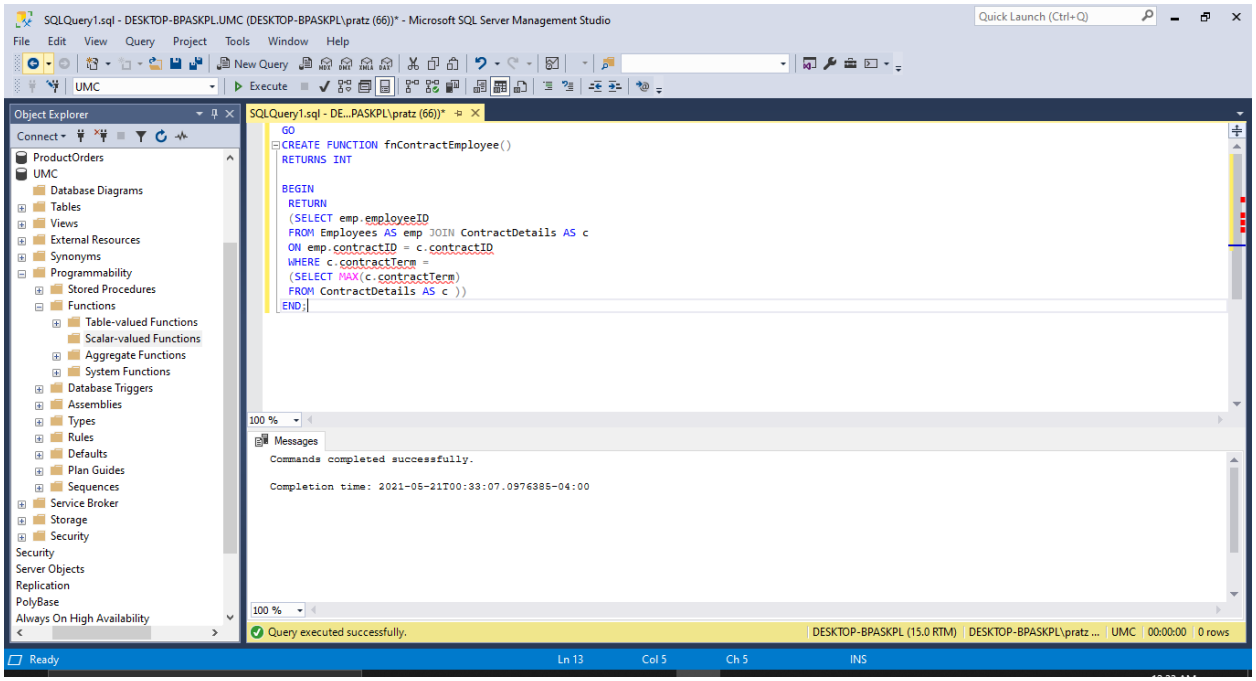


Second User-Defined Function

Next we will create a scalar valued function that returns the employee with the longest contract term. This function returns a single row.

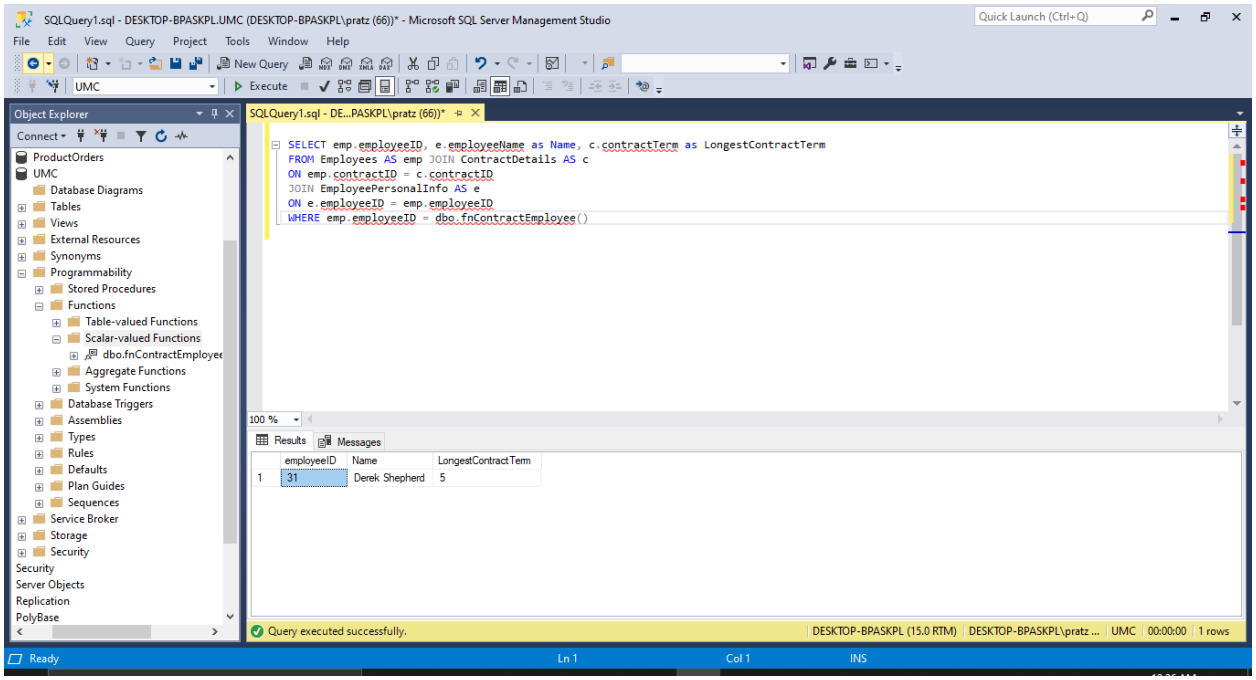
CODE:

```
GO
CREATE FUNCTION fnContractEmployee()
RETURNS INT

BEGIN
 RETURN
 (SELECT emp.employeeID
 FROM Employees AS emp JOIN ContractDetails AS c
 ON emp.contractID = c.contractID
 WHERE c.contractTerm =
 (SELECT MAX(c.contractTerm)
 FROM ContractDetails AS c ))
END;
```



RESULT

```
SELECT emp.employeeID, e.employeeName as Name, c.contractTerm as LongestContractTerm
FROM Employees AS emp JOIN ContractDetails AS c
ON emp.contractID = c.contractID
JOIN EmployeePersonalInfo AS e
ON e.employeeID = emp.employeeID
WHERE emp.employeeID = dbo.fnContractEmployee()
```

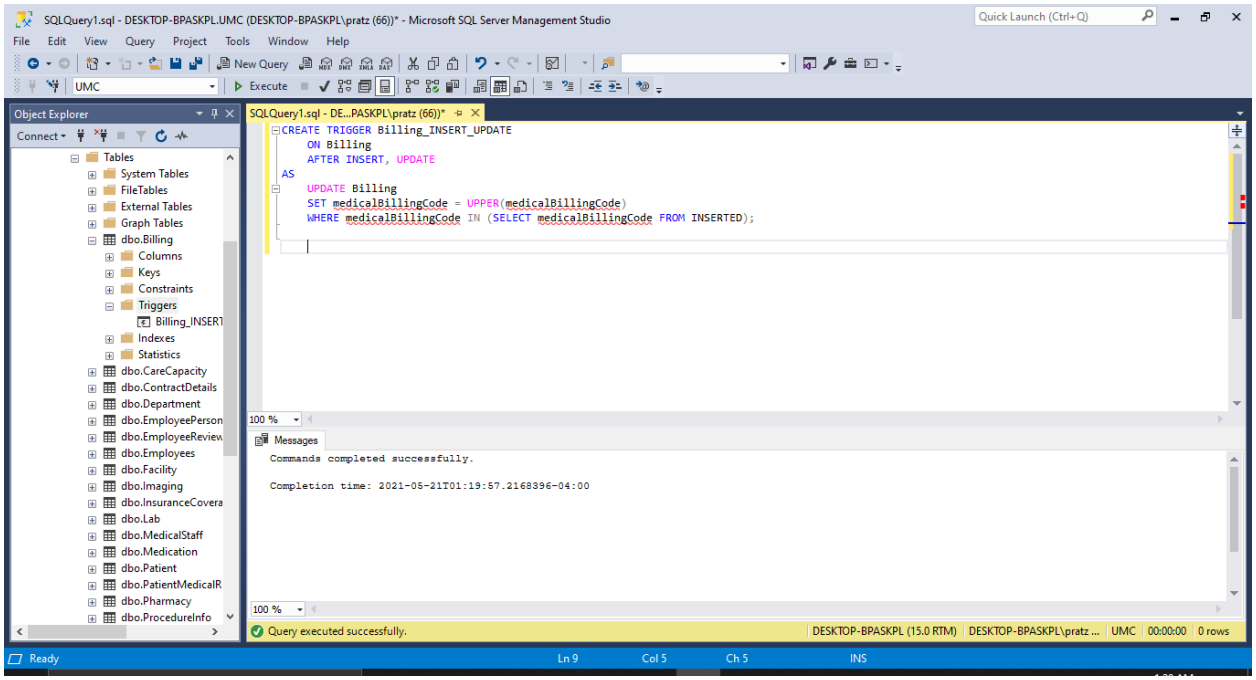


Trigger 1

We are creating a trigger that corrects the procedure code to upper case if the value has been inserted in mixed case form. The procedure codes need to always be in uppercase. In the result it is shown that the medical billing code is in uppercase.

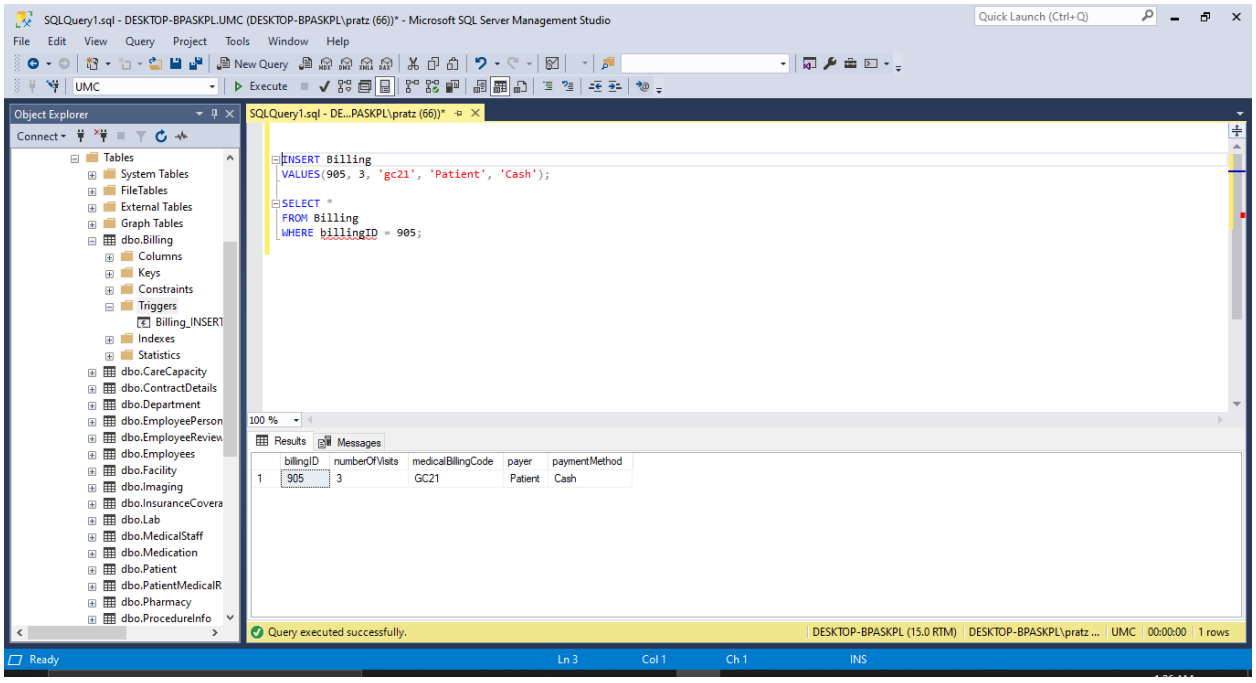
CODE:

```
CREATE TRIGGER Billing_INSERT_UPDATE
        ON Billing
        AFTER INSERT, UPDATE
AS
        UPDATE Billing
        SET medicalBillingCode = UPPER(medicalBillingCode)
        WHERE medicalBillingCode IN (SELECT medicalBillingCode FROM INSERTED);
```



--An insert statement that fires the trigger.

INSERT Billing
VALUES(905, 3, ‘gc21’, ‘Patient’, ‘Cash’);



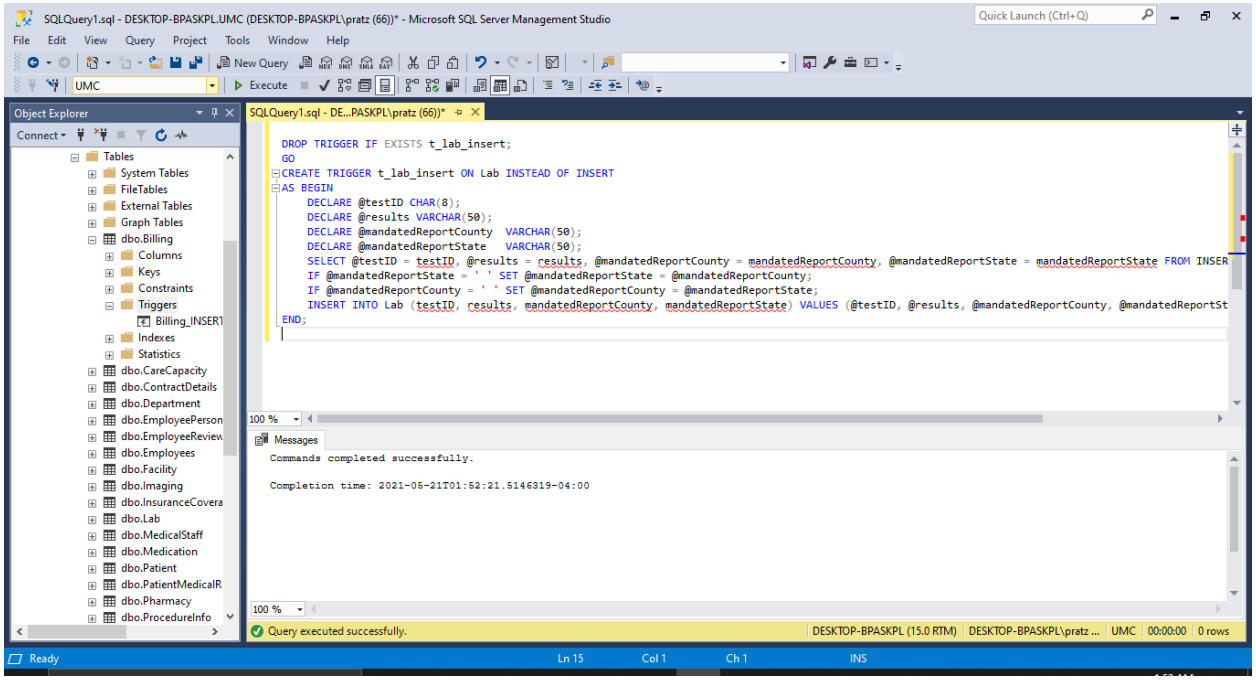
Trigger 2

We have created a trigger t_lab_insert. IF an empty value is inserted in mandatedReportCounty or mandatedReportState, the value of the other column is inserted instead of the empty entry. As we can see in the example, mandatedReportCounty had an empty value, so the value of

mandatedReportState, ‘NY’ was inserted instead. The reason for this trigger is so that we can deal with empty values and the report can go to what is returned. As we can see the trigger successfully performs the required.

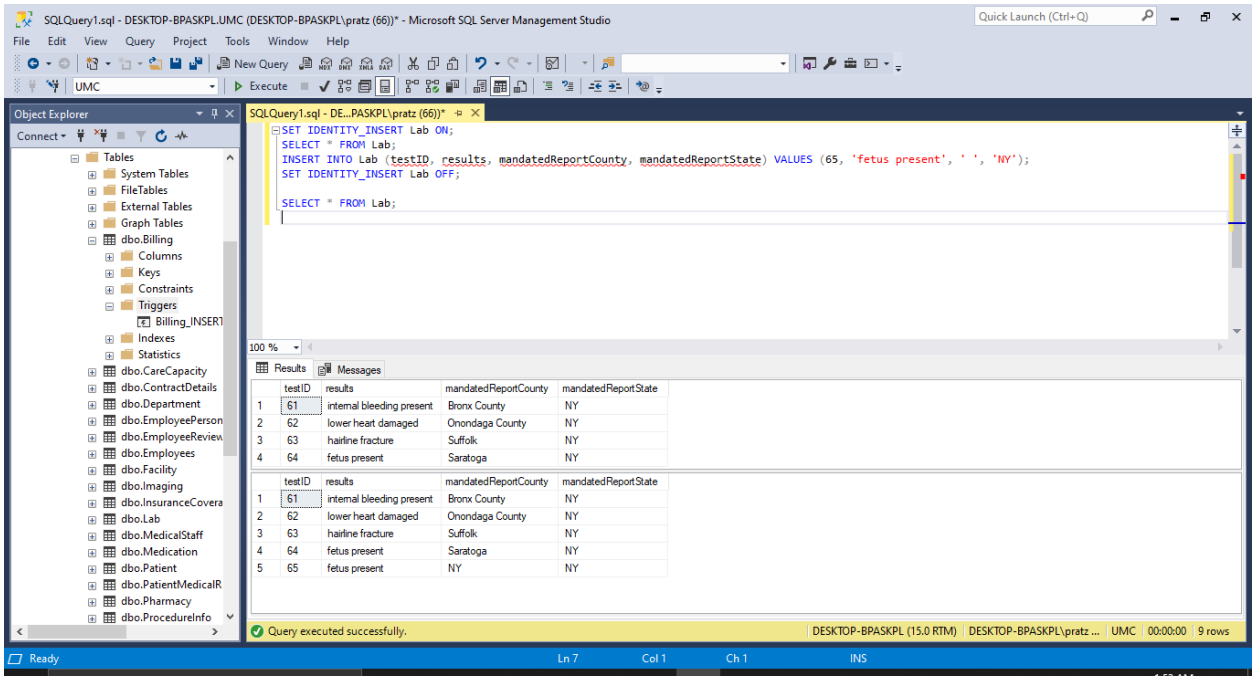
CODE:

```
DROP TRIGGER IF EXISTS t_lab_insert;
GO
CREATE TRIGGER t_lab_insert ON Lab INSTEAD OF INSERT
AS BEGIN
    DECLARE @testID CHAR(8);
    DECLARE @results VARCHAR(50);
    DECLARE @mandatedReportCounty  VARCHAR(50);
    DECLARE @mandatedReportState   VARCHAR(50);
    SELECT @testID = testID, @results = results, @mandatedReportCounty =
mandatedReportCounty, @mandatedReportState = mandatedReportState FROM INSERTED;
    IF @mandatedReportState = ' ' SET @mandatedReportState = @mandatedReportCounty;
    IF @mandatedReportCounty = ' ' SET @mandatedReportCounty = @mandatedReportState;
    INSERT INTO Lab (testID, results, mandatedReportCounty, mandatedReportState)
VALUES (@testID, @results, @mandatedReportCounty, @mandatedReportState);
END;
```



SET IDENTITY_INSERT Lab ON;
SELECT * FROM Lab;
INSERT INTO Lab (testID, results, mandatedReportCounty, mandatedReportState) VALUES (65, 'fetus present', ' ', 'NY');
SET IDENTITY_INSERT Lab OFF;

RESULT
SELECT * FROM Lab;

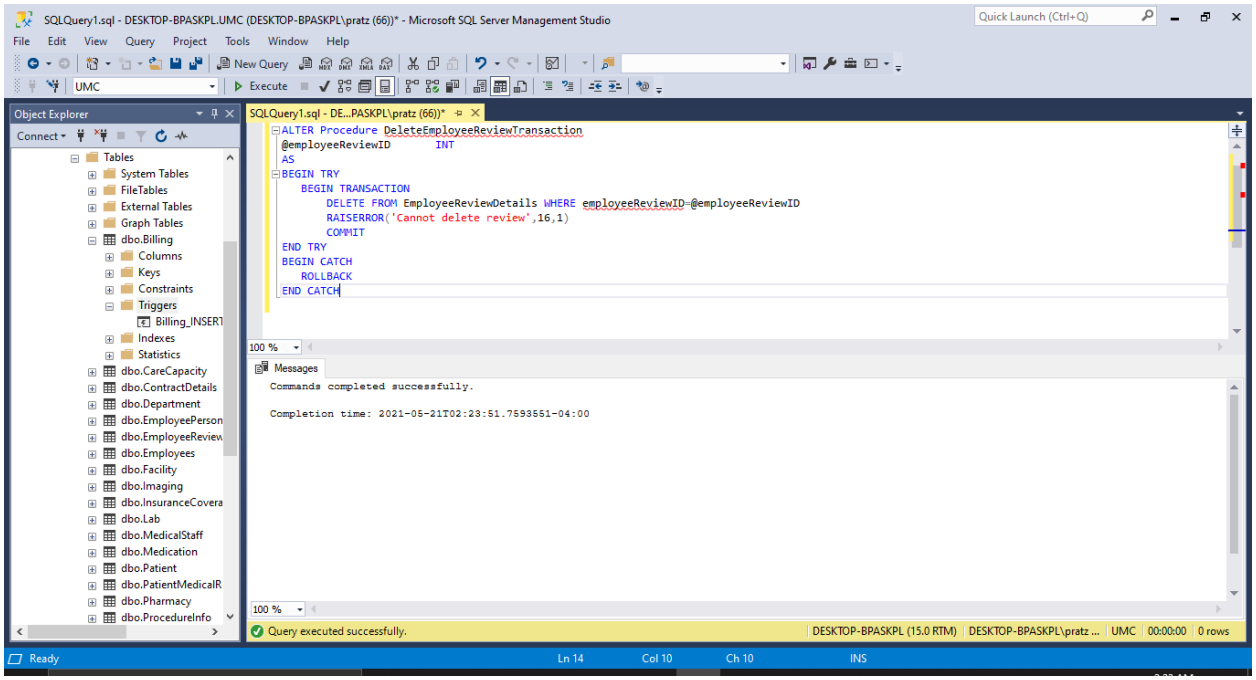


Transaction

I created a transaction that does not allow a user to delete posted employee reviews. This is to ensure that there are honest reviews of the staff at UMC for new patients to make a more informed decision about treatment. If I get any error in the TRY block then control moves to the CATCH block. In the CATCH block I am doing the ROLLBACK, which means make all the statements after the Begin Transaction (here the delete one) becomes void and returns back to the previous state.

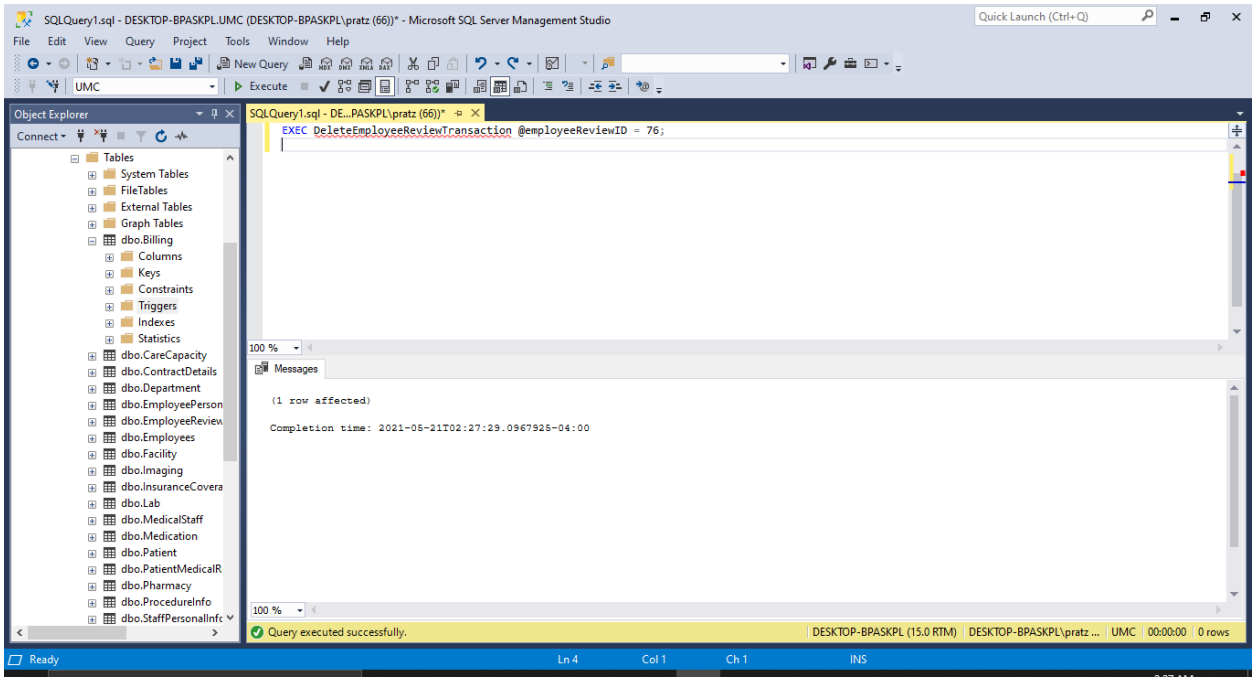
CODE:

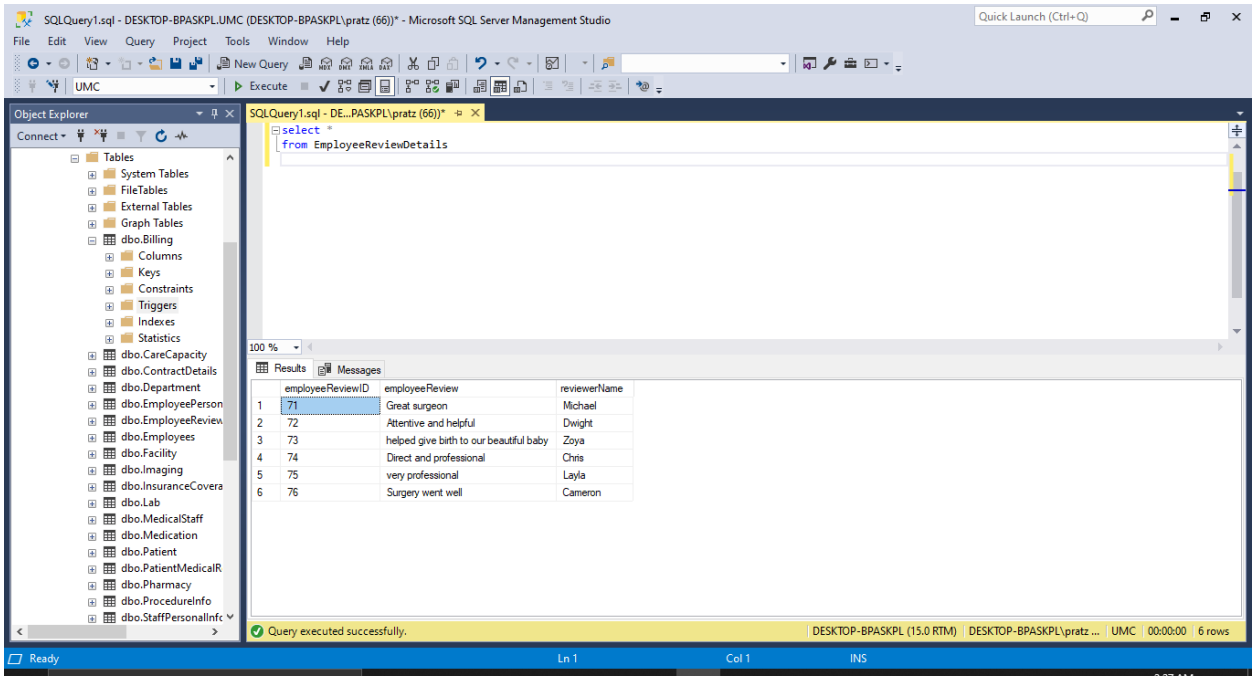
```
ALTER Procedure DeleteEmployeeReviewTransaction
@employeeReviewID      INT
AS
BEGIN TRY
  BEGIN TRANSACTION
     DELETE FROM EmployeeReviewDetails WHERE employeeReviewID=@employeeReviewID
     RAISERROR('Cannot delete review',16,1)
     COMMIT
END TRY
BEGIN CATCH
  ROLLBACK
END CATCH
```



RESULT

EXEC DeleteEmployeeReviewTransaction @employeeReviewID = 76





As we can see, the row with employeeReviewID = 76 was not deleted.

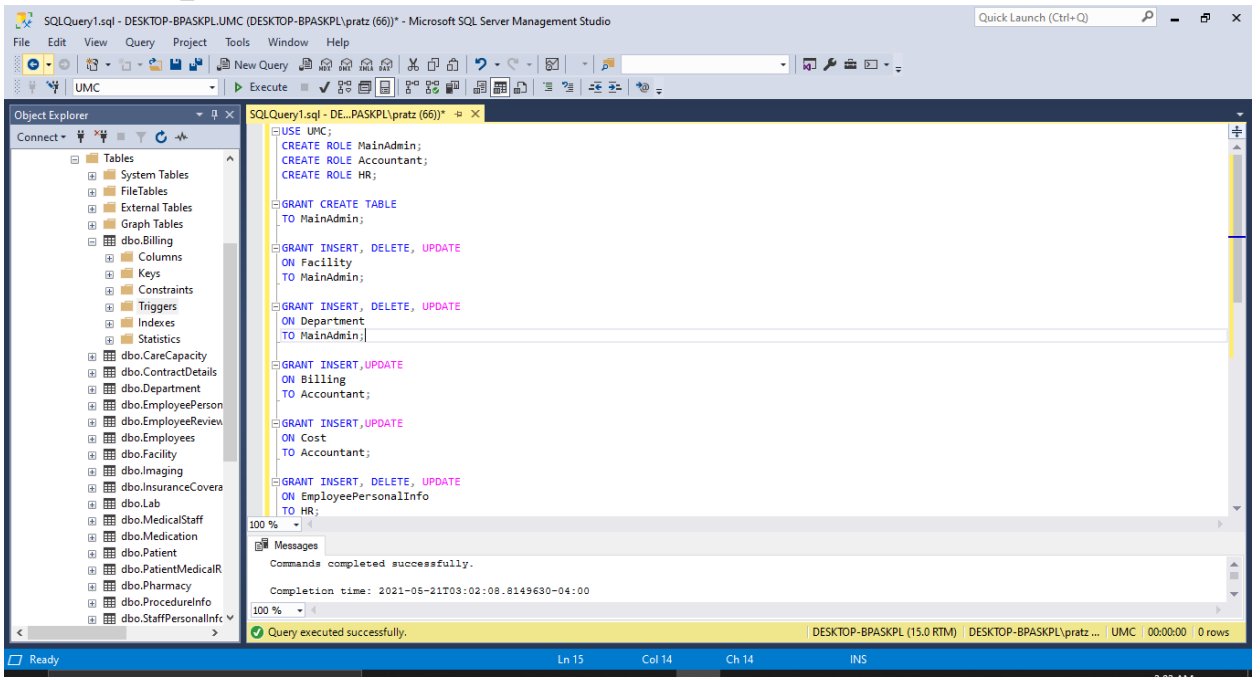
First Script for Security

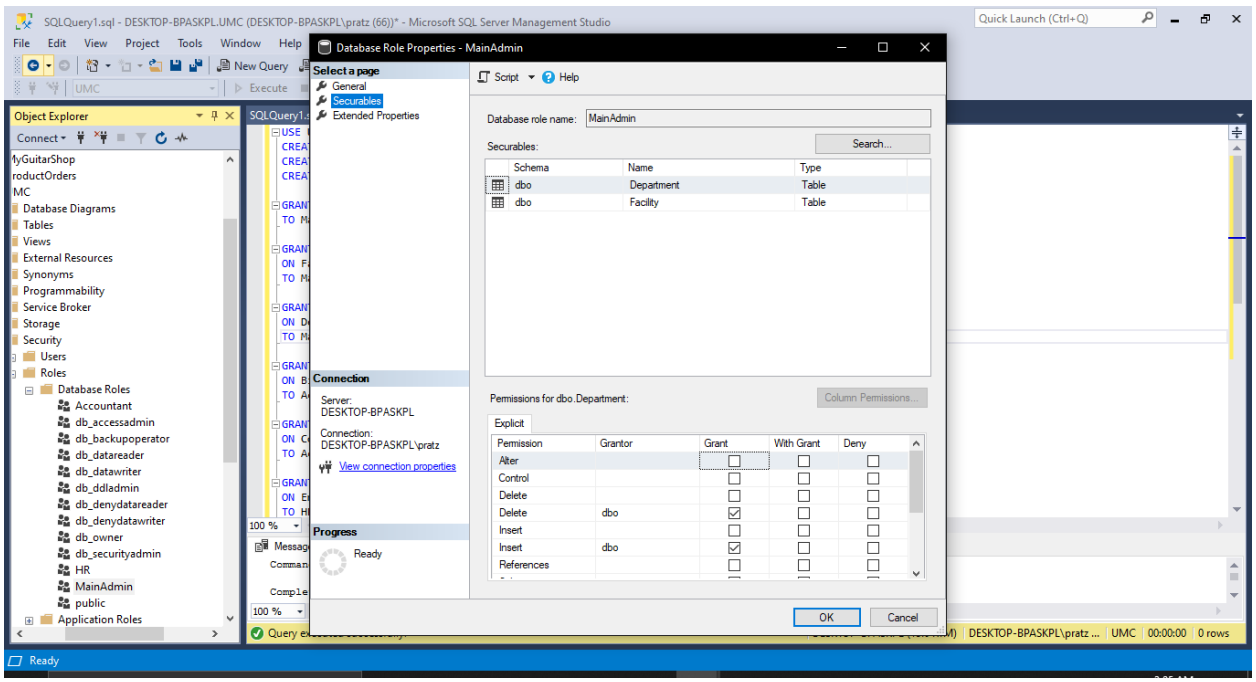
This script creates three user-defined database roles named MainAdmin, Accountant, HR in the UMC database. Each role is granted various permissions in respect to their security levels to access the tables.

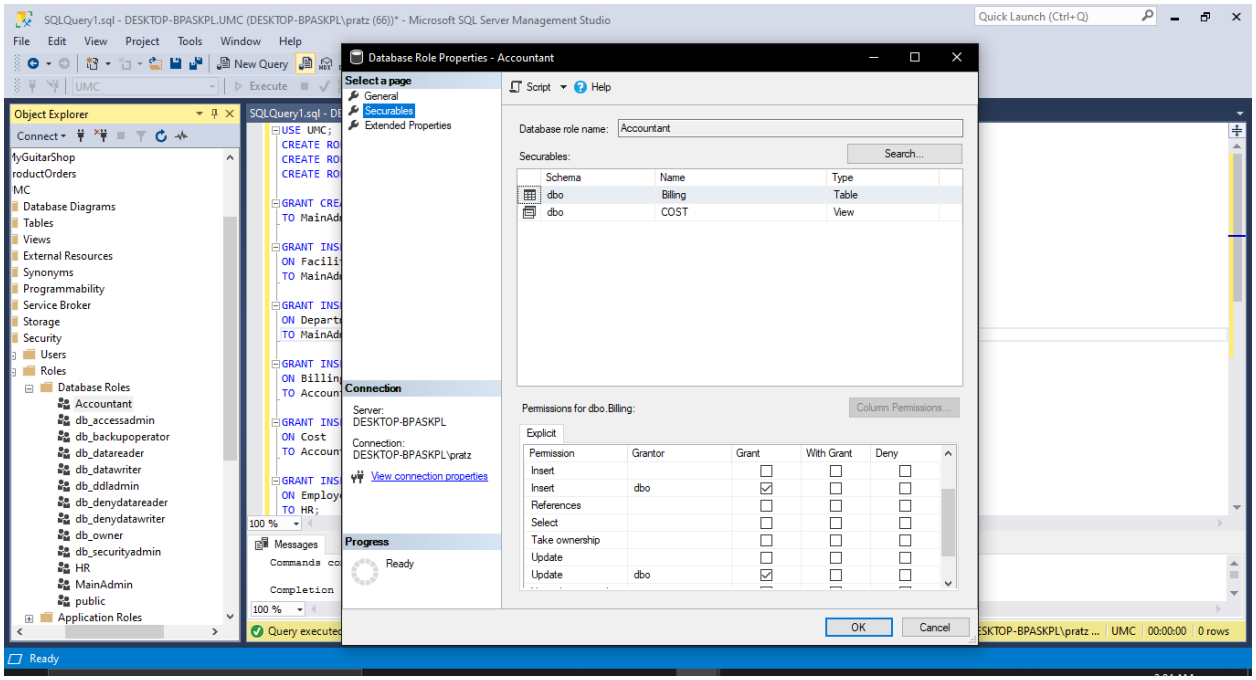
CODE:

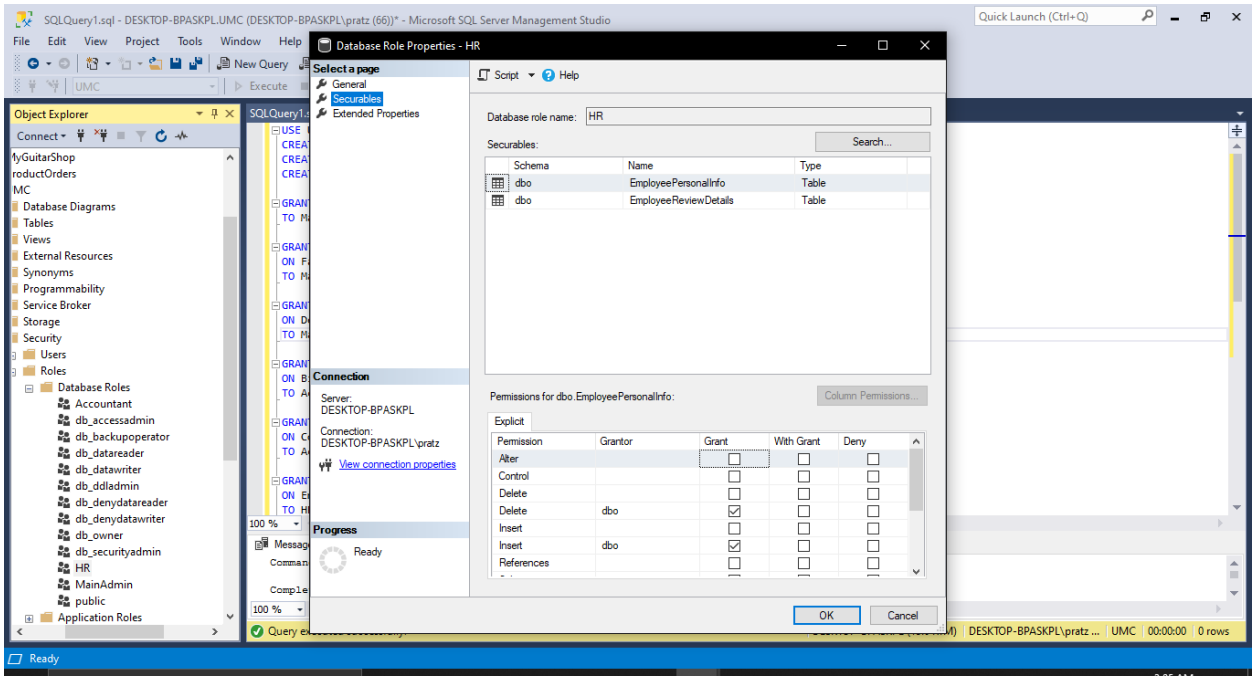
```
USE UMC;
CREATE ROLE MainAdmin;
CREATE ROLE Accountant;
CREATE ROLE HR;

GRANT CREATE TABLE
TO MainAdmin;

GRANT INSERT, DELETE, UPDATE
ON Facility
TO MainAdmin;

GRANT INSERT, DELETE, UPDATE
ON Department
TO MainAdmin;

GRANT INSERT,UPDATE
ON Billing
TO Accountant;

GRANT INSERT,UPDATE
ON Cost
TO Accountant;

GRANT INSERT, DELETE, UPDATE
```

```
ON EmployeePersonalInfo
TO HR;

GRANT INSERT, DELETE, UPDATE
ON EmployeeReviewDetails
TO HR;

ALTER ROLE db_datareader ADD MEMBER MainAdmin;
ALTER ROLE db_datareader ADD MEMBER Accountant;
ALTER ROLE db_datareader ADD MEMBER HR;
```









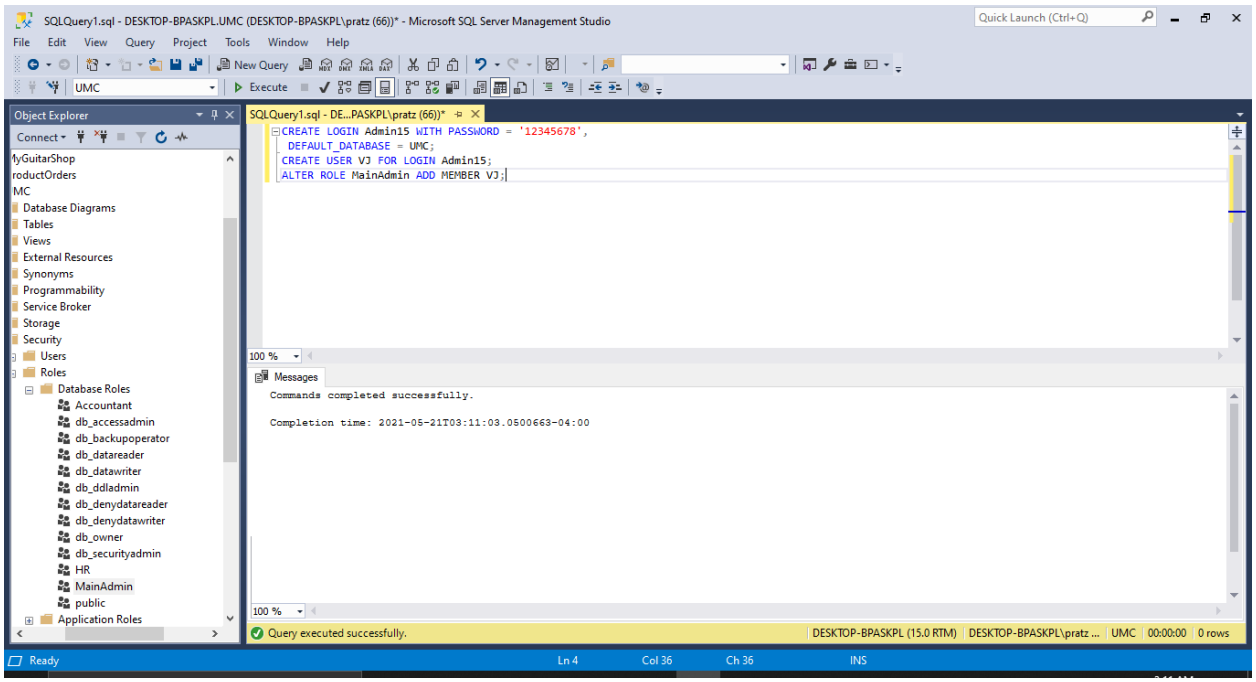
Second Script for Security

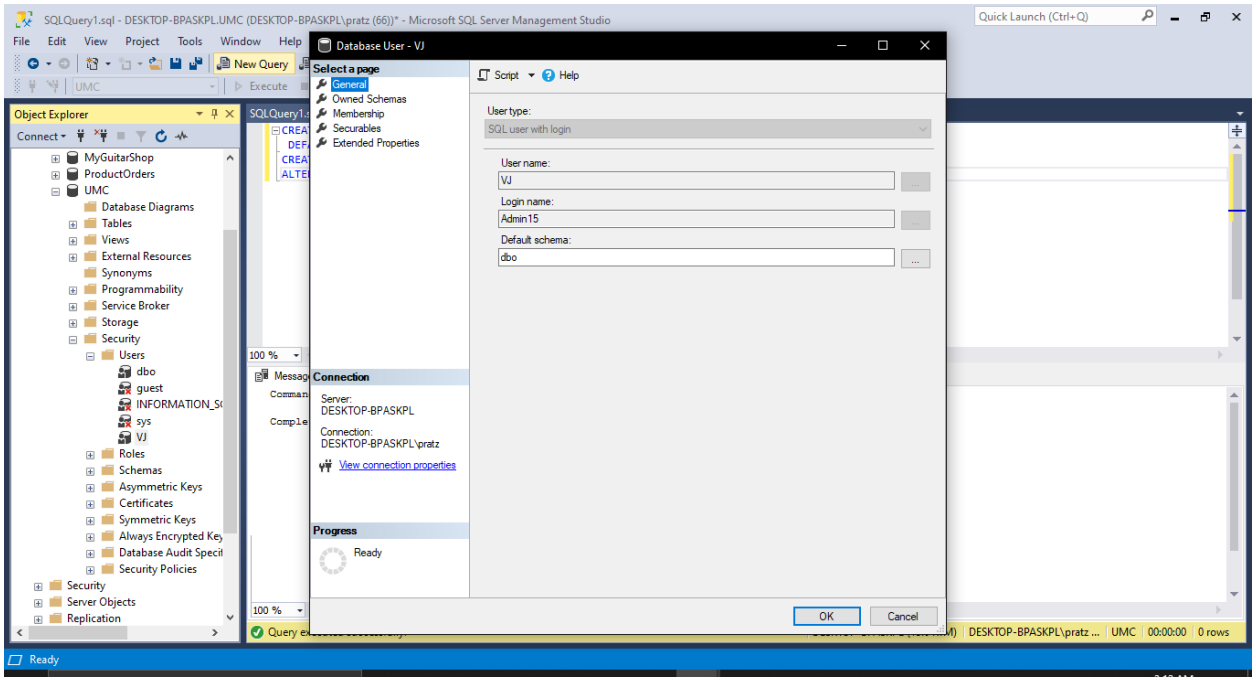
This script (1) creates a login ID named “Admin15” with the password “12345678”; (2) sets the default database for the login to the UMMC database; (3) creates a user named “VJ” for the login; and (4) assigns the user to the MainAdmin role to be granted the permissions given in script 1.

CODE:

```
CREATE LOGIN Admin15 WITH PASSWORD = '12345678',
```

```
 DEFAULT_DATABASE = UMC;
CREATE USER VJ FOR LOGIN Admin15;
ALTER ROLE MainAdmin ADD MEMBER VJ;
```





CONCLUSIONS/ REMARKS

- In this Project, I learned how to design, implement and test a database for a University Medical Center.

- The project was mainly divided into 3 phases which included designing of the database using E/R diagram and drawing conclusions from the same
- In the implementation phase, determination of tables, columns, primary keys, datatypes, nullabilities and relationships was achieved.
- In the testing phase, several business logics and performance and efficiency improvements were applied.