UNIVERSITY MEDICAL CENTER DATABASE

Abstract

In today's fast-paced healthcare environment, efficient patient management and seamless coordination among healthcare providers are vital for delivering optimal care. This project aims to design and implement a comprehensive database system for the Patient Access and Registration branch of the University Medical Centre, focusing on tracking patients from admission to discharge. The database will facilitate care coordination, improve patient care, reduce errors, and optimize resource utilization by capturing essential patient information and streamlining hospital operations.

Shaunak Deo

Table of Content

A. Database Design:

The University Medical Centre database is designed to manage various aspects of a hospital, including patient records, employee information, and department details. It keeps track of important data such as patient admissions, medical procedures, medications, and appointments. Additionally, the database records insurance information, billing details, and laboratory results. Furthermore, it helps manage hospital resources such as equipment, donations, and feedback. Overall, the database provides an organized and comprehensive system for managing a hospital's daily operations and patient care.

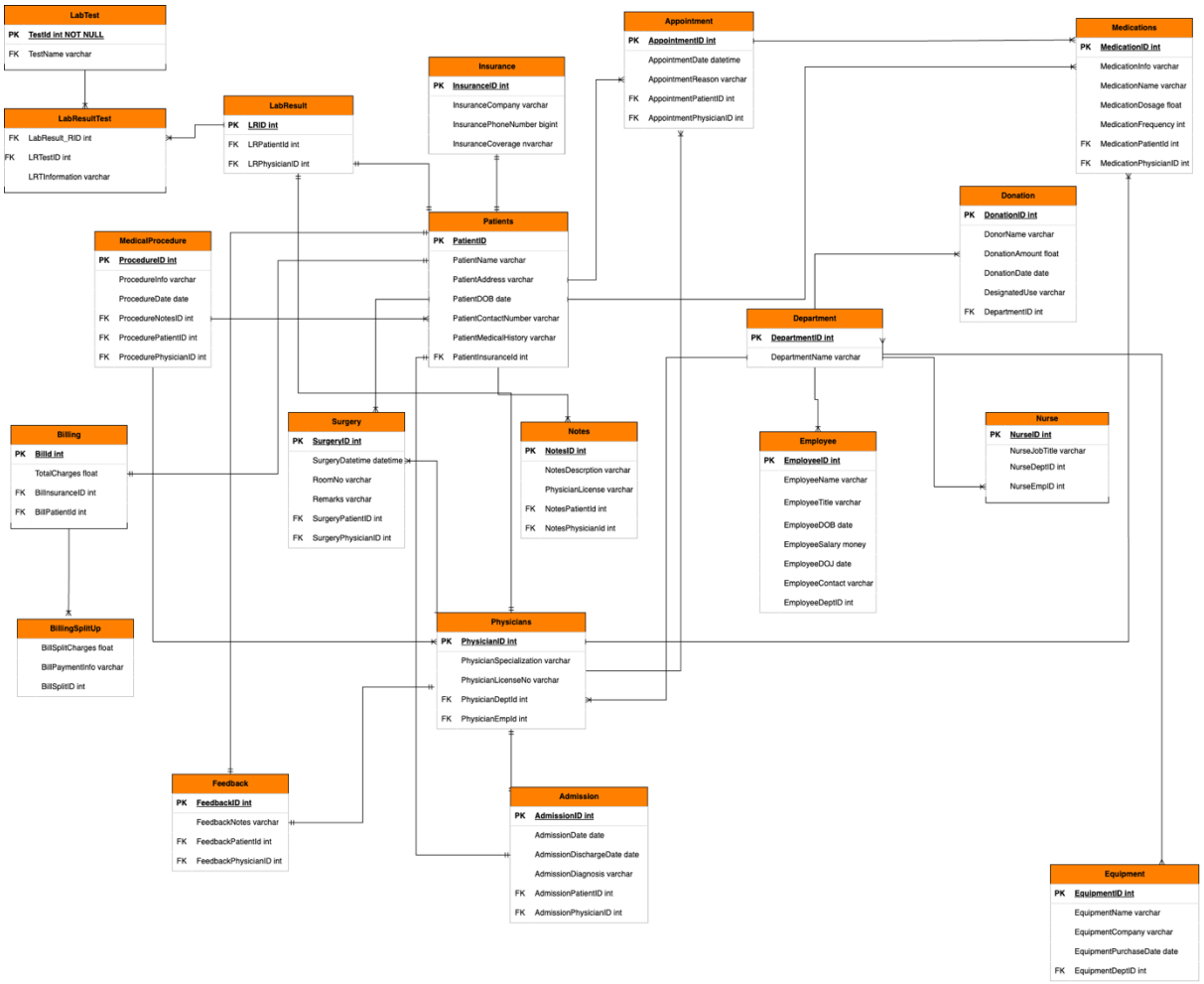
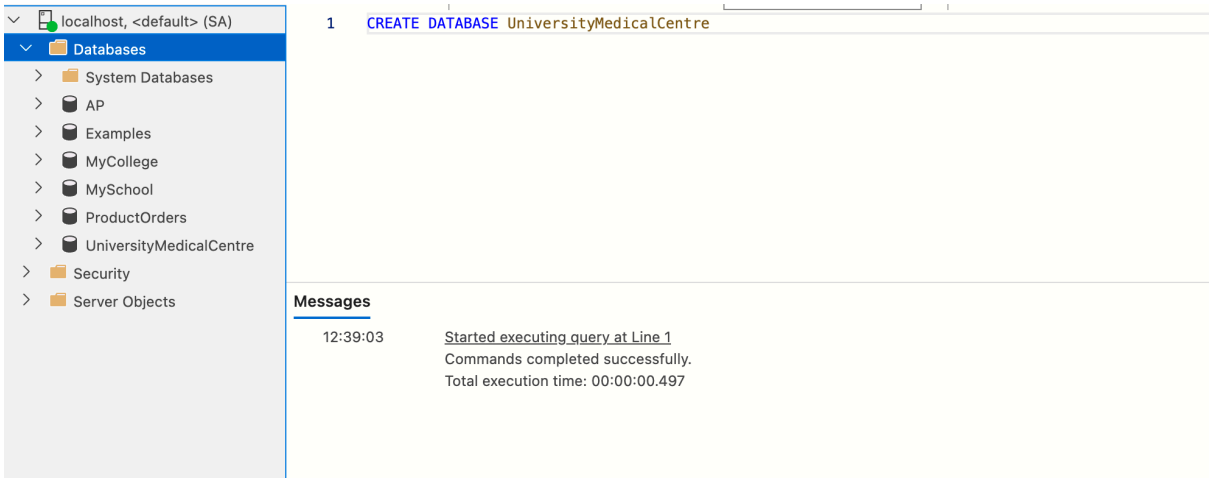


Table Name	Description
Insurance	Contains information about insurance companies, including their name, phone number, and coverage type.
Patient	Stores patient information, including name, address, date of birth, contact number, medical history, and insurance details.
Department	Contains information about hospital departments, such as their name and ID.
Employee	Holds information about hospital employees, including name, title, date of birth, salary, date of joining, contact number, and department ID.
Physician	Contains information about physicians, including their specialization, license number, department ID, and employee ID.
Notes	Stores notes created by physicians for patients, including the note description, patient ID, and physician ID.
Nurse	Contains information about nurses, including their job title, department ID, and employee ID.
Admission	Stores information about patient admissions, including admission date, discharge date, diagnosis, patient ID, and physician ID.
MedicalProcedure	Contains information about medical procedures performed on patients, including procedure info, date, notes ID, patient ID, and physician ID.
Medications	Stores information about medications prescribed to patients, including medication info, name, dosage, frequency, patient ID, and physician ID.
Appointment	Contains information about appointments, including the appointment date, reason, patient ID, and physician ID.
Billing	Stores information about billing, including total charges, insurance ID, and patient ID.
BillingSplitUp	Contains information about the breakdown of charges within a bill, including the charges and payment info.
LabResult	Contains information about lab results, including patient ID and physician ID.
LabTest	Stores information about lab tests, including the test name.
LabResultTest	Contains information about lab result tests, including lab result ID, test ID, and test information.
Donation	Stores information about donations made to the hospital, including the donor name, donation amount, donation date, designated use, and department ID.
Equipment	Contains information about hospital equipment, including the equipment name, company, purchase date, and department ID.
Feedback	Stores feedback from patients, including the feedback notes, patient ID, and physician ID.
Surgery	Contains information about surgeries performed, including surgery datetime, room number, remarks, patient ID, and physician ID.

B. Implementation and Testing

Create Database:

```
CREATE DATABASE UniversityMedicalCentre
```



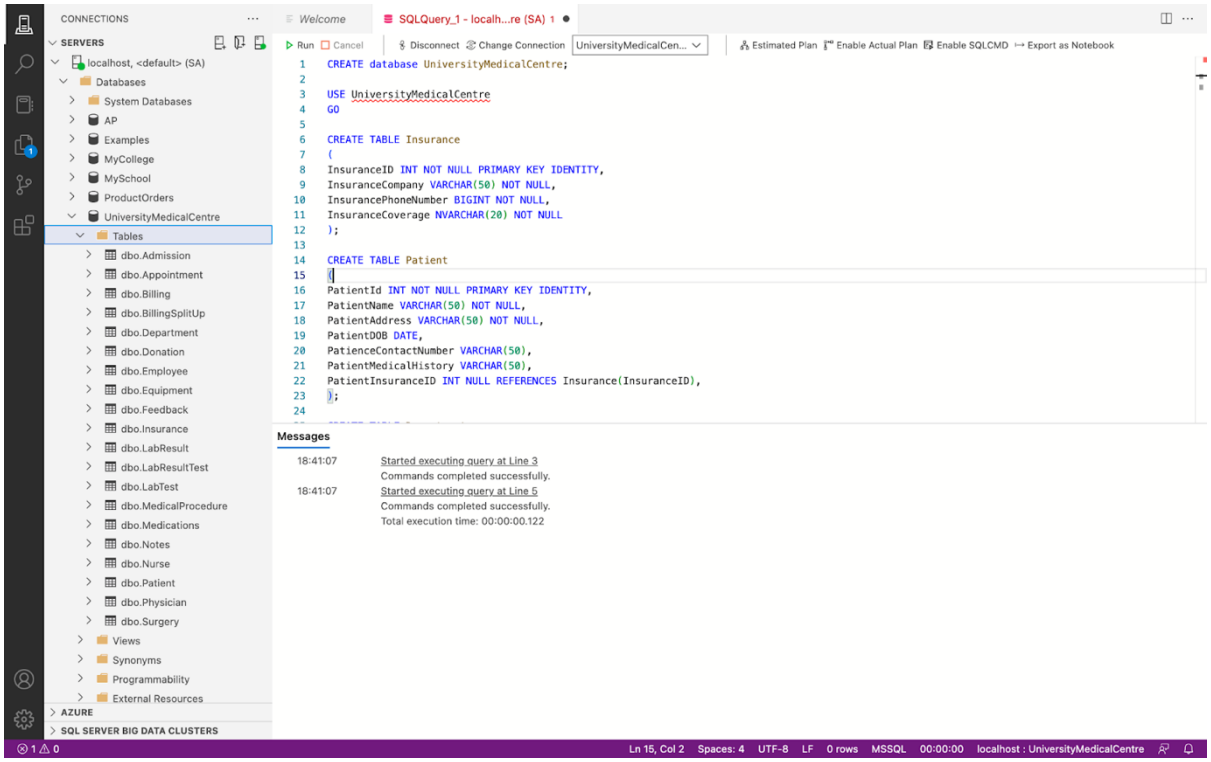
Create Tables:

```
CREATE TABLE Insurance
(
InsuranceID INT NOT NULL PRIMARY KEY IDENTITY,
InsuranceCompany VARCHAR(50) NOT NULL,
InsurancePhoneNumber BIGINT NOT NULL,
InsuranceCoverage NVARCHAR(20) NOT NULL
);

CREATE TABLE Patient
(
PatientId INT NOT NULL PRIMARY KEY IDENTITY,
PatientName VARCHAR(50) NOT NULL,
PatientAddress VARCHAR(50) NOT NULL,
PatientDOB DATE,
PatienceContactNumber VARCHAR(50),
PatientMedicalHistory VARCHAR(50),
PatientInsuranceID INT NULL REFERENCES Insurance(InsuranceID),
);

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

```
-- Employees Table
CREATE TABLE Employee
(
EmployeeID INT NOT NULL PRIMARY KEY IDENTITY,
EmployeeName VARCHAR(50) NOT NULL,
EmployeeTitle VARCHAR(50) NOT NULL,
EmployeeDOB DATE,
EmployeeSalary MONEY NOT NULL,
EmployeeDOJ DATE,
EmployeeContact VARCHAR(50) NOT NULL,
EmployeeDeptId INT NOT NULL REFERENCES Department(DepartmentID),
);

CREATE TABLE Physician
(
PhysicianID INT NOT NULL PRIMARY KEY IDENTITY,
PhysicianSpecialization VARCHAR(50),
PhysicanLicenseNo VARCHAR(50),
PhysicianDeptId INT NOT NULL REFERENCES Department(DepartmentID),
PhysicianEmpId INT NOT NULL REFERENCES Employee(EmployeeID)
);

CREATE TABLE Notes
(
NotesID INT NOT NULL PRIMARY KEY IDENTITY,
NotesDescription VARCHAR(100),
NotesPatientId INT NOT NULL REFERENCES Patient(PatientId),
NotesPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

CREATE TABLE Nurse
(
NurseID INT NOT NULL PRIMARY KEY IDENTITY,
NurseJobTitle VARCHAR(60),
NurseDeptId INT NOT NULL REFERENCES Department(DepartmentID),
NurseEmpId INT NOT NULL REFERENCES Employee(EmployeeID)
);

CREATE TABLE Admission
(
AdmissionID INT NOT NULL PRIMARY KEY IDENTITY,
AdmissionDate DATE,
AdmissionDischargeDate DATE,
AdmissionDiagnosis VARCHAR(60),
AdmissionPatientId INT NOT NULL REFERENCES Patient(PatientId),
AdmissionPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

CREATE TABLE MedicalProcedure
(
ProcedureID INT NOT NULL PRIMARY KEY IDENTITY,
```

```
ProcedureInfo VARCHAR(60),
ProcedureDate DATE,
ProcedureNotesID INT NOT NULL REFERENCES Notes(NotesID),
ProcedurePatientId INT NOT NULL REFERENCES Patient(PatientId),
ProcedurePhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

CREATE TABLE Medications
(
MedicationID INT NOT NULL PRIMARY KEY IDENTITY,
MedicationInfo VARCHAR(60),
MedicationName VARCHAR(60),
MedicationDosage FLOAT,
MedicationFrequency INT,
MedicationPatientId INT NOT NULL REFERENCES Patient(PatientId),
MedicationPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

CREATE TABLE Appointment
(
AppointmentID INT NOT NULL PRIMARY KEY IDENTITY,
AppointmentDate DATETIME,
AppointmentReason VARCHAR(60),
AppointmentPatientId INT NOT NULL REFERENCES Patient(PatientId),
AppointmentPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

Create table Billing
(
BillID INT NOT NULL PRIMARY KEY IDENTITY,
TotalCharges FLOAT,
BillInsuranceID INT NOT NULL REFERENCES Insurance(InsuranceID),
BillPatientId INT NOT NULL REFERENCES Patient(PatientId)
);

Create table BillingSplitUp
(
BillSplitCharges FLOAT,
BillPaymentInfo VARCHAR(60),
BillSplitID INT NOT NULL REFERENCES Billing(BillID)
);

Create table LabResult
(
LRID INT NOT NULL PRIMARY KEY IDENTITY,
LRPatientId INT NOT NULL REFERENCES Patient(PatientId),
LRPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

Create table LabTest
(
```

```
TestID INT NOT NULL PRIMARY KEY IDENTITY,
TestName VARCHAR(60)
);

Create table LabResultTest
(
LabResultLRID INT NOT NULL REFERENCES LabResult(LRID),
LRTestID INT NOT NULL REFERENCES LabTest(TestID),
LRTInformation VARCHAR(60)
);

CREATE TABLE Donation
(
DonationID INT NOT NULL PRIMARY KEY IDENTITY,
DonorName VARCHAR(60),
DonationAmount FLOAT NOT NULL,
DonationDate DATE,
DesignatedUse VARCHAR(50),
DepartmentID INT NOT NULL REFERENCES Department(DepartmentID)
);


CREATE TABLE Equipment
(
EquipmentID INT NOT NULL PRIMARY KEY IDENTITY,
EquipmentName VARCHAR(60) NOT NULL,
EquipmentCompany VARCHAR(60) NOT NULL,
EquipmentPurchaseDate DATE NOT NULL,
EquipmentDeptID INT NOT NULL REFERENCES Department(DepartmentID)
);

CREATE TABLE Feedback
(
FeedbackID INT NOT NULL PRIMARY KEY IDENTITY,
FeedbackNotes VARCHAR(60) NOT NULL,
FeedbackPatientId INT NOT NULL REFERENCES Patient(PatientId),
FeedbackPhysicianID INT NOT NULL REFERENCES Physician(PhysicianID)
);

CREATE TABLE Surgery(
SurgeryID int NOT NULL PRIMARY KEY IDENTITY,
SurgeryDatetime datetime,
RoomNo varchar(5),
Remarks varchar(200),
SurgeryPatientId int NOT NULL REFERENCES Patient(PatientId),
SurgeryPhysicianID int NOT NULL REFERENCES Physician(PhysicianID)
);
```



The tables created for the University Medical Centre database cover various aspects of the medical center's operations, such as patient information, employee details, appointments, medical procedures, and billing. The choices made in the table designs emphasize data integrity, normalization, and maintainability.

Each table is designed with a primary key to uniquely identify each record and ensure data integrity. Foreign key relationships are established where necessary, to enforce referential integrity between related tables, ensuring that data is consistent across the database.

The database schema follows a normalized design, reducing redundancy and improving efficiency. For example, the Employee table stores common information about all employees, with the Physician and Nurse tables containing additional information specific to those roles.

The table structure is designed to accommodate various use cases, such as storing patient medical history, tracking appointments, managing billing, and recording feedback. Additionally, the database includes tables for lab results, donations, and equipment to support a comprehensive view of the medical center's operations.

Overall, the choices made in the design of these tables aim to provide a robust and efficient data structure that can be easily maintained and adapted as the medical center's needs evolve.

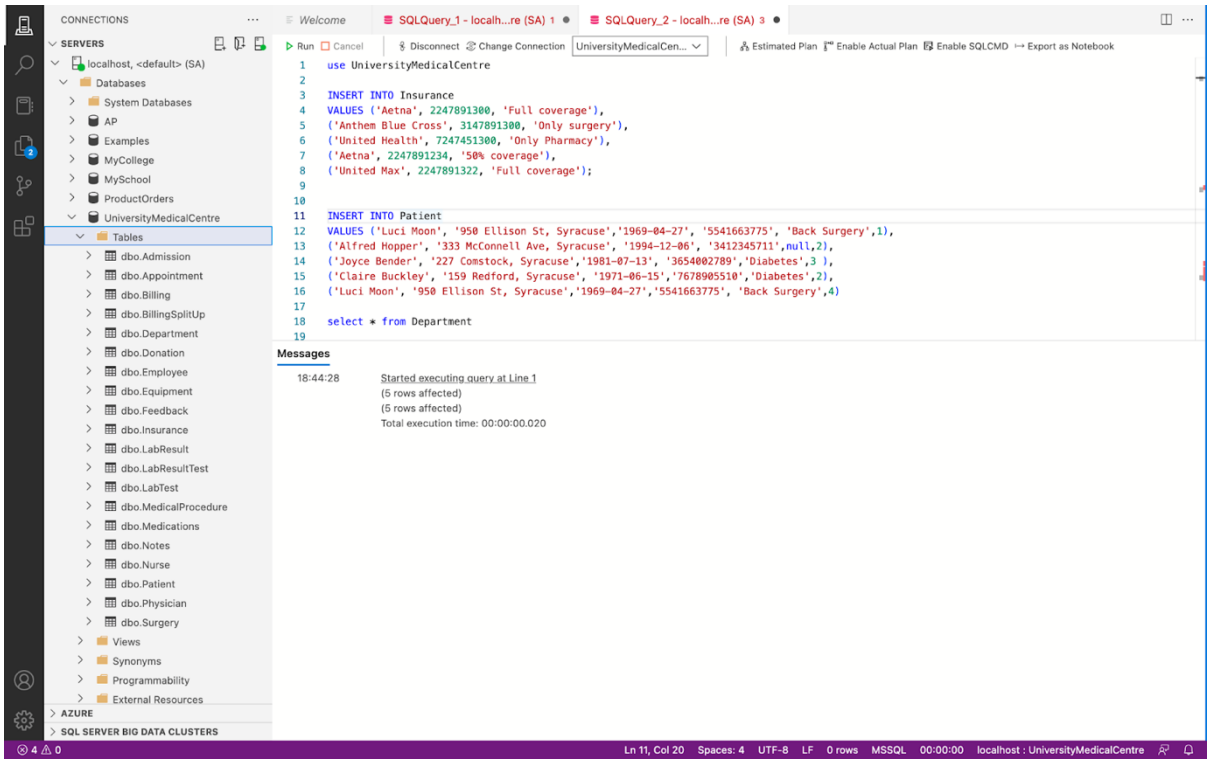
Insert:

```
INSERT INTO Insurance
VALUES ('Aetna', 2247891300, 'Full coverage'),
('Anthem Blue Cross', 3147891300, 'Only surgery'),
('United Health', 7247451300, 'Only Pharmacy'),
('Aetna', 2247891234, '50% coverage'),
('United Max', 2247891322, 'Full coverage');


INSERT INTO Patient
VALUES ('Luci Moon', '950 Ellison St, Syracuse','1969-04-27', '5541663775', 'Back
Surgery',1),
('Alfred Hopper', '333 McConnell Ave, Syracuse', '1994-12-06',
'3412345711',null,2),
('Joyce Bender', '227 Comstock, Syracuse','1981-07-13', '3654002789','Diabetes',3
),
('Claire Buckley', '159 Redford, Syracuse', '1971-06-
15','7678905510','Diabetes',2),
('Luci Moon', '950 Ellison St, Syracuse','1969-04-27','5541663775', 'Back
Surgery',4)

select * from Department


INSERT INTO Department VALUES
('General Surgery'),
('Emergency Department'),
('Gynecology'),
('Orthopedics'),
('Burn Care ');


INSERT INTO Employee  VALUES
( 'Derek Shepherd','Doctor','1969-04-27', 10000.00, '2020-04-27','2486663775',1),
( 'Meredith Grey','Nurse','1979-04-27', 10500.00, '2018-04-27','2486663773',2),
( 'Jackson Avery','Doctor','1965-12-20', 40000.00, '2020-04-27','2486665675',3),
( 'April Ludgard','Physicians','1964-04-27', 670000.00, '2019-04-
27','2486663835',1),
( 'Zack Hall','Doctor','1969-07-24', 880000.00, '2020-04-27','2486663775',4),
( 'Naz Ja','Nurse','1982-04-27', 10500.00, '2018-04-27','2486663773',2),
( 'Amay Grey','Nurse','1979-08-27', 10800.00, '2018-04-27','2486663473',2)


insert into Physician
values('Cardiologist','L121',1,4),
('Cardiologist','L121',2,1),
('Oncologist','L122',3,2),
('Surgeon','L123',2,3),
('Dermatologist','L124',5,5)
```

```
insert into Notes
values('Sign of Flues',9,1),
('Sign of infection',10,3),
('Sign of back pain',11,3),
('Sign of tonsils',12,4),
('Sign of skin infection',13,5)


insert into Nurse
values('Nurse Supervisor',1,2),
('Nurse Junior',2,6),
('Nurse sub Junior',5,7)


insert into Admission
values('2023-04-25','2023-04-26','',9)
    ('2022-03-15', '2022-03-20', 'Pneumonia', 1234, 5678);
select * from Patient
select * from Physician

insert into Admission
values
    ('2022-03-15', '2022-03-20', 'Pneumonia', 10, 2),
    ('2022-04-01', '2022-04-07', 'Appendicitis', 11, 3),
     ('2022-05-10', '2022-05-15', 'Fractured femur', 12, 1),
     ('2022-02-20', '2022-02-25', 'Myocardial infarction', 13, 4),
     ('2022-01-15', '2022-02-05', 'Chronic obstructive pulmonary disease', 9, 5)


insert into Billing values
(1250.50,3,5),
(800.00,5,1),
(2350.75,1,2),
(450.25,2,3),
(1900.00,4,4)


insert into BillingSplitUp values
(250.5,'Paid by Credit Card ending in 1234',1),
(1000.00,'Paid in Cash',1),
(2350.75,'Paid by Check #8394',3),
(450.25,'Paid in Cash',4),
(1900,'Paid by Insurance Company X',5)

insert into Appointment values
('2023-05-25 10:00:00.000','Follow-up for femur injury',3,12),
('2023-05-03 15:30:00.000','Annual physical exam',1,13),
('2023-05-05 11:15:00.000','Sinus infection',5,11),
('2023-05-07 13:45:00.000','Psoriasis checkup',2,13),
```

```
('2023-05-27 09:30:00.000','Routine blood work',3,10)

Insert into MedicalProcedure
values('Appendectomy','2022-03-15',6,1,1),
('Knee replacement','2021-07-22',7,2,2),
('Hernia Removal','2022-01-05',8,3,3),
('Throat Infection checkup','2023-02-18',9,4,4),
('Skin Laser Treatment','2022-06-09',10,5,5);

Insert into Feedback values
('Dr. Hall was very thorough during my exam.',1,12),
('Nurse Jones was very friendly and helpful.',5,12),
('The office staff was very efficient.',4   ,11),
('The wait time was a bit long, but worth it.',3    ,13)
('Dr. Hall explained everything clearly.',2,12)

insert into Donation values
('John Doe', 500.00, '2022-03-15', 'Research', 1),
('Jane Smith', 1000.00, '2022-04-01', 'Scholarship', 2),
('Michael Lee', 250.00, '2022-05-12', 'Equipment', 3),
('Emily Jones', 100.00, '2022-06-22', 'General Fund', 4),
('Robert Johnson', 750.00, '2022-07-05', 'Student Services', 5)


insert into LabTest
values('Complete blood count'),
('Basic metabolic panel'),
('Comprehensive metabolic panel'),
('Lipid panel'),
('Thyroid panel');


insert into LabResult
values(1,1),
(2,2),
(3,3),
(4,4),
(5,5);


insert into LabResultTest
values(1,1,'Positive'),
(2,2,'Positive'),
(3,3,'Positive'),
(4,4,'Positive'),
(5,5,'Positive');


INSERT into Equipment values
    ('Stethoscope','Medtronic','2022-03-20',6),
    ('Sphygmomanometer','Johnson & Johnson','2022-03-20',7),
    ('Stretcher','Abbott Laboratories','2021-03-10',8),
```

```
    ('Scalpel','Abbott Laboratories','2021-09-10',10),
    ('Syringe','Cardinal Health','2020-09-20',11)


INSERT into Surgery values
('2022-03-20',302,'BP is normal', 10,12),
('2022-03-20',200,'Pre-checks look good',9,11),
('2022-03-20',305,'Hip replacement is needed', 11,11),
('2022-03-20',309,'Eye surgery is needed for cataract in this case', 12,9),
('2022-03-20',201,'BP is borderline', 8,8)


Insert into Medications values
('For pain relief', 'Ibuprofen', 200, 3, 9, 1),
('For high blood pressure', 'Lisinopril', 10, 1, 10, 2),
('For anxiety', 'Alprazolam', 0.5, 2, 11, 3),
('For heartburn', 'Omeprazole', 20, 1, 12, 4),
('For migraines', 'Sumatriptan', 50, 2, 13, 5);
```



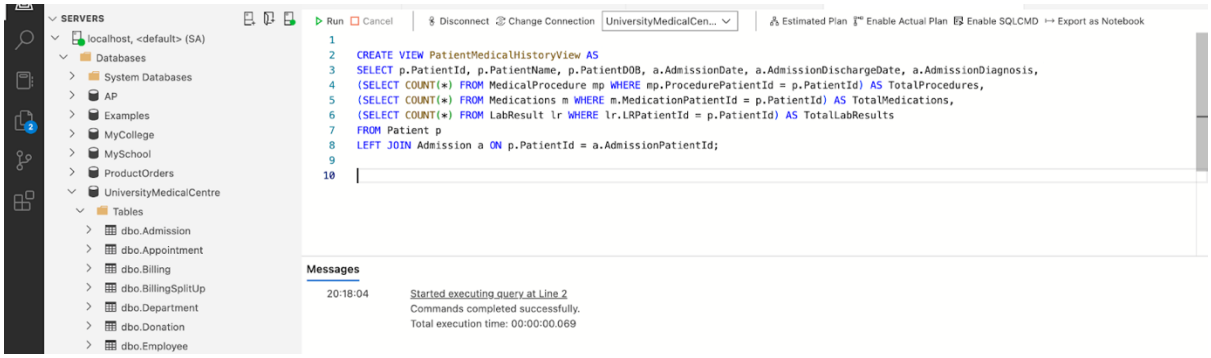
The provided SQL statements create and populate tables for the University Medical Centre database, covering different aspects of the center's operations like patient information, insurance, departments, employees, appointments, and more. The INSERT statements add sample data to the tables, allowing for testing and further development. These sample records will help in understanding the database structure, relationships between tables, and how to query data effectively. With this foundation, the database can be expanded and adapted to better serve the medical center's evolving needs.

Views:

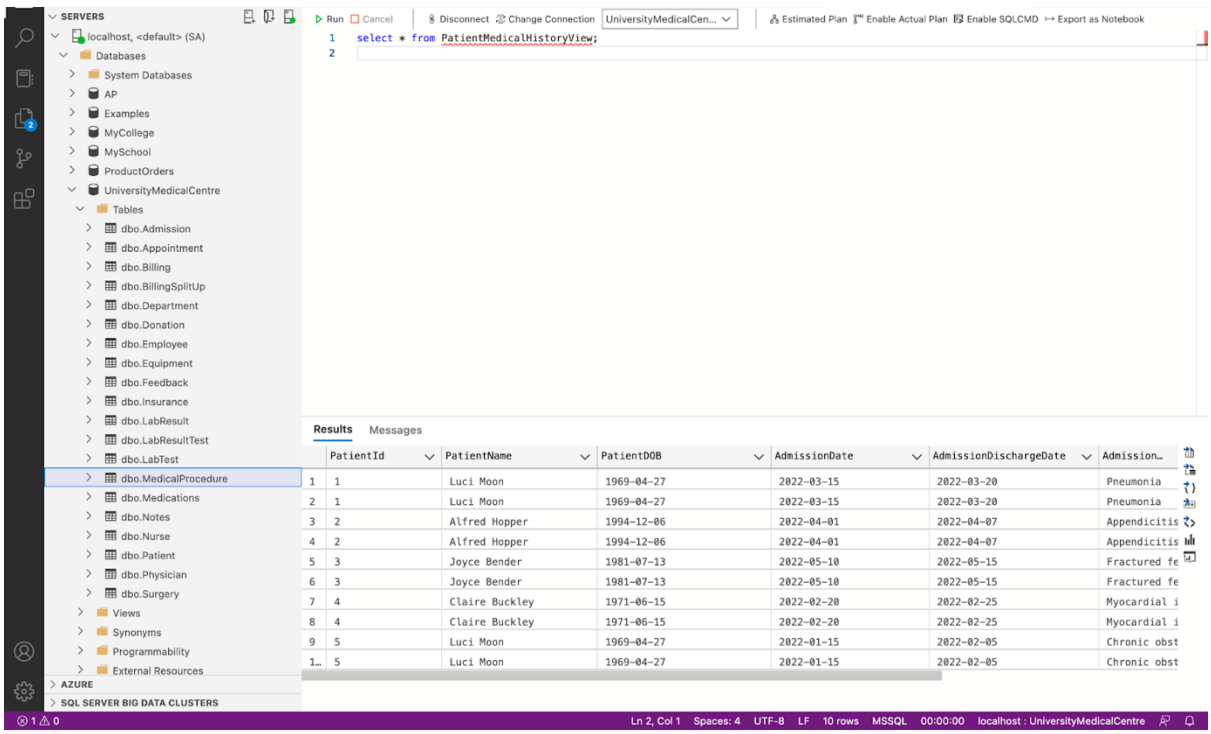
View 1: Patient Medical History Summary

This view provides a summary of each patient's medical history, including the total number of procedures, medications, and lab results. It combines data from the Patient, Admission, MedicalProcedure, Medications, and LabResult tables.

```
CREATE VIEW PatientMedicalHistoryView AS
SELECT p.PatientId, p.PatientName, p.PatientDOB, a.AdmissionDate,
a.AdmissionDischargeDate, a.AdmissionDiagnosis,
(SELECT COUNT(*) FROM MedicalProcedure mp WHERE mp.ProcedurePatientId =
p.PatientId) AS TotalProcedures,
(SELECT COUNT(*) FROM Medications m WHERE m.MedicationPatientId = p.PatientId) AS
TotalMedications,
(SELECT COUNT(*) FROM LabResult lr WHERE lr.LRPatientId = p.PatientId) AS
TotalLabResults
FROM Patient p
LEFT JOIN Admission a ON p.PatientId = a.AdmissionPatientId;
```



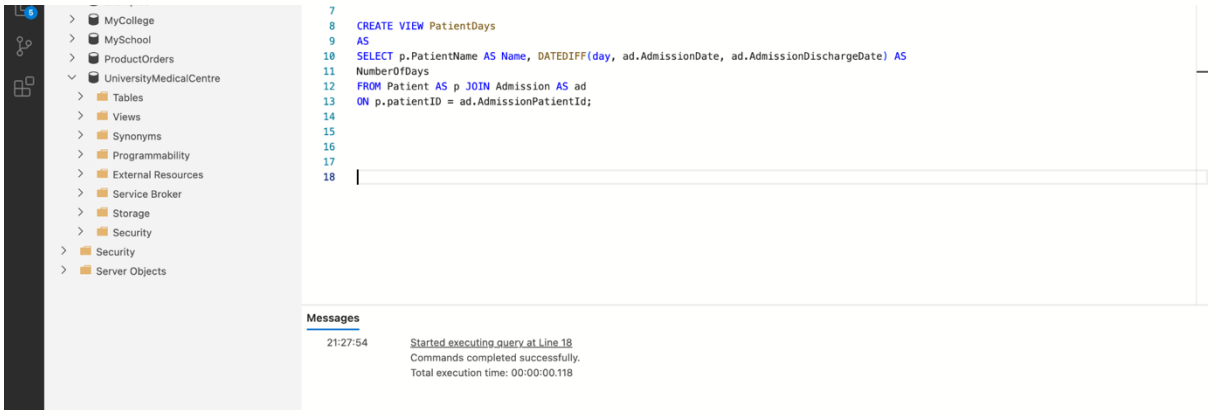

```
Select * from PatientMedicalHistoryView;
```



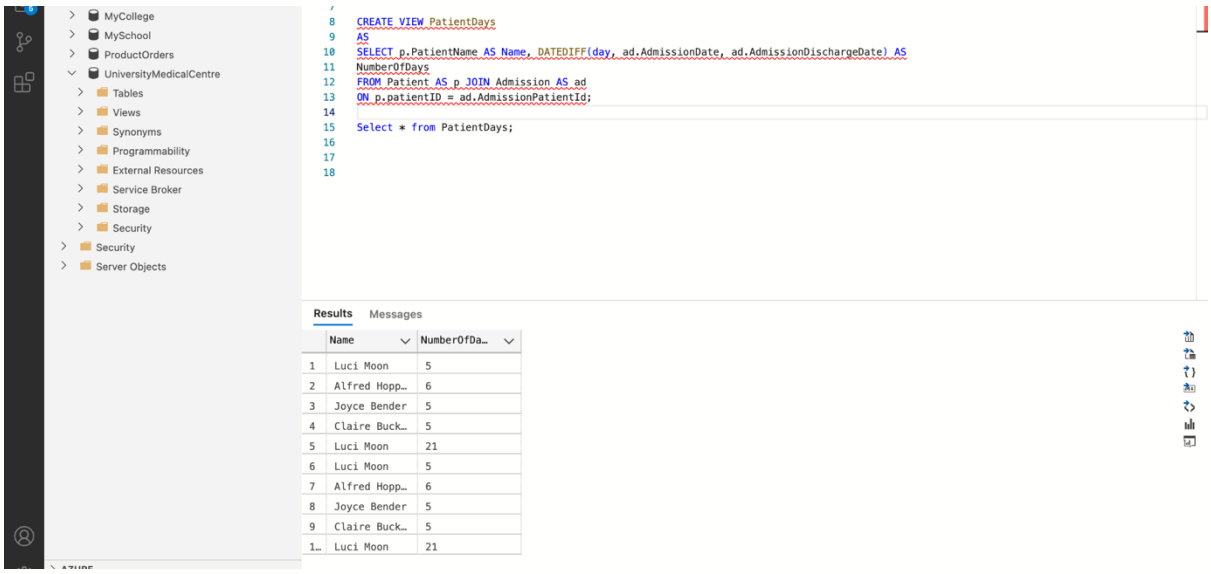
View 2: Patient Hospital Stay Duration

This view calculates the total number of days each patient was admitted to the hospital. It returns two columns: the patient's name and the number of days they spent at the hospital, ordered from the longest to shortest stay duration.

```
CREATE VIEW PatientDays
AS
SELECT p.PatientName AS Name, DATEDIFF(day, ad.AdmissionDate,
ad.AdmissionDischargeDate) AS
NumberOfDays
FROM Patient AS p JOIN Admission AS ad
ON p.patientID = ad.AdmissionPatientId;
```



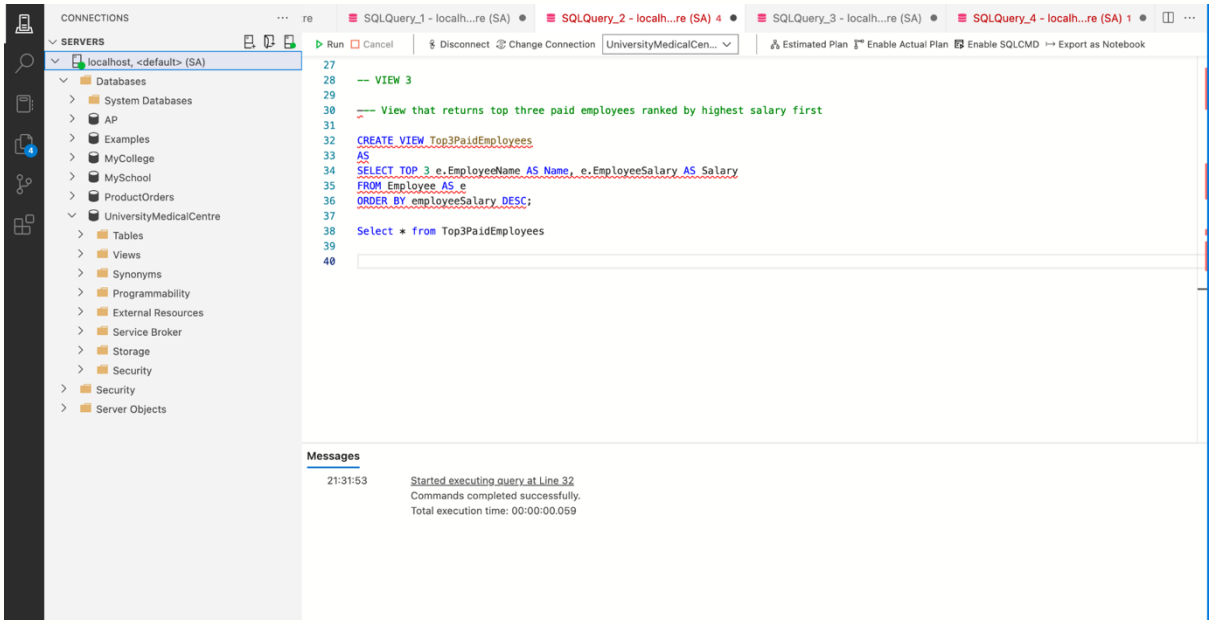

```
Select * from PatientDays;
```



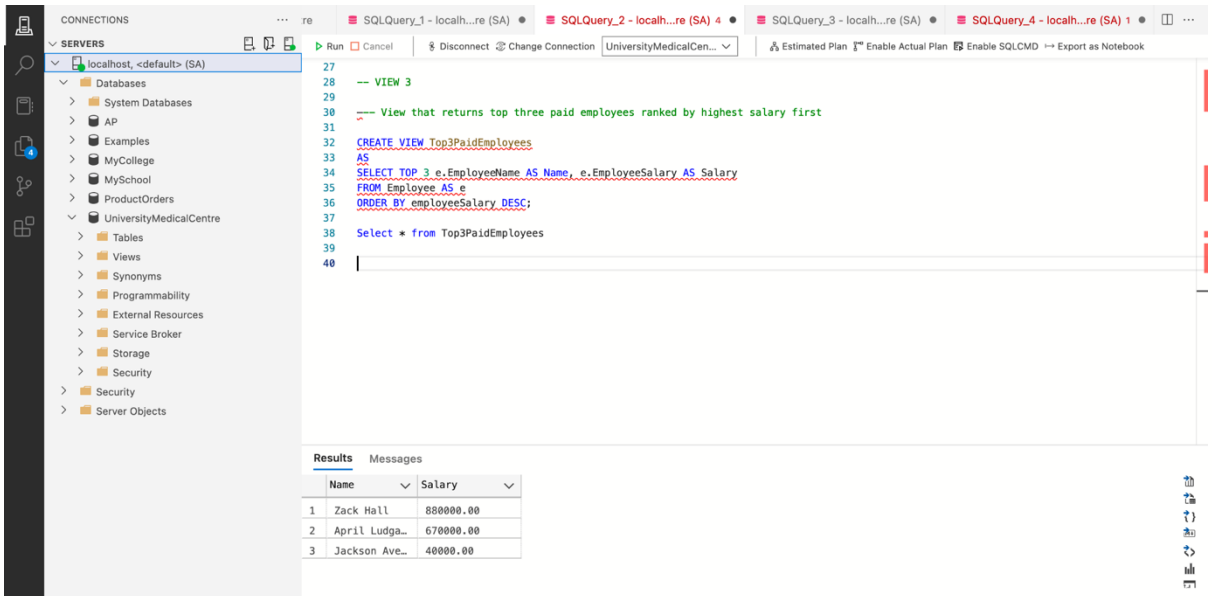
View 3: Top 3 Highest Paid Employees

This view ranks the top three highest-paid employees based on their salary. It returns two columns: the employee's name and their salary.

```
CREATE VIEW Top3PaidEmployees
AS
SELECT TOP 3 e.EmployeeName AS Name, e.EmployeeSalary AS Salary
FROM Employee AS e
ORDER BY employeeSalary DESC;
Select * from Top3PaidEmployees
```



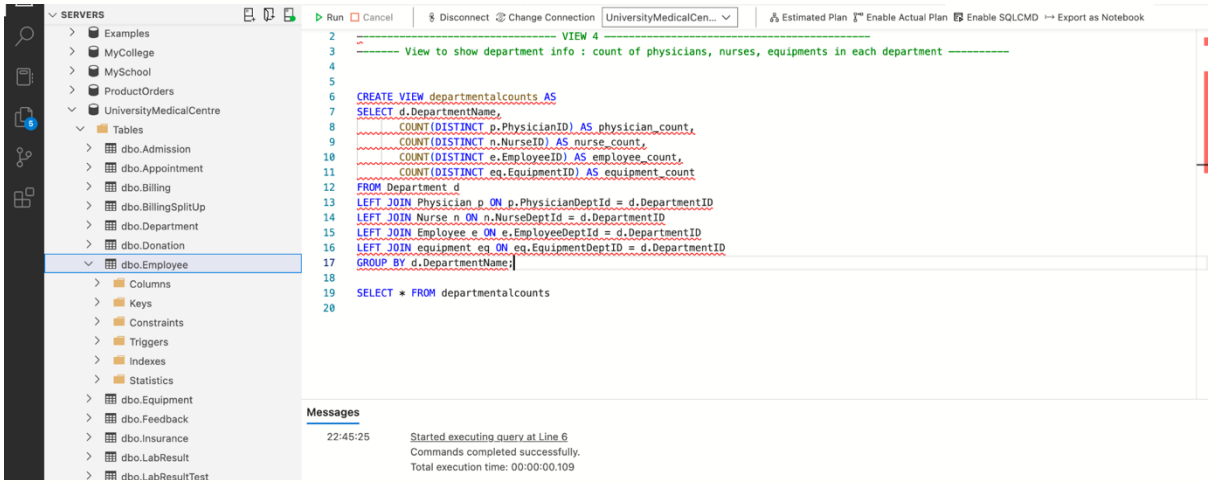

```
Select * from Top3PaidEmployees
```



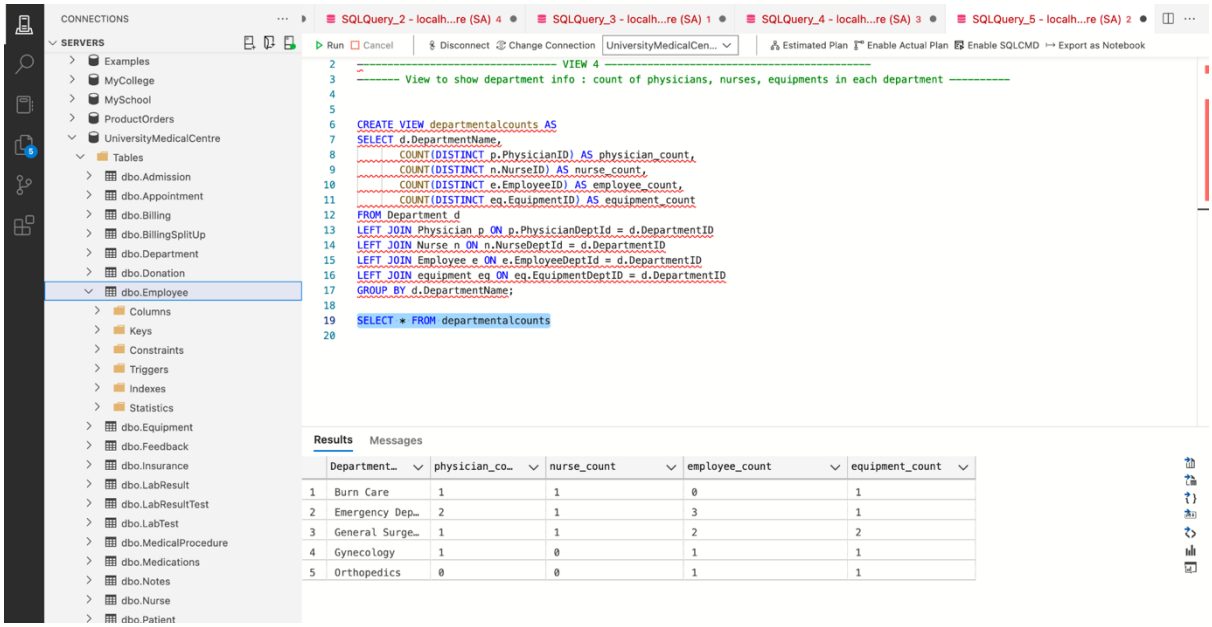
View 4: Department Overview

This view provides an overview of each department, including the number of physicians, nurses, employees, and equipment within each department. It combines data from the Department, Physician, Nurse, Employee, and Equipment tables.

```
CREATE VIEW departmentalcounts AS
SELECT d.DepartmentName,
       COUNT(DISTINCT p.PhysicianID) AS physician_count,
       COUNT(DISTINCT n.NurseID) AS nurse_count,
       COUNT(DISTINCT e.EmployeeID) AS employee_count,
       COUNT(DISTINCT eq.EquipmentID) AS equipment_count
FROM Department d
LEFT JOIN Physician p ON p.PhysicianDeptId = d.DepartmentID
LEFT JOIN Nurse n ON n.NurseDeptId = d.DepartmentID
LEFT JOIN Employee e ON e.EmployeeDeptId = d.DepartmentID
LEFT JOIN equipment eq ON eq.EquipmentDeptID = d.DepartmentID
GROUP BY d.DepartmentName;
```




```
SELECT * FROM departmentalcounts
```

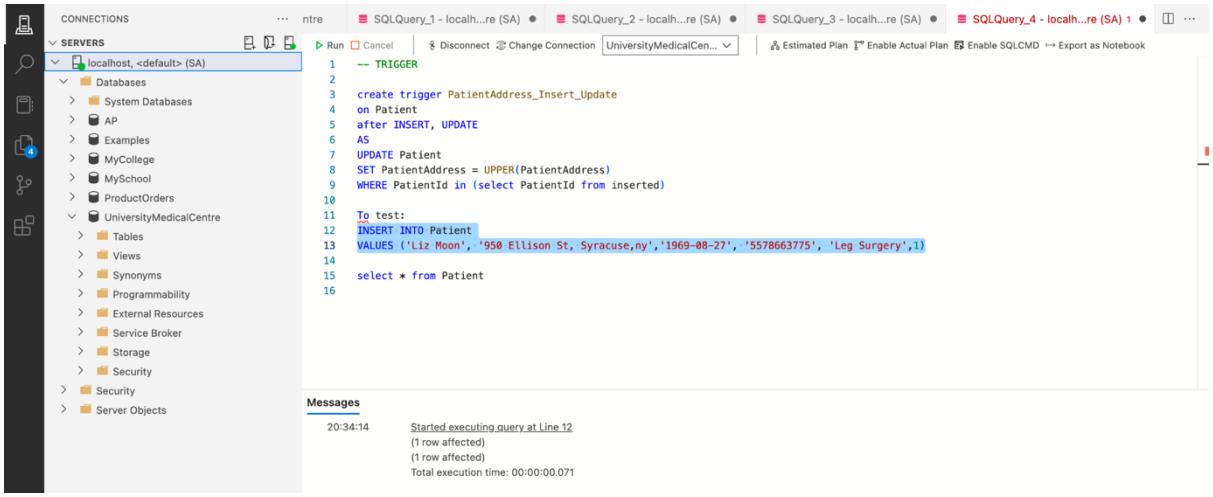


Triggers:

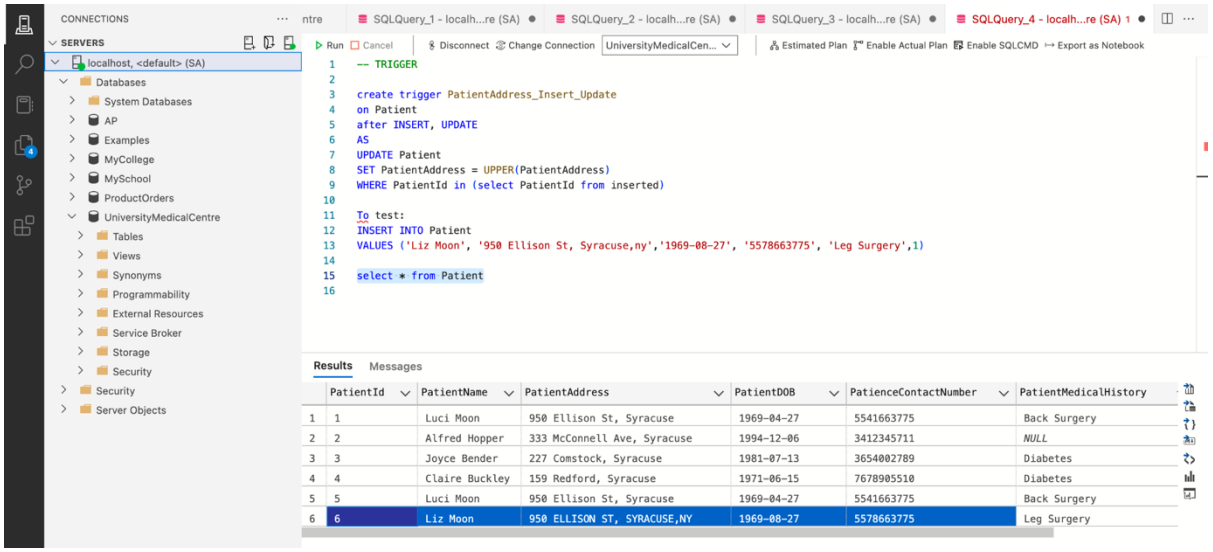
Trigger 1: Capitalize Patient Address

This trigger is designed to capitalize the PatientAddress field when a new patient record is inserted or an existing patient record is updated. It is applied to the Patient table and affects both INSERT and UPDATE operations.

```
create trigger PatientAddress_Insert_Update
on Patient
after INSERT, UPDATE
AS
UPDATE Patient
SET PatientAddress = UPPER(PatientAddress)
WHERE PatientId in (select PatientId from inserted)

--To test:
INSERT INTO Patient
VALUES ('Liz Moon', '950 Ellison St, Syracuse,ny','1969-08-27', '5578663775', 'Leg Surgery',1)
```




```
Select * from Patient
```

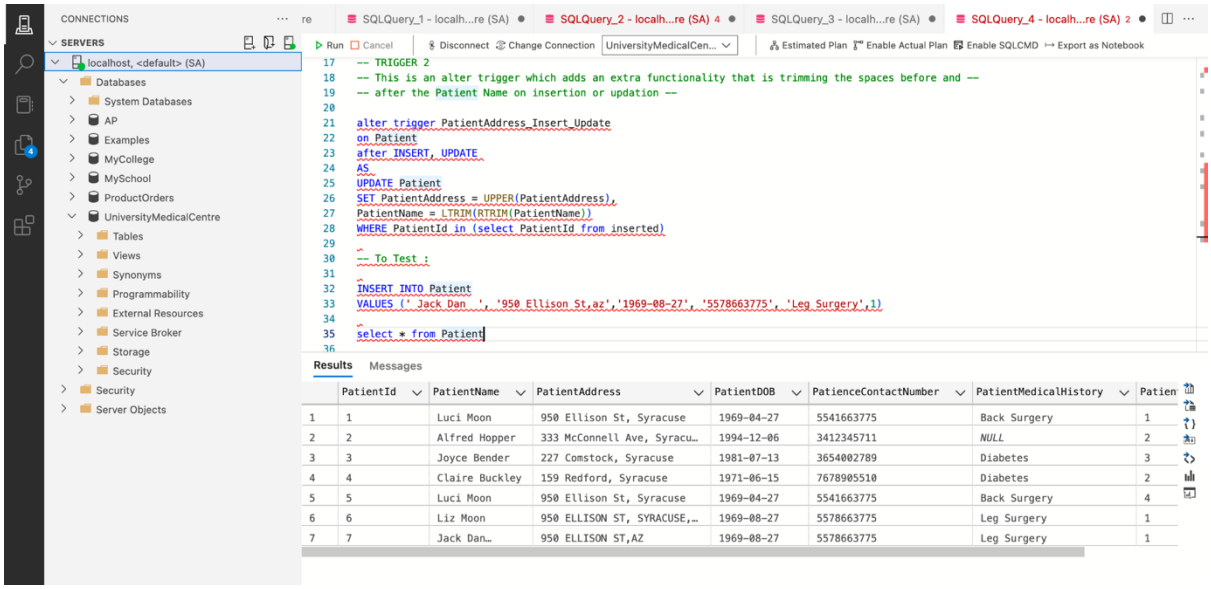


Trigger 2: Trim Patient Name

This trigger is an updated version of Trigger 1, with an added functionality to trim any leading and trailing spaces from the PatientName field when a new patient record is inserted or an existing patient record is updated. It is applied to the Patient table and affects both INSERT and UPDATE operations.

```
alter trigger PatientAddress_Insert_Update
on Patient
after INSERT, UPDATE
AS
UPDATE Patient
SET PatientAddress = UPPER(PatientAddress),
PatientName = LTRIM(RTRIM(PatientName))
WHERE PatientId in (select PatientId from inserted)

-- To Test :
INSERT INTO Patient
VALUES (' Jack Dan  ', '950 Ellison St,az','1969-08-27', '5578663775', 'Leg
Surgery',1)
```

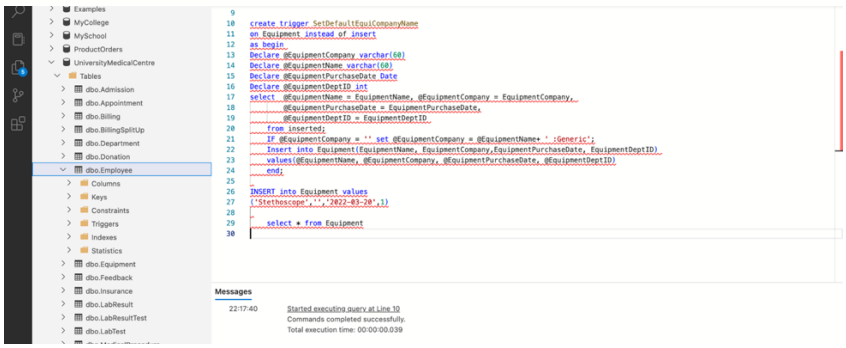

```
select * from Patient;
```



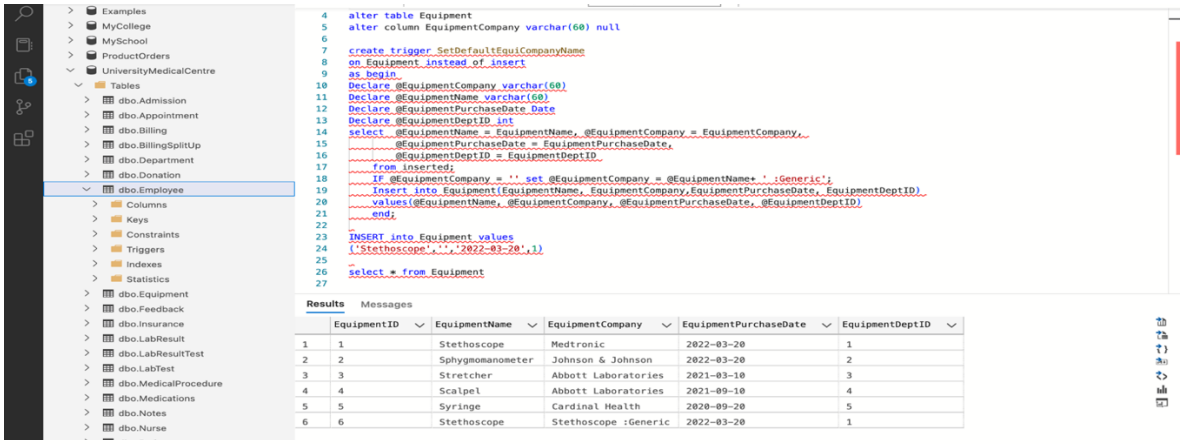
Trigger 3: Set Default Equipment Company Name

This trigger ensures that if no value is provided for the EquipmentCompany field in the Equipment table when a new record is inserted, a default value will be assigned as a combination of the EquipmentName and the text " :Generic". This trigger uses the INSTEAD OF INSERT type to replace the original insert operation with a custom insert operation that sets the default value if necessary.

```
alter table Equipment
alter column EquipmentCompany varchar(60) null
create trigger SetDefaultEquiCompanyName
on Equipment instead of insert
as begin
Declare @EquipmentCompany varchar(60)
Declare @EquipmentName varchar(60)
Declare @EquipmentPurchaseDate Date
Declare @EquipmentDeptID int
select  @EquipmentName = EquipmentName, @EquipmentCompany = EquipmentCompany,
       @EquipmentPurchaseDate = EquipmentPurchaseDate,
       @EquipmentDeptID = EquipmentDeptID
   from inserted;
   IF @EquipmentCompany = '' set @EquipmentCompany = @EquipmentName+ ' :Generic';
   Insert into Equipment(EquipmentName, EquipmentCompany,EquipmentPurchaseDate,
EquipmentDeptID)
   values(@EquipmentName, @EquipmentCompany, @EquipmentPurchaseDate,
@EquipmentDeptID)
   end;

INSERT into Equipment values
('Stethoscope','','2022-03-20',1)
```



```
Select * from Equipment
```

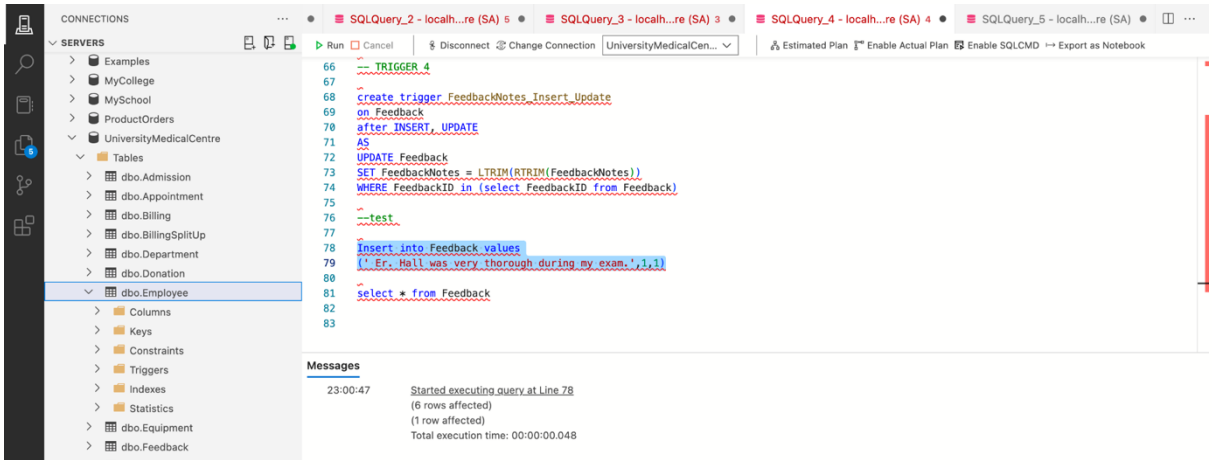


	EquipmentID	EquipmentName	EquipmentCompany	EquipmentPurchaseDate	EquipmentDeptID
1	1	Stethoscope	Medtronic	2022-03-20	1
2	2	Sphygmomanometer	Johnson & Johnson	2022-03-20	2
3	3	Stretcher	Abbott Laboratories	2021-03-10	3
4	4	Scalpel	Abbott Laboratories	2021-09-10	4
5	5	Syringe	Cardinal Health	2020-09-20	5
6	6	Stethoscope	Stethoscope :Generic	2022-03-20	1

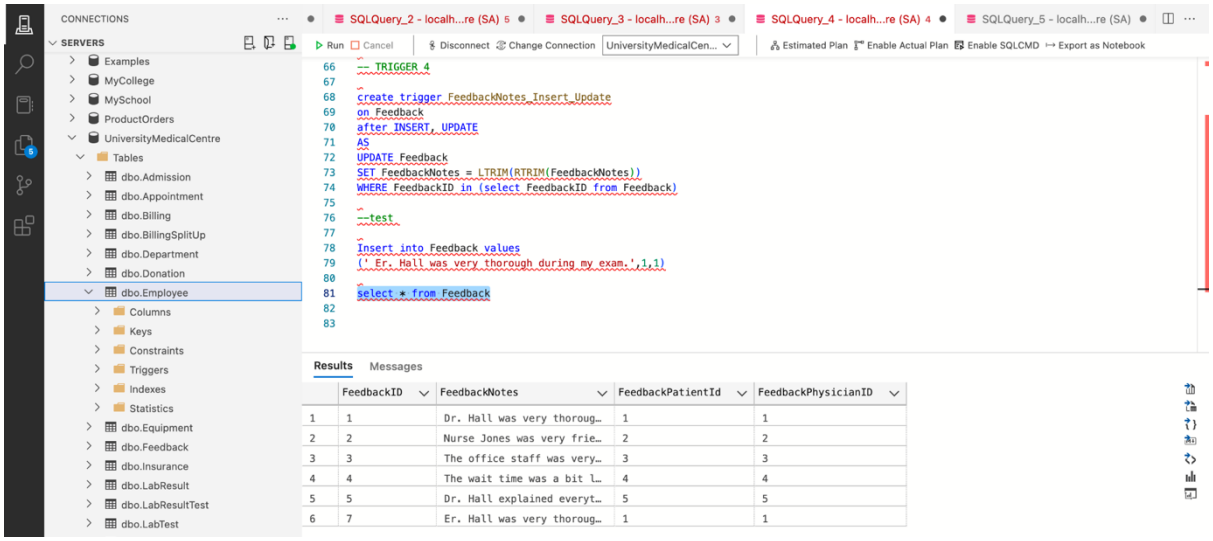
Trigger 4: Trim Feedback Notes

This trigger trims any leading and trailing spaces from the FeedbackNotes field when a new feedback record is inserted or an existing feedback record is updated. It is applied to the Feedback table and affects both INSERT and UPDATE operations.

```
create trigger FeedbackNotes_Insert_Update
on Feedback
after INSERT, UPDATE
AS
UPDATE Feedback
SET FeedbackNotes = LTRIM(RTRIM(FeedbackNotes))
WHERE FeedbackID in (select FeedbackID from Feedback)

--test
Insert into Feedback values
(' Er. Hall was very thorough during my exam.',1,1)
```




```
select * from Feedback
```

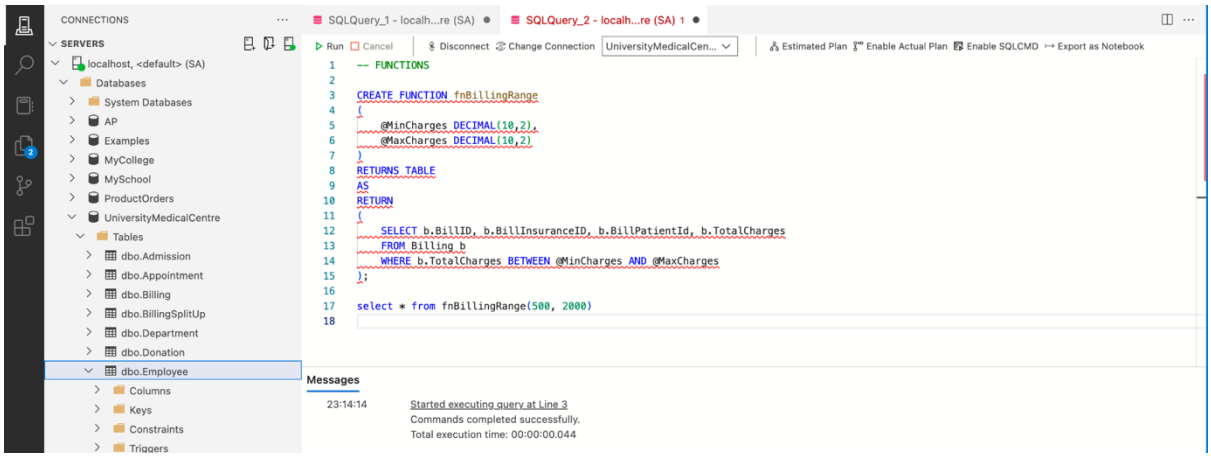


Functions:

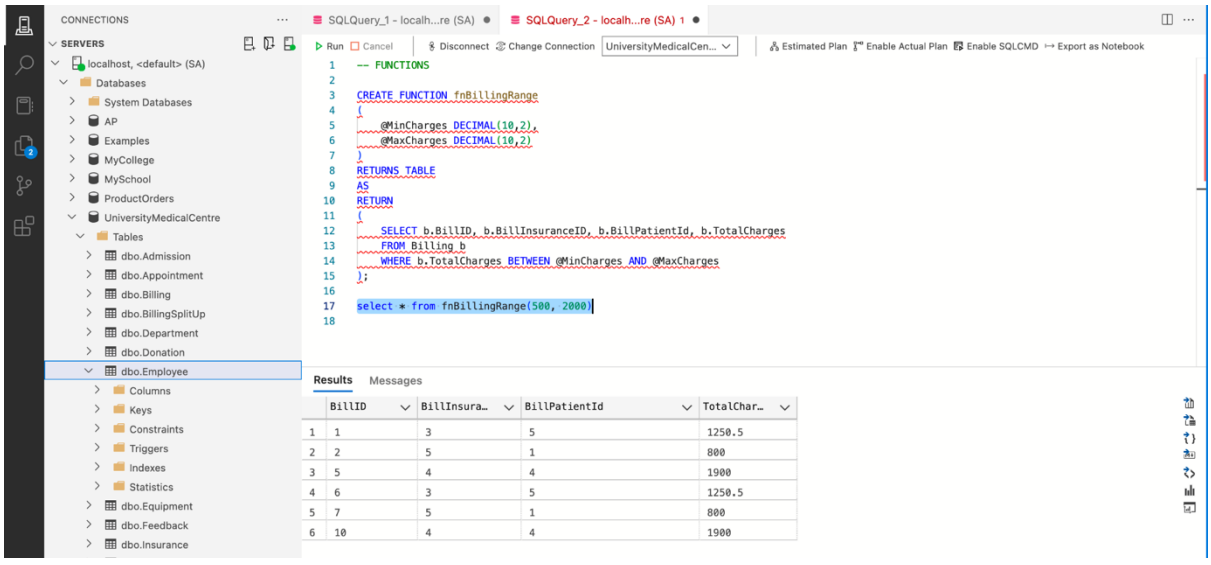
Function 1: Billing Range

This function retrieves billing records with total charges within a specified range. The user provides the minimum and maximum charges as parameters, and the function returns the BillID, BillInsuranceID, BillPatientId, and TotalCharges for the matching records.

```
CREATE FUNCTION fnBillingRange
(
   @MinCharges DECIMAL(10,2),
   @MaxCharges DECIMAL(10,2)
)
RETURNS TABLE
AS
RETURN
(
   SELECT b.BillID, b.BillInsuranceID, b.BillPatientId, b.TotalCharges
   FROM Billing b
   WHERE b.TotalCharges BETWEEN @MinCharges AND @MaxCharges
);
```



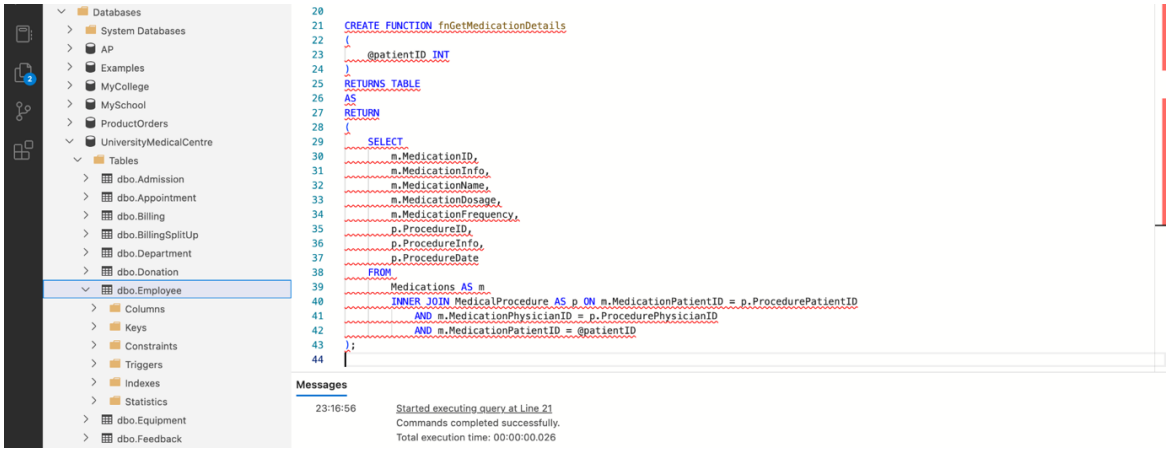

```
select * from fnBillingRange(500, 2000)
```



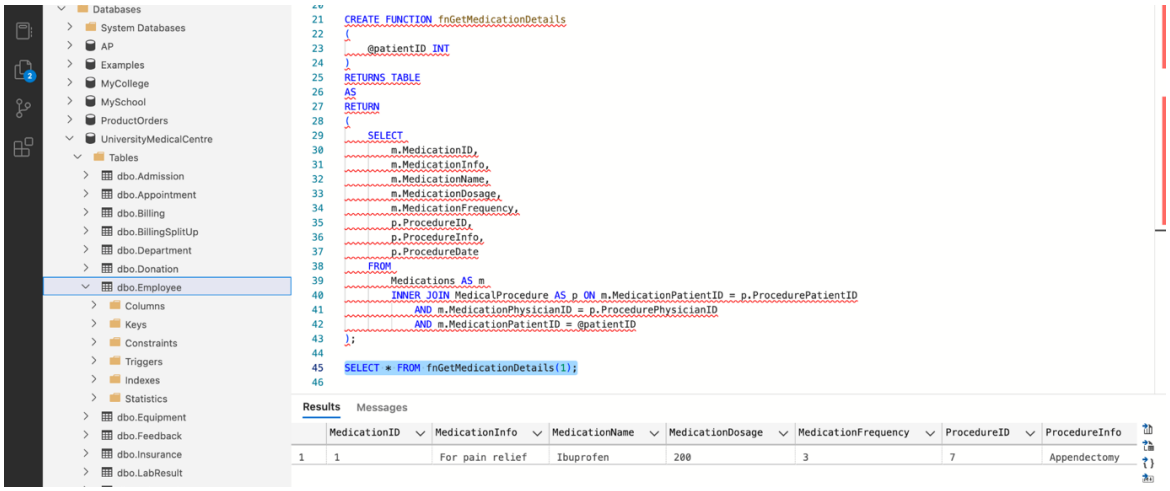
Function 2: Get Medication Details

This function returns medication and procedure information for a specific patient. The user provides the patient ID as a parameter, and the function returns details such as MedicationID, MedicationInfo, MedicationName, MedicationDosage, MedicationFrequency, ProcedureID, ProcedureInfo, and ProcedureDate.

```
CREATE FUNCTION fnGetMedicationDetails
(
   @patientID INT
)
RETURNS TABLE
AS
RETURN
(
   SELECT
        m.MedicationID,
        m.MedicationInfo,
        m.MedicationName,
        m.MedicationDosage,
        m.MedicationFrequency,
        p.ProcedureID,
        p.ProcedureInfo,
        p.ProcedureDate
   FROM
        Medications AS m
        INNER JOIN MedicalProcedure AS p ON m.MedicationPatientID =
p.ProcedurePatientID
            AND m.MedicationPhysicianID = p.ProcedurePhysicianID
            AND m.MedicationPatientID = @patientID
);
```



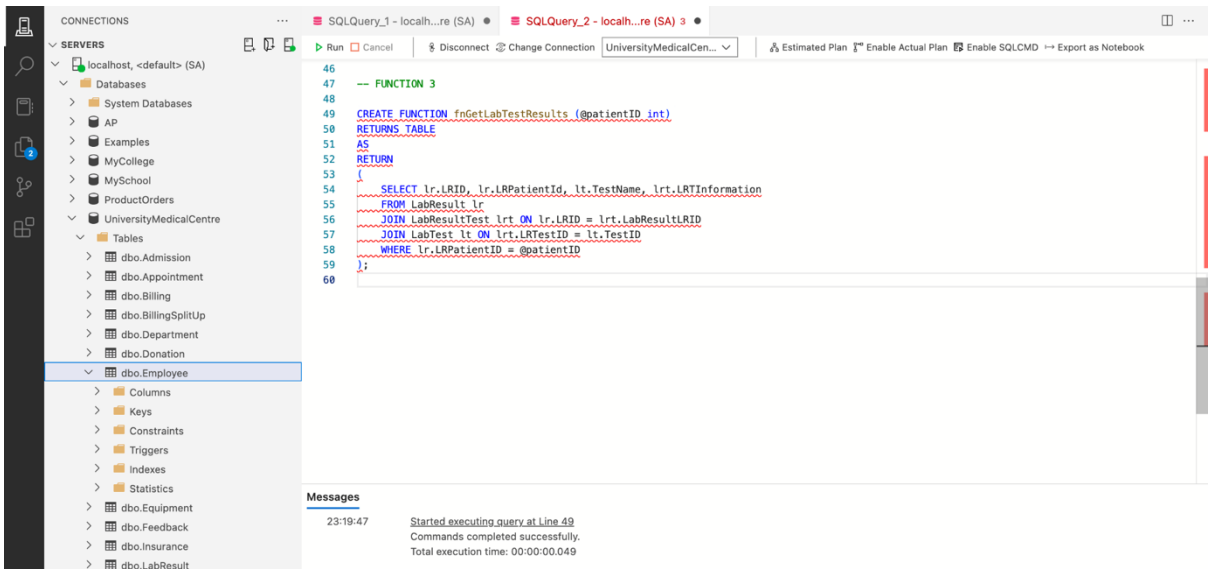

```
SELECT * FROM fnGetMedicationDetails(1);
```



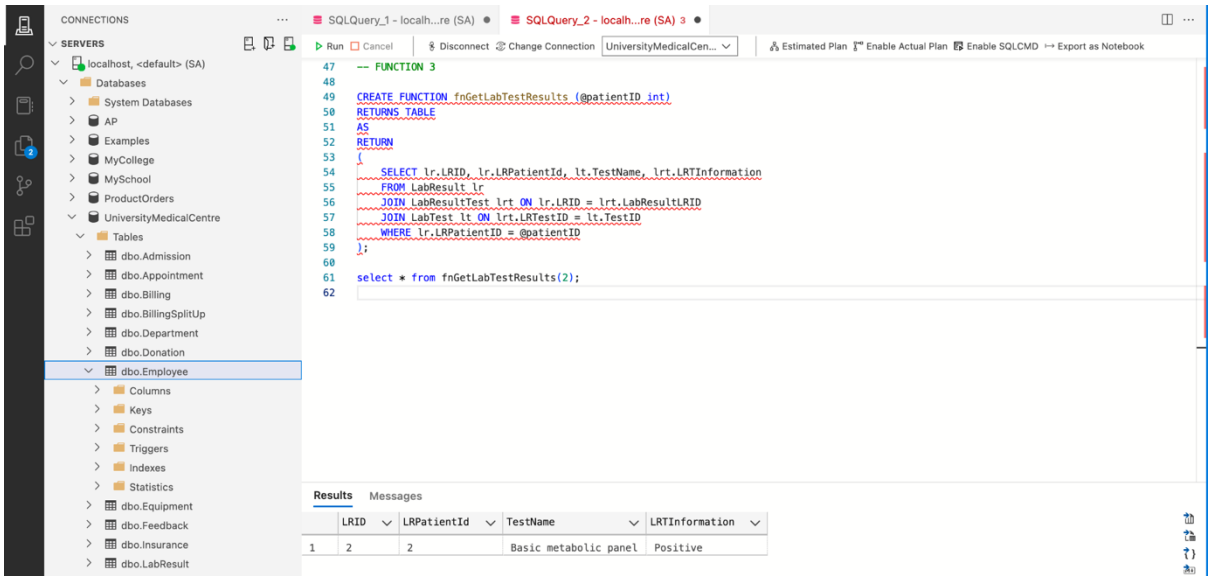
Function 3: Get Lab Test Results

This function retrieves lab test results for a specific patient. The user provides the patient ID as a parameter, and the function returns details such as LRID, LRPatientId, TestName, and LRTInformation.

```
CREATE FUNCTION fnGetLabTestResults (@patientID int)
RETURNS TABLE
AS
RETURN
(
   SELECT lr.LRID, lr.LRPatientId, lt.TestName, lrt.LRTInformation
   FROM LabResult lr
   JOIN LabResultTest lrt ON lr.LRID = lrt.LabResultLRID
   JOIN LabTest lt ON lrt.LRTestID = lt.TestID
   WHERE lr.LRPatientID = @patientID
);
```



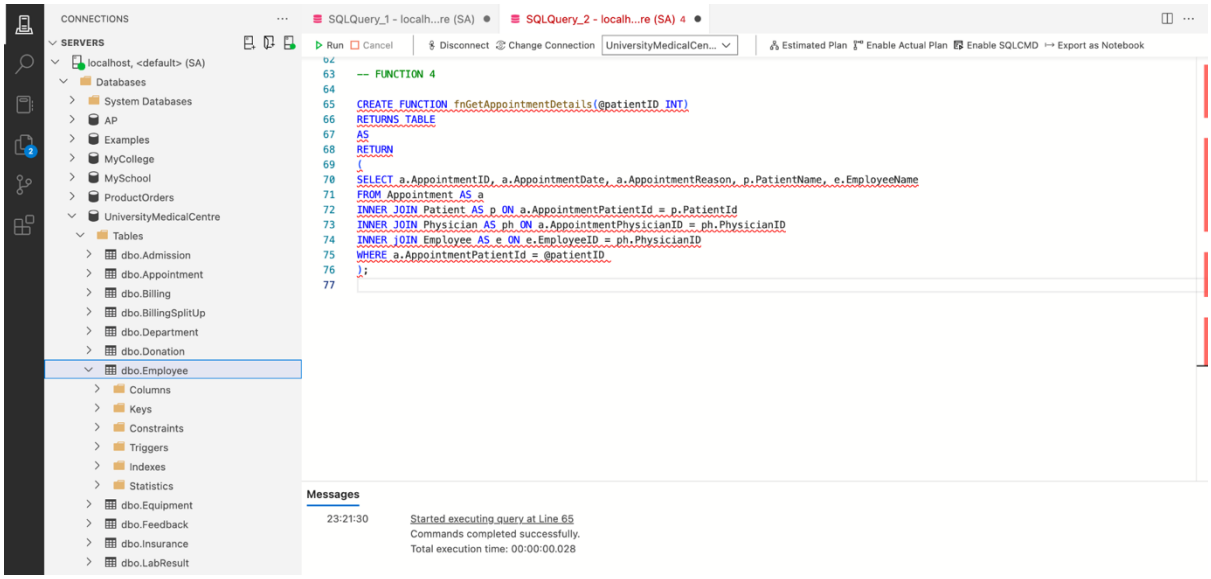

```
select * from fnGetLabTestResults(2);
```



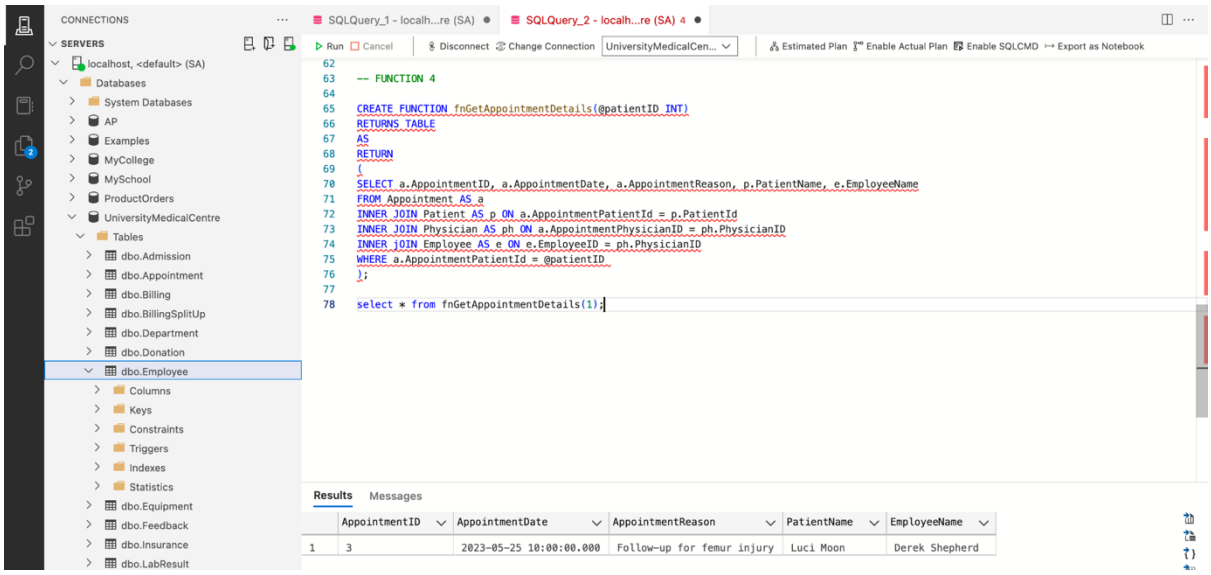
Function 4: Get Appointment Details

This function returns appointment details for a specific patient. The user provides the patient ID as a parameter, and the function returns details such as AppointmentID, AppointmentDate, AppointmentReason, PatientName, and EmployeeName (physician).

```
CREATE FUNCTION fnGetAppointmentDetails(@patientID INT)
RETURNS TABLE
AS
RETURN
(
SELECT a.AppointmentID, a.AppointmentDate, a.AppointmentReason, p.PatientName,
e.EmployeeName
FROM Appointment AS a
INNER JOIN Patient AS p ON a.AppointmentPatientId = p.PatientId
INNER JOIN Physician AS ph ON a.AppointmentPhysicianID = ph.PhysicianID
INNER jOIN Employee AS e ON e.EmployeeID = ph.PhysicianID
WHERE a.AppointmentPatientId = @patientID
);
```




```
Select * from fnGetAppointmentDetails(1)
```

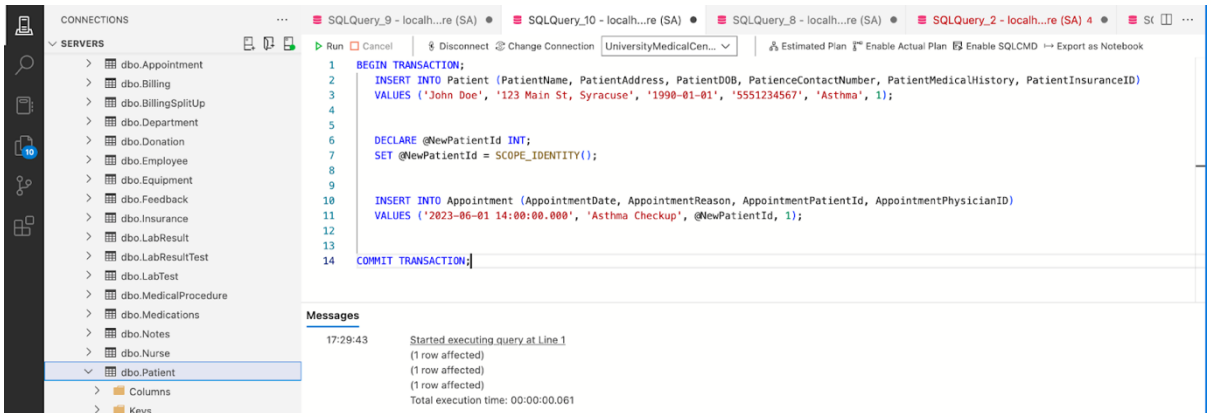


Transactions:

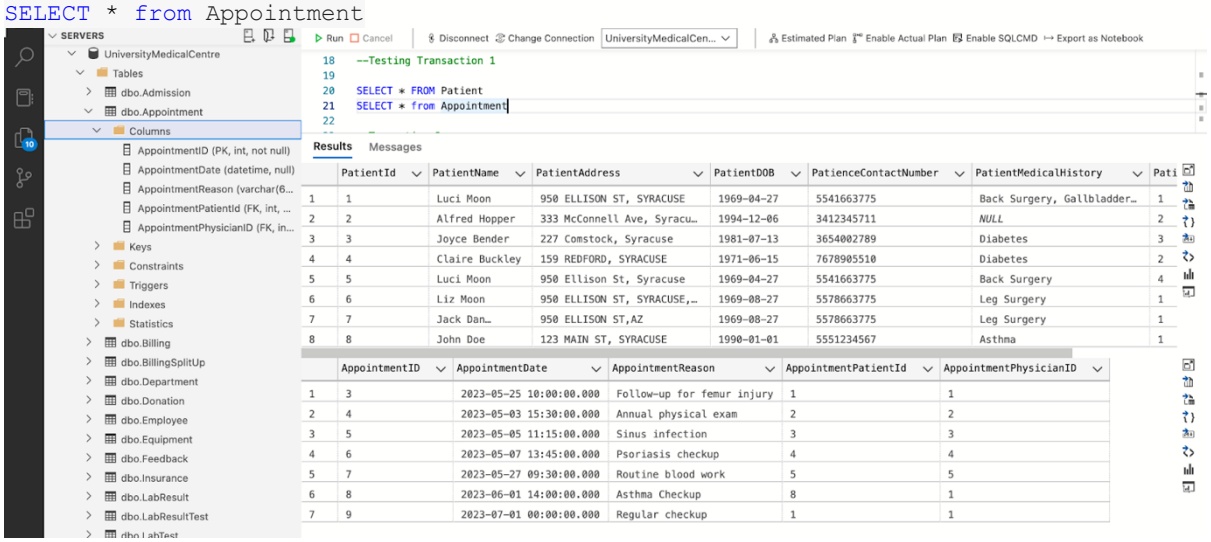
Transaction 1: Add New Patient and Appointment

This transaction inserts a new patient record and creates an appointment for the new patient. It starts by adding a patient with their details and then uses the newly created patient ID to insert an appointment record for that patient. After executing, the transaction is committed.

```
BEGIN TRANSACTION;
   INSERT INTO Patient (PatientName, PatientAddress, PatientDOB,
PatienceContactNumber, PatientMedicalHistory, PatientInsuranceID)
   VALUES ('John Doe', '123 Main St, Syracuse', '1990-01-01', '5551234567',
'Asthma', 1);

   DECLARE @NewPatientId INT;
   SET @NewPatientId = SCOPE_IDENTITY();

   INSERT INTO Appointment (AppointmentDate, AppointmentReason,
AppointmentPatientId, AppointmentPhysicianID)
   VALUES ('2023-06-01 14:00:00.000', 'Asthma Checkup', @NewPatientId, 1);

COMMIT TRANSACTION;
```



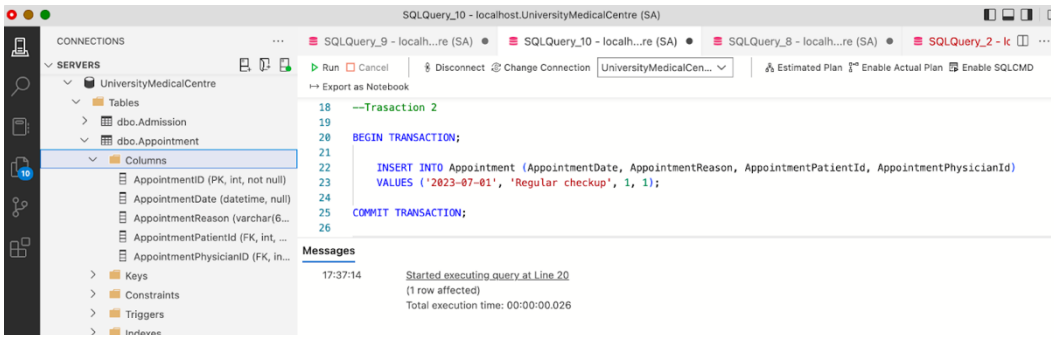

```
--Testing Transaction 1
SELECT * FROM Patient
SELECT * from Appointment
```



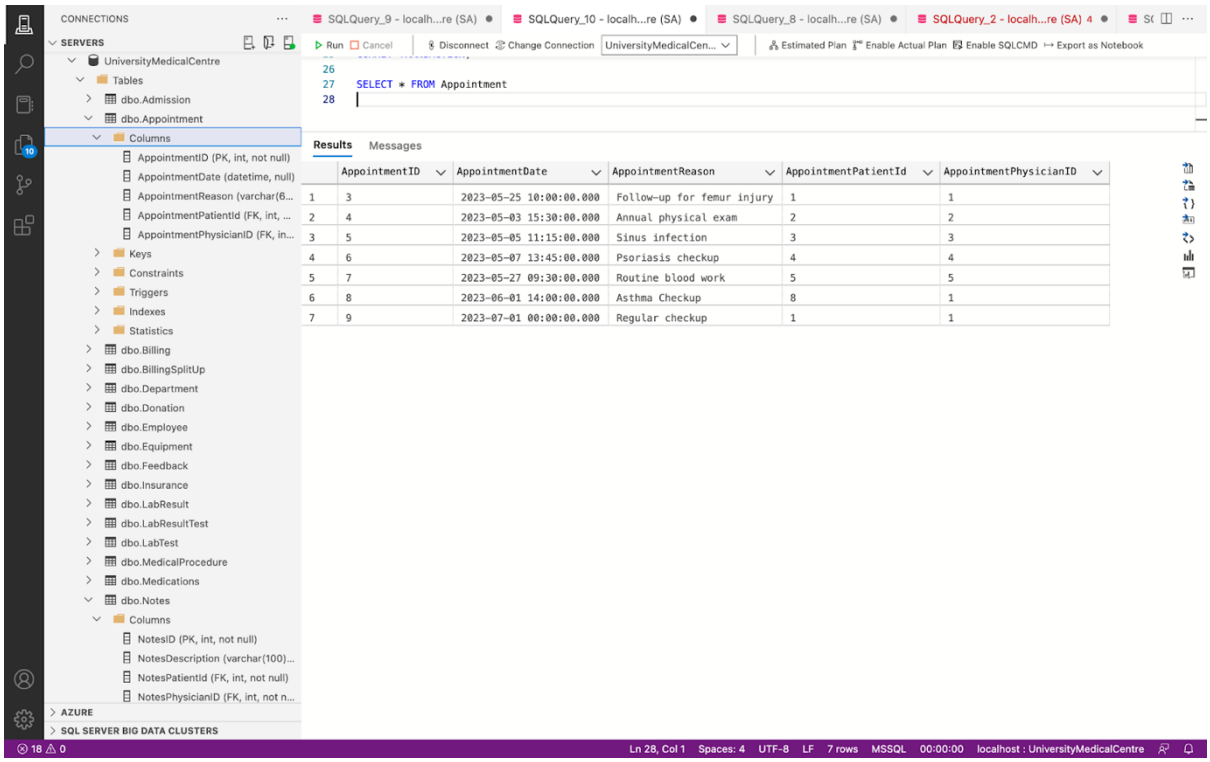
Transaction 2: Add New Appointment

This transaction creates a new appointment for an existing patient with a physician. It inserts a new appointment record with the patient ID, physician ID, appointment date, and appointment reason, then commits the transaction.

```
BEGIN TRANSACTION;

   INSERT INTO Appointment (AppointmentDate, AppointmentReason,
AppointmentPatientId, AppointmentPhysicianId)
   VALUES ('2023-07-01', 'Regular checkup', 1, 1);

COMMIT TRANSACTION;
```



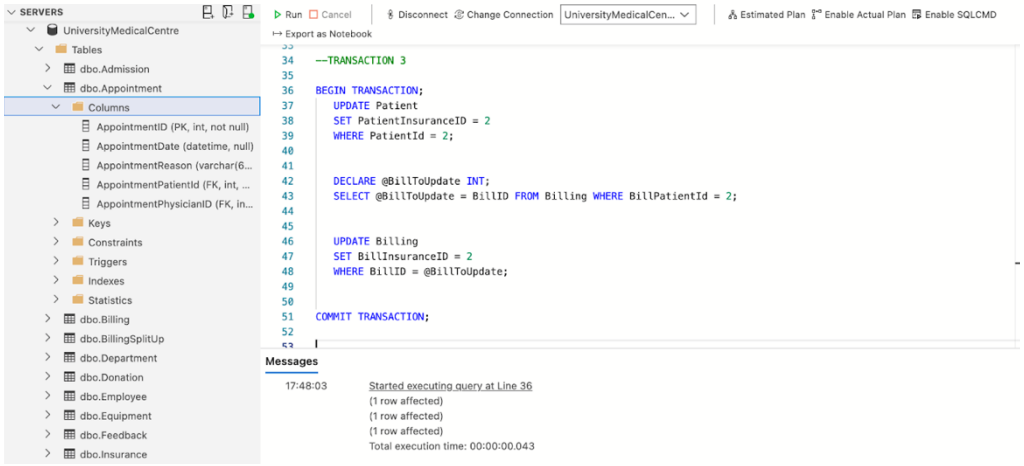
--Testing Transaction 1



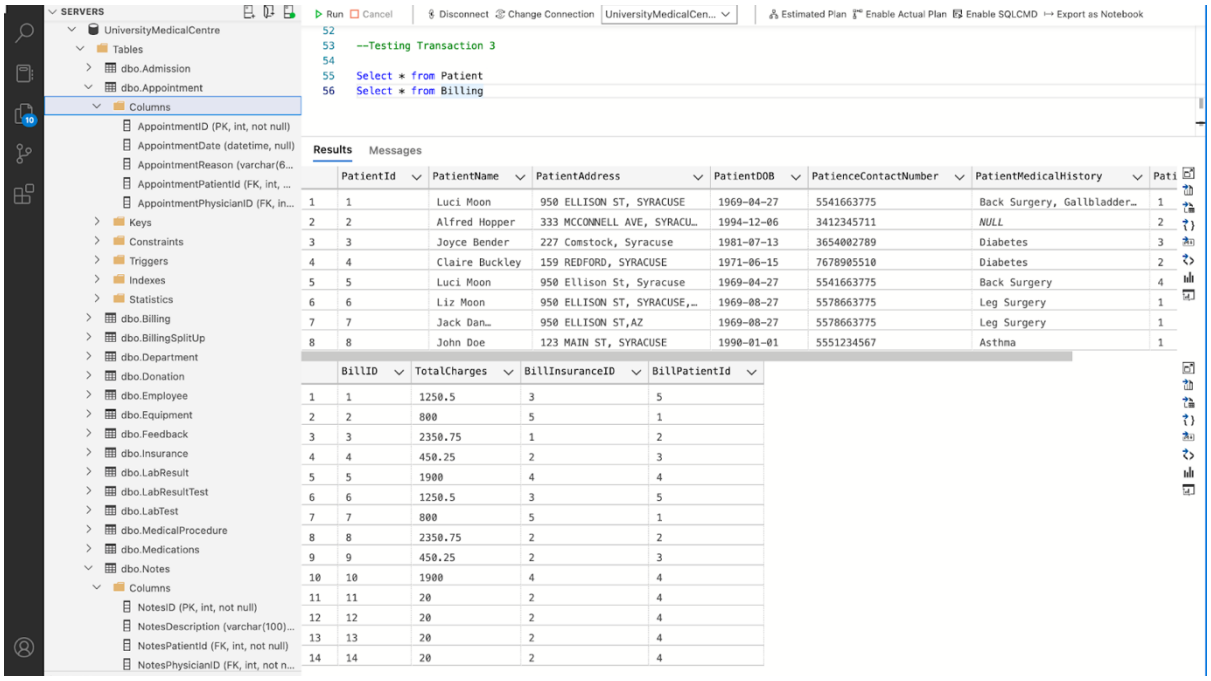
Transaction 3: Update Patient Insurance and Adjust Billing

This transaction updates a patient's insurance information and adjusts their billing record accordingly. It first updates the patient's insurance ID and then finds the corresponding billing record using the patient ID. The billing record's insurance ID is then updated, and the transaction is committed.

```
BEGIN TRANSACTION;
   UPDATE Patient
   SET PatientInsuranceID = 2
   WHERE PatientId = 2;
   DECLARE @BillToUpdate INT;
   SELECT @BillToUpdate = BillID FROM Billing WHERE BillPatientId = 2;
   UPDATE Billing
   SET BillInsuranceID = 2
   WHERE BillID = @BillToUpdate;
COMMIT TRANSACTION;
```



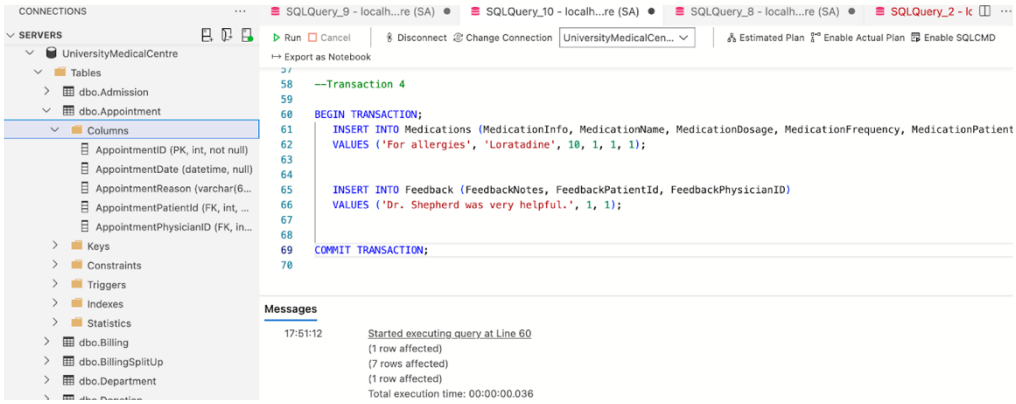

```
--Testing Transaction 3
Select * from Patient
Select * from Billing
```



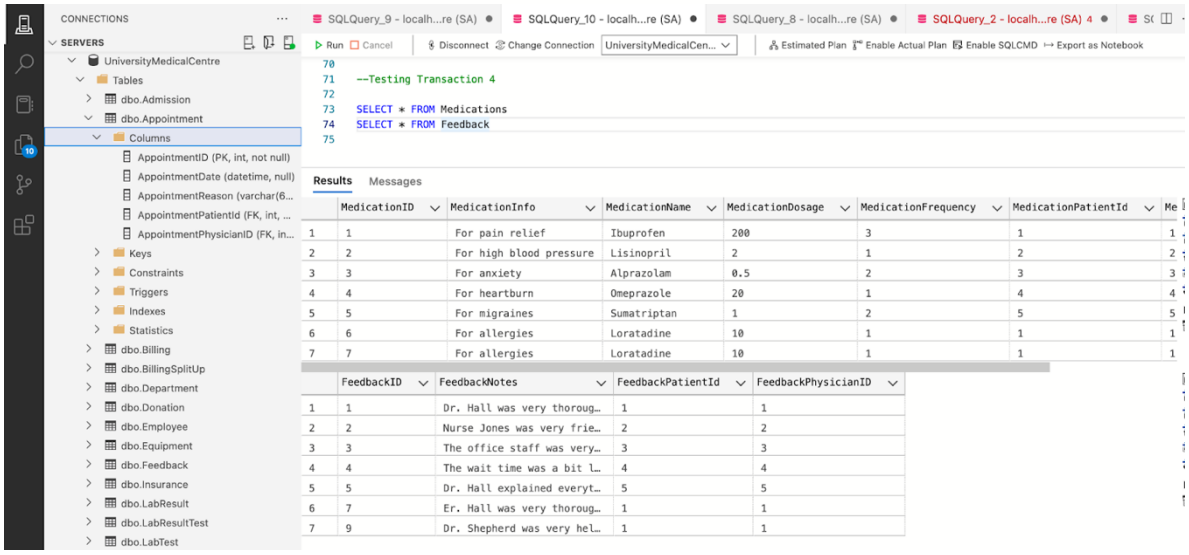
Transaction 4: Record Medication Prescription and Provide Feedback

This transaction inserts a medication prescription record for a patient and adds feedback related to the patient's experience. It starts by adding a new medication record with the patient ID, physician ID, and medication details. Next, it inserts a feedback record containing the patient's feedback notes and the involved physician's ID. Finally, the transaction is committed.

```
BEGIN TRANSACTION;
   INSERT INTO Medications (MedicationInfo, MedicationName, MedicationDosage,
MedicationFrequency, MedicationPatientId, MedicationPhysicianID)
   VALUES ('For allergies', 'Loratadine', 10, 1, 1, 1);

   INSERT INTO Feedback (FeedbackNotes, FeedbackPatientId, FeedbackPhysicianID)
   VALUES ('Dr. Shepherd was very helpful in explaining my allergy treatment.', 1,
1);

COMMIT TRANSACTION;
```




```
--Testing Transaction 4
SELECT * FROM Medications
SELECT * FROM Feedback
```

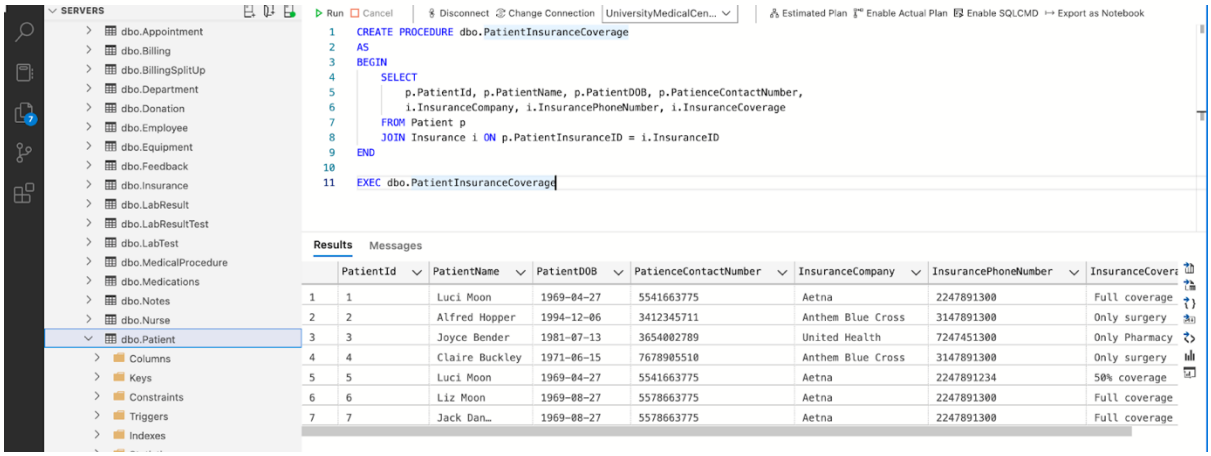


PROCEDURES and BUSINESS REPORTS:

PROCEDURE 1: Patient Insurance Coverage Report

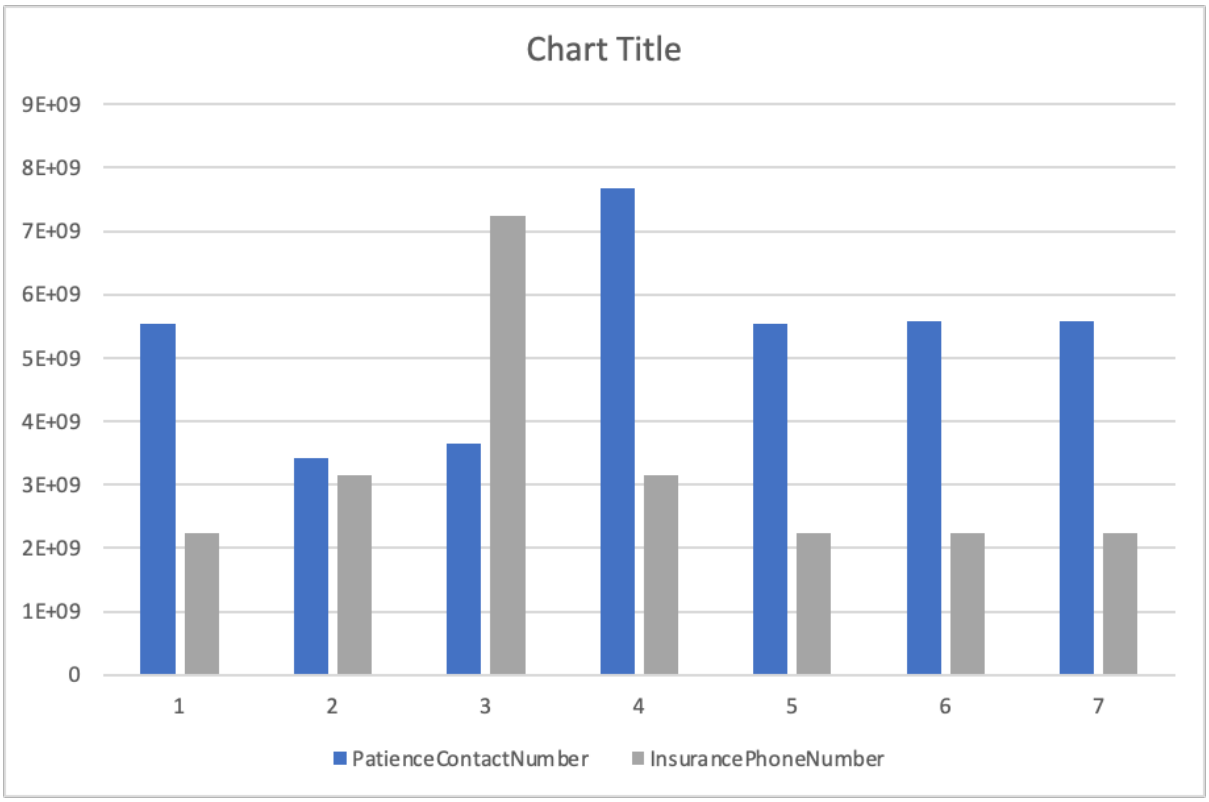
This stored procedure retrieves a list of patients and their insurance coverage details. It selects the patient's details, insurance company, insurance phone number, and insurance coverage by joining the Patient and Insurance tables.

```
CREATE PROCEDURE dbo.PatientInsuranceCoverage
AS
BEGIN
    SELECT
        p.PatientId, p.PatientName, p.PatientDOB, p.PatienceContactNumber,
        i.InsuranceCompany, i.InsurancePhoneNumber, i.InsuranceCoverage
    FROM Patient p
    JOIN Insurance i ON p.PatientInsuranceID = i.InsuranceID
END

EXEC dbo.PatientInsuranceCoverage
```



Visualization:

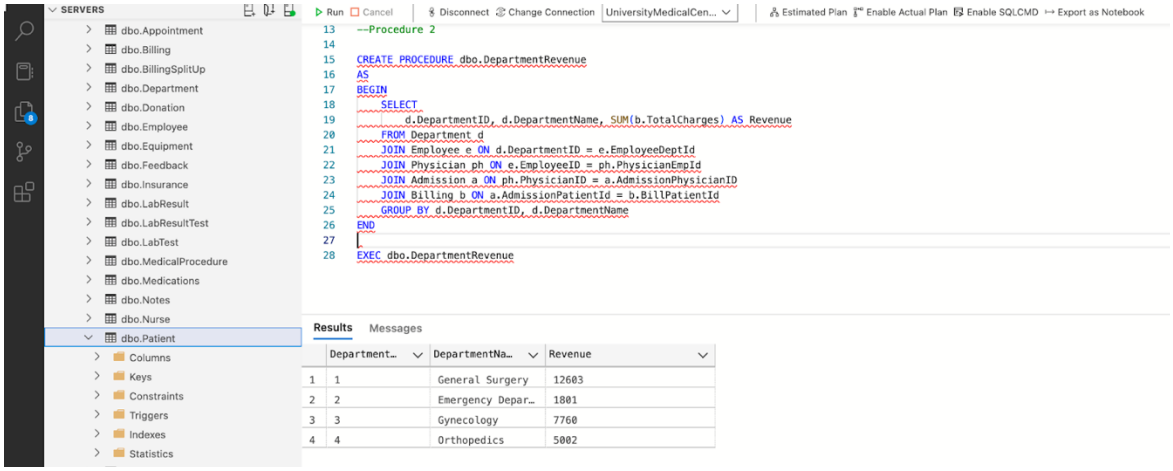
A representation of the patient's details, insurance company, insurance phone number, and insurance coverage. Each row represents a patient and their insurance information.



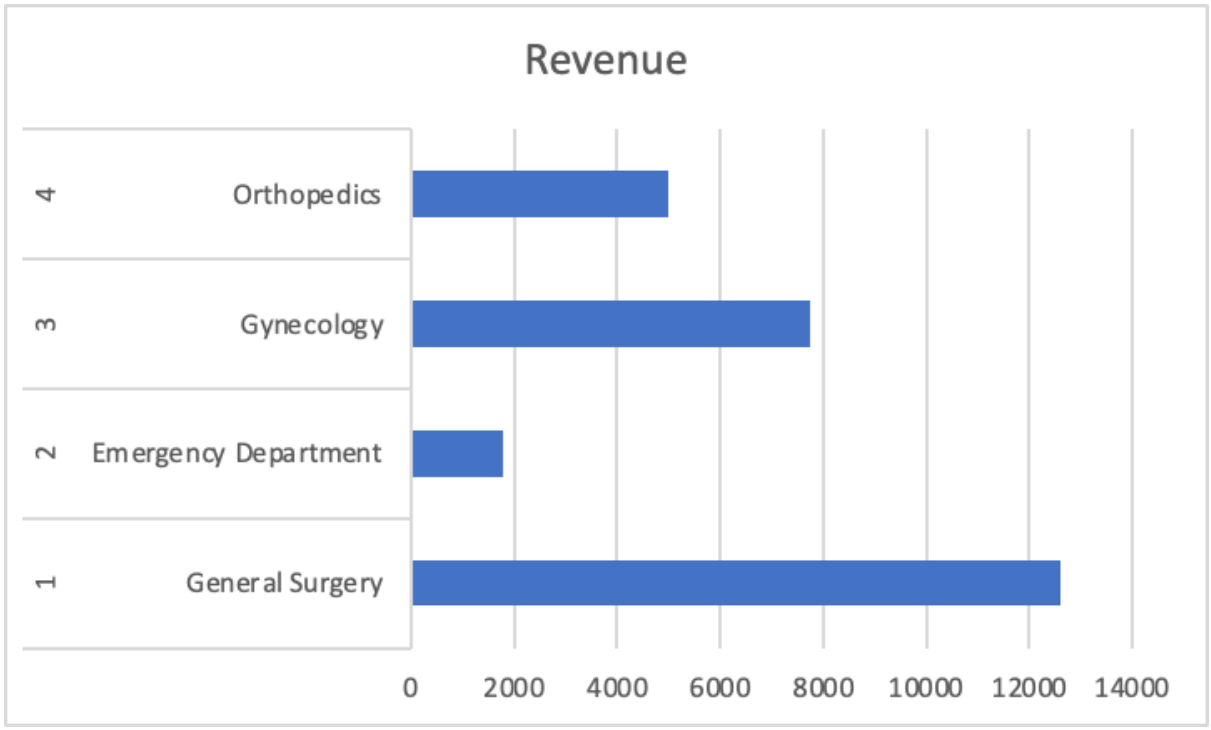
PROCEDURE 2: Department-wise Revenue Report

This stored procedure calculates the revenue generated by each department. It joins the Department, Employee, Physician, Admission, and Billing tables to sum up the total charges per department.

```
CREATE PROCEDURE dbo.DepartmentRevenue
AS
BEGIN
   SELECT
       d.DepartmentID, d.DepartmentName, SUM(b.TotalCharges) AS Revenue
   FROM Department d
   JOIN Employee e ON d.DepartmentID = e.EmployeeDeptId
   JOIN Physician ph ON e.EmployeeID = ph.PhysicianEmpId
   JOIN Admission a ON ph.PhysicianID = a.AdmissionPhysicianID
   JOIN Billing b ON a.AdmissionPatientId = b.BillPatientId
   GROUP BY d.DepartmentID, d.DepartmentName
END
```



Visualization 2:
A chart showing the revenue generated per department, with the department names on the X-axis and the revenue amounts on the Y-axis. Each bar represents a department and its revenue.

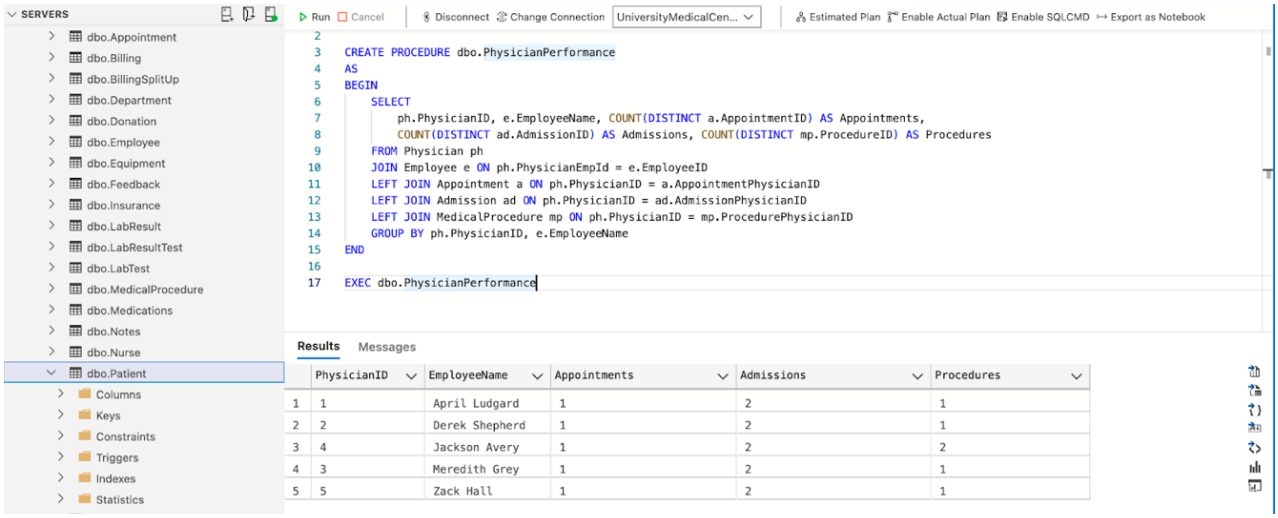


PROCEDURE 3: Physician Performance Report

This stored procedure lists the number of appointments, admissions, and procedures performed by each physician. It selects the physician's details and counts the number of appointments, admissions, and procedures by joining the Physician, Employee, Appointment, Admission, and MedicalProcedure tables.

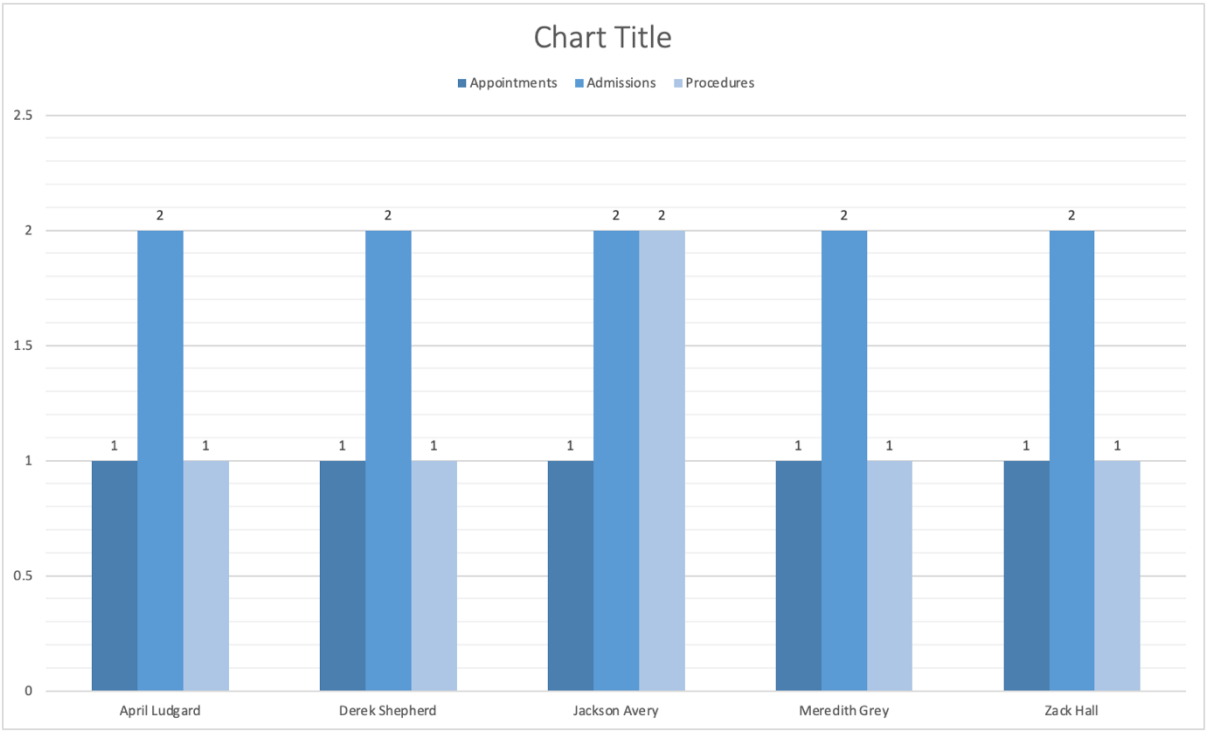
```
CREATE PROCEDURE dbo.PhysicianPerformance
AS
BEGIN
   SELECT
       ph.PhysicianID, e.EmployeeName, COUNT(DISTINCT a.AppointmentID) AS
Appointments,
       COUNT(DISTINCT ad.AdmissionID) AS Admissions, COUNT(DISTINCT mp.ProcedureID)
AS Procedures
   FROM Physician ph
   JOIN Employee e ON ph.PhysicianEmpId = e.EmployeeID
   LEFT JOIN Appointment a ON ph.PhysicianID = a.AppointmentPhysicianID
   LEFT JOIN Admission ad ON ph.PhysicianID = ad.AdmissionPhysicianID
   LEFT JOIN MedicalProcedure mp ON ph.PhysicianID = mp.ProcedurePhysicianID
   GROUP BY ph.PhysicianID, e.EmployeeName
END
```

```
EXEC dbo.PhysicianPerformance
```



Visualization 3:

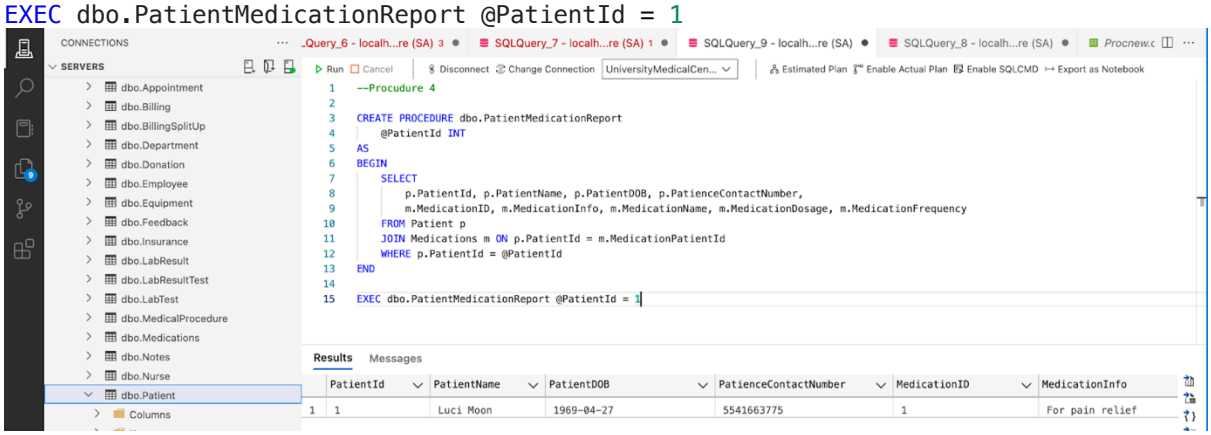
A chart showing the number of appointments, admissions, and procedures for each physician, with the physician names on the X-axis and the counts on the Y-axis. Each group of bars represents a physician and their performance metrics.



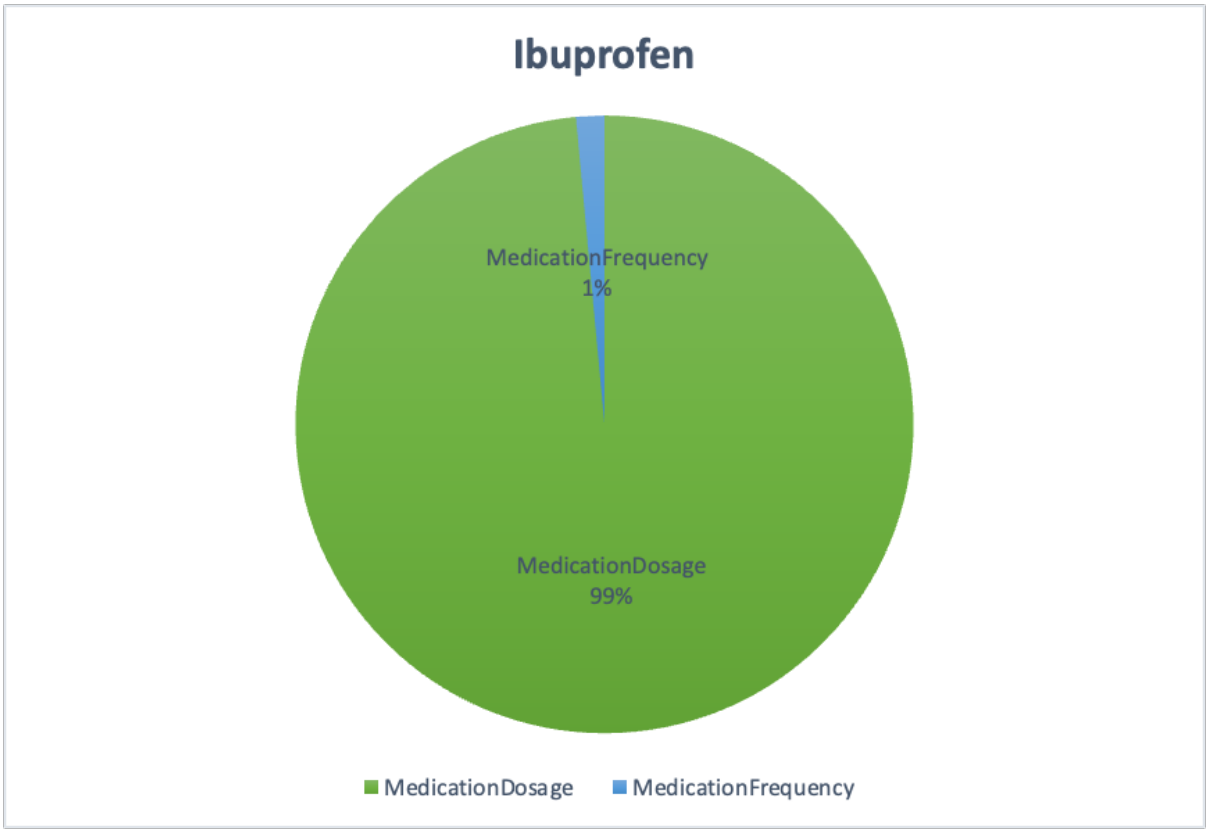
PROCEDURE 4: Patient Medication Report

This stored procedure lists all medications prescribed to a specific patient. It selects the patient's details and medication information by joining the Patient and Medications tables and filtering by patient ID.

```
CREATE PROCEDURE dbo.PatientMedicationReport
    @PatientId INT
AS
BEGIN
    SELECT
        p.PatientId, p.PatientName, p.PatientDOB, p.PatienceContactNumber,
        m.MedicationID, m.MedicationInfo, m.MedicationName, m.MedicationDosage,
m.MedicationFrequency
    FROM Patient p
    JOIN Medications m ON p.PatientId = m.MedicationPatientId
    WHERE p.PatientId = @PatientId
END
EXEC dbo.PatientMedicationReport @PatientId = 1
```



Visualization 4:
A representation of the patient's details and their prescribed medications. Each row represents a medication prescribed to the patient, along with the medication's details.

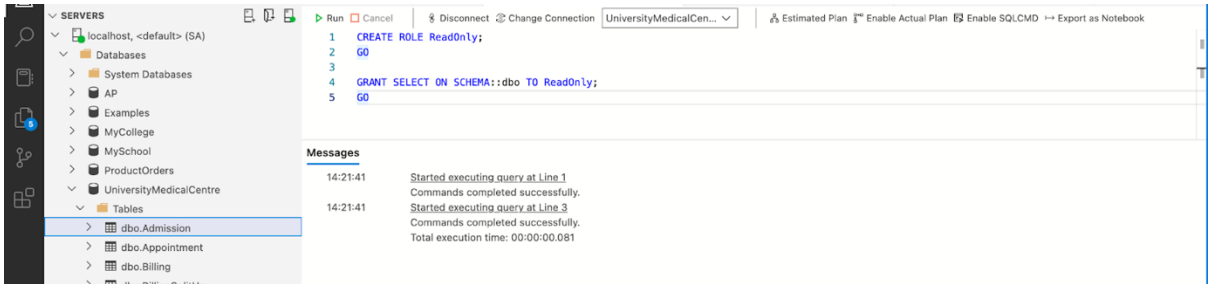


Users with various security levels, passwords, and roles:

1. ReadOnly User:

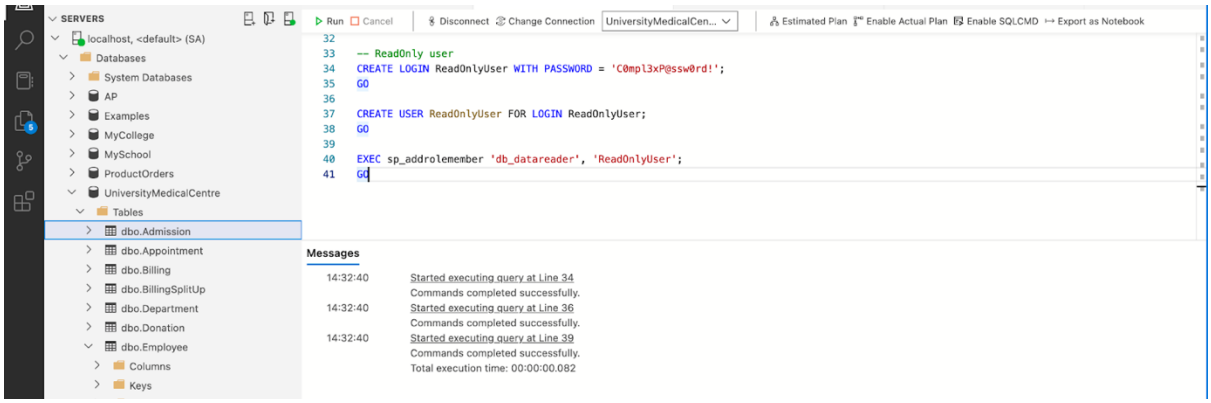
This user has read-only access to the data in the database. They can only execute SELECT statements to view the data but cannot modify it. The script creates the role 'ReadOnly' and grants SELECT permissions on the schema dbo. Then, it creates a login and user named 'ReadOnlyUser' and adds the user to the 'ReadOnly' role.

```
CREATE ROLE ReadOnly;
GO

GRANT SELECT ON SCHEMA::dbo TO ReadOnly;
GO
```



Assigning role:

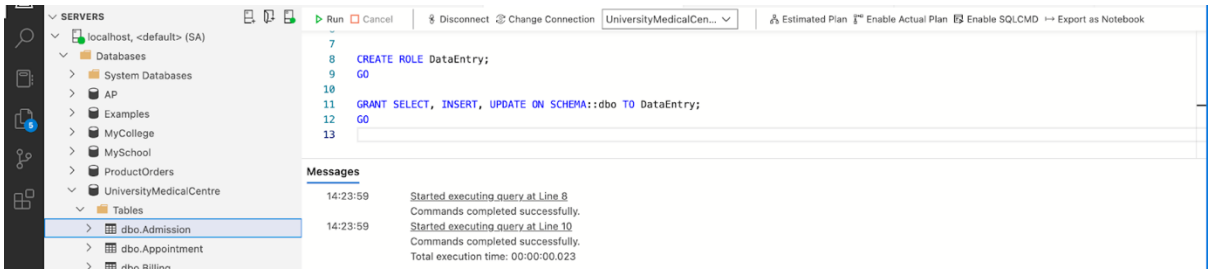
```
CREATE LOGIN ReadOnlyUser WITH PASSWORD = 'C0mpl3xP@ssw0rd!';
GO

CREATE USER ReadOnlyUser FOR LOGIN ReadOnlyUser;
GO

EXEC sp_addrolemember 'db_datareader', 'ReadOnlyUser';
GO
```



2. DataEntry User:

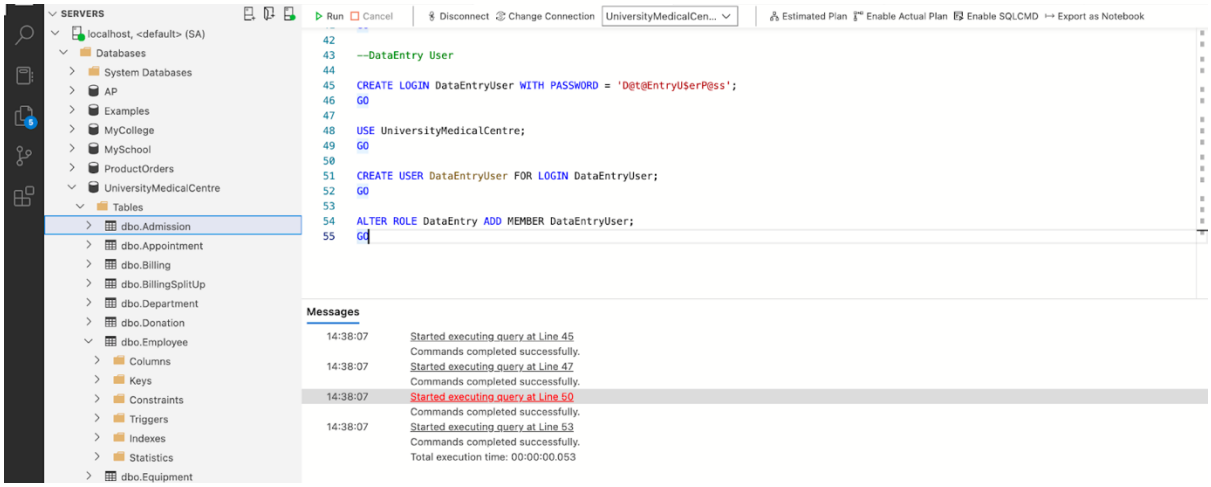
This user can view, insert, and update data in the database but cannot delete it. The script creates the role 'DataEntry' and grants SELECT, INSERT, and UPDATE permissions on the schema dbo. Then, it creates a login and user named 'DataEntryUser' and adds the user to the 'DataEntry' role.

```
USE UniversityMedicalCentre;
GO

CREATE ROLE DataEntry;
GO

GRANT SELECT, INSERT, UPDATE ON SCHEMA::dbo TO DataEntry;
GO
```



Assigning role:

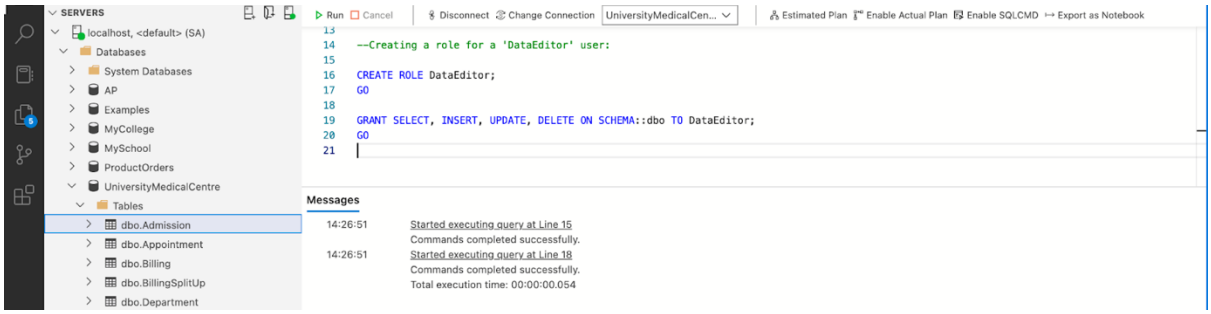
```
CREATE LOGIN DataEntryUser WITH PASSWORD = 'D@t@EntryU$erP@ss';
GO
USE UniversityMedicalCentre;
GO
CREATE USER DataEntryUser FOR LOGIN DataEntryUser;
GO
ALTER ROLE DataEntry ADD MEMBER DataEntryUser;
GO
```



3. DataEditor User:

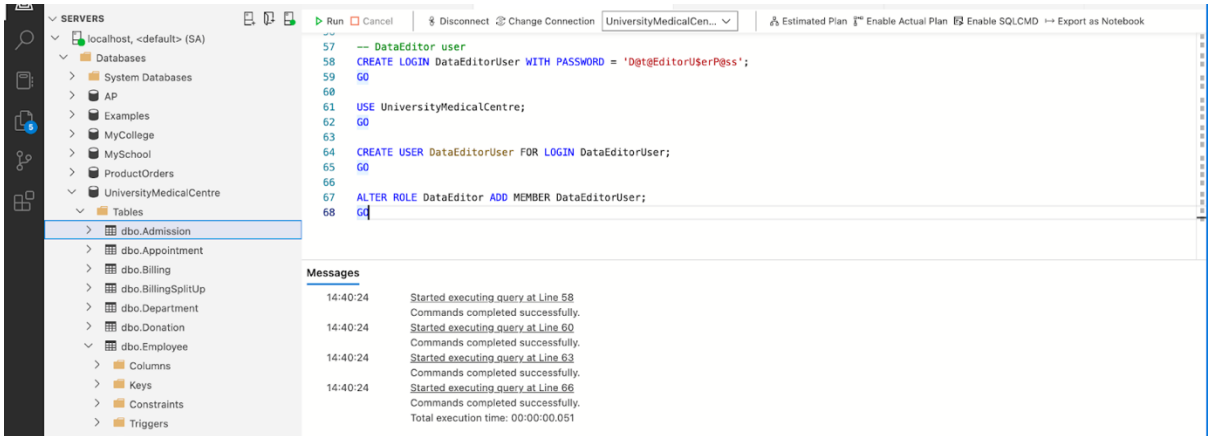
This user has full read and write access to the data in the database, including the ability to delete records. The script creates the role 'DataEditor' and grants SELECT, INSERT, UPDATE, and DELETE permissions on the schema dbo. Then, it creates a login and user named 'DataEditorUser' and adds the user to the 'DataEditor' role.

```
USE UniversityMedicalCentre;
GO
CREATE ROLE DataEditor;
GO
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO DataEditor;
GO
```



Assigning role:

```
CREATE LOGIN DataEditorUser WITH PASSWORD = 'D@t@EditorU$erP@ss';
GO
USE UniversityMedicalCentre;
GO
CREATE USER DataEditorUser FOR LOGIN DataEditorUser;
GO
ALTER ROLE DataEditor ADD MEMBER DataEditorUser;
GO
```



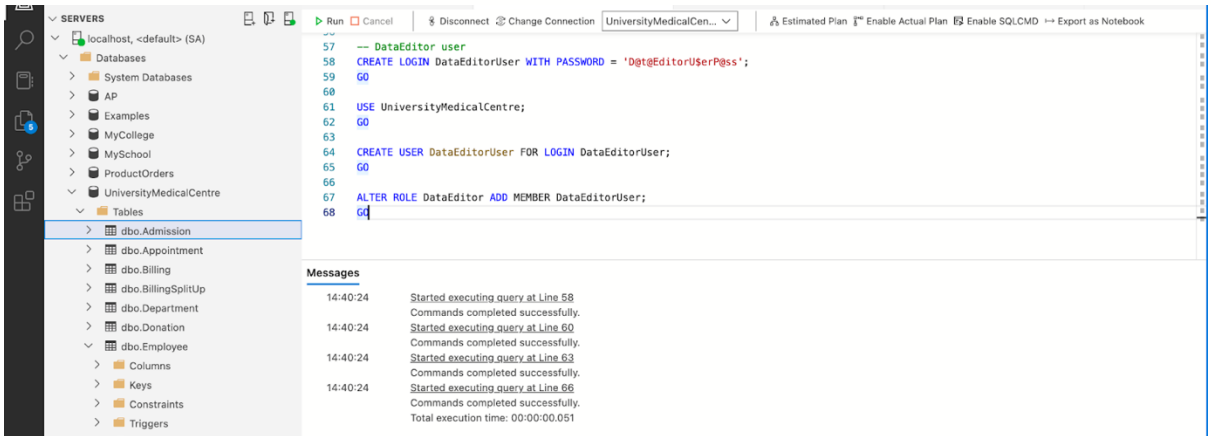
4. DatabaseAdmin User:

This user has full control over the database, including schema modification, user management, and other administrative tasks. The script creates the role 'DatabaseAdmin' and grants CONTROL permissions on the entire database. Then, it creates a login and user named 'DatabaseAdminUser' and adds the user to the 'DatabaseAdmin' role. ‘

```
USE UniversityMedicalCentre;
GO
CREATE ROLE DatabaseAdmin;
GO
GRANT CONTROL ON DATABASE::UniversityMedicalCentre TO DatabaseAdmin;
GO
```

Assigning role:

```
CREATE LOGIN DatabaseAdminUser WITH PASSWORD = 'DB@dm!nU$erP@ss';
GO
USE UniversityMedicalCentre;
GO
CREATE USER DatabaseAdminUser FOR LOGIN DatabaseAdminUser;
GO
ALTER ROLE DatabaseAdmin ADD MEMBER DatabaseAdminUser;
GO
```



C. Conclusion

In conclusion, the database system meticulously designed and implemented for the Patient Access and Registration branch of the University Medical Center stands as a testament to the power of innovation and technology in revolutionizing patient care. By capturing and organizing vital patient information in a structured and efficient manner, this database facilitates seamless coordination among healthcare providers, ultimately contributing to better patient outcomes and experiences.

The project's success in addressing complex healthcare challenges demonstrates the potential for further advancements in the field. As healthcare facilities continue to evolve, the integration of advanced data analytics, artificial intelligence, and machine learning technologies could elevate this database system to new heights, providing healthcare providers with even more profound insights into patient care patterns and trends.

Additionally, the database system's adaptability and scalability make it a promising foundation for future improvements and extensions. With the potential to be integrated with other departments within the medical center, the system can contribute to a more unified and streamlined healthcare experience for patients, providers, and administrators alike.

Finally, this project serves as an inspiring example of how technology can be harnessed to address some of the most pressing challenges in healthcare today. By continually seeking innovative solutions and embracing the power of technology, we can collectively work towards a future where high-quality patient care is accessible, efficient, and sustainable for all.