

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

B. Implementation and Testing

1. Create Database
2. Create Tables
3. Insert
4. Views
5. Triggers
6. Functions
7. Transactions
8. Procedures and Business Reports
9. Users with various security levels, passwords, and roles:

C. Conclusions

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.

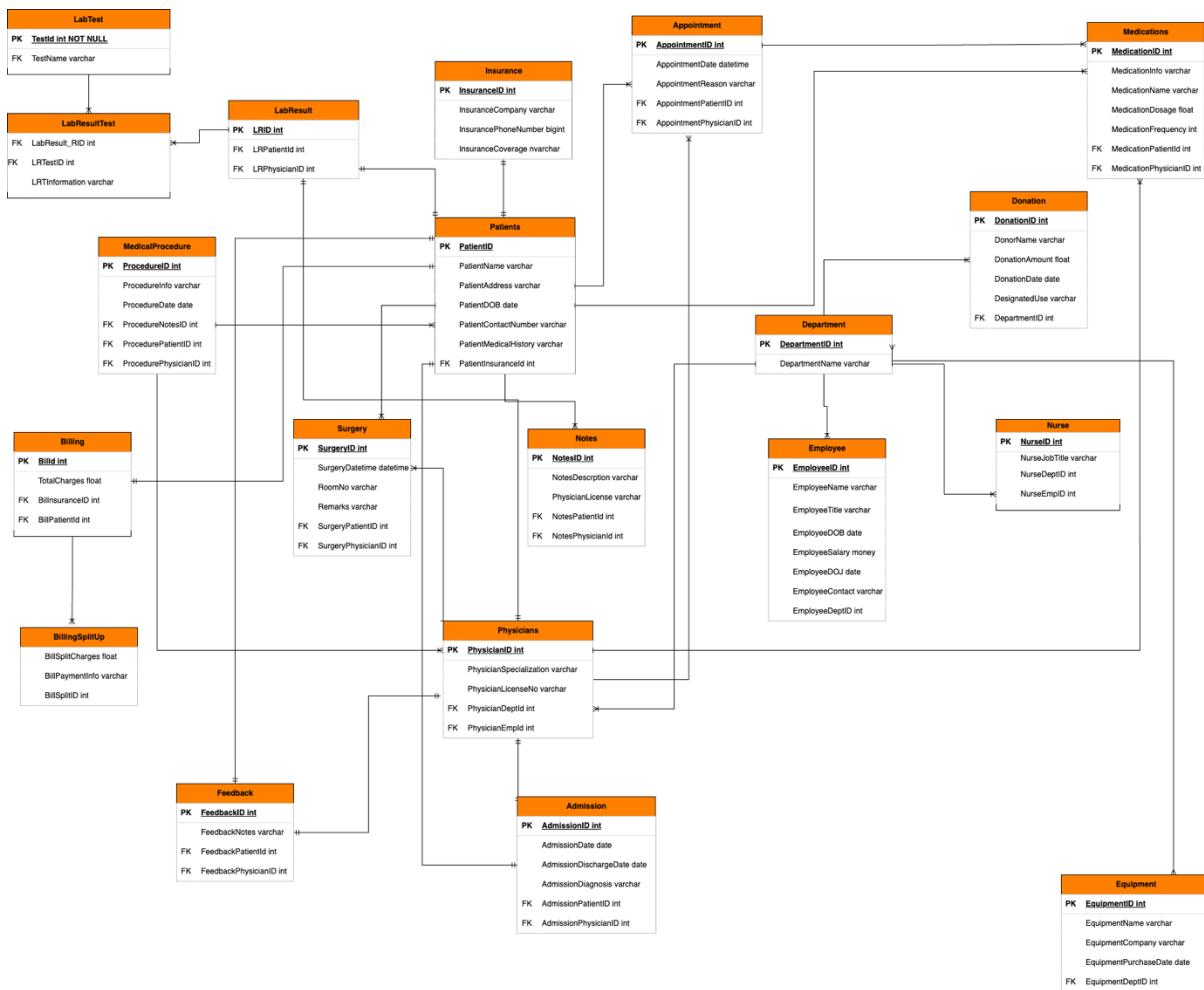


Table Name	Description
Insurance	Contains information about insurance companies, including their name, phone number, and coverage type.
Patient	Stores patient information, including name, address, date of birth, contact number, medical history, and insurance details.
Department	Contains information about 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
```



The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer tree shows the 'Databases' node under 'localhost, <default> (SA)'. Inside 'Databases', several databases are listed: System Databases, AP, Examples, MyCollege, MySchool, ProductOrders, and UniversityMedicalCentre. On the right, the 'Messages' pane displays the execution of the 'CREATE DATABASE' command. The message log shows: 'Started executing query at Line 1', 'Commands completed successfully.', and 'Total execution time: 00:00:00.497'. The timestamp for the messages is 12:39:03.

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),
PhysicianLicenseNo 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),
LRTTestID 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 screenshot shows the SSMS interface with the following details:

- Connections:** localhost, <default> (SA)
- Servers:** localhost, <default> (SA) -> Databases
- Tables:** Under UniversityMedicalCentre, the following tables are listed:
 - dbo.Admission
 - dbo.Appointment
 - dbo.Billing
 - dbo.BillingSplitUp
 - dbo.Department
 - dbo.Donation
 - dbo.Employee
 - dbo.Equipment
 - dbo.Feedback
 - dbo.Insurance
 - dbo.LabResult
 - dbo.LabResultTest
 - dbo.LabTest
 - dbo.MedicalProcedure
 - dbo.Medications
 - dbo.Notes
 - dbo.Nurse
 - dbo.Patient
 - dbo.Physician
 - dbo.Surgery
 - Views
 - Synonyms
 - Programmability
 - External Resources

SQL Query Editor:

```

CREATE database UniversityMedicalCentre;
USE UniversityMedicalCentre
GO
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,
    PatientContactNumber VARCHAR(50),
    PatientMedicalHistory VARCHAR(50),
    PatientInsuranceID INT NULL REFERENCES Insurance(InsuranceID),
);

```

Messages:

```

18:41:07 Started executing query at Line 3
Commands completed successfully.
18:41:07 Started executing query at Line 5
Commands completed successfully.
Total execution time: 00:00:00.122

```

Status Bar: Ln 15, Col 2 | Spaces: 4 | UTF-8 | LF | 0 rows | MSSQL | 00:00:00 | localhost : UniversityMedicalCentre

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);
```

```
CONNECTIONS ... Welcome SQLQuery_1 - localhost...re (SA) 1 SQLQuery_2 - localhost...re (SA) 3
... Run Cancel Disconnect Change Connection UniversityMedicalCen...
1 use UniversityMedicalCentre
2
3 INSERT INTO Insurance
4 VALUES ('Aetna', 2247891300, 'Full coverage'),
5 ('Anthem Blue Cross', 3147891300, 'Only surgery'),
6 ('United Health', 7247451300, 'Only Pharmacy'),
7 ('Aetna', 2247891234, '50% coverage'),
8 ('United Max', 2247891322, 'Full coverage');
9
10
11 INSERT INTO Patient
12 VALUES ('Luci Moon', '950 Ellison St, Syracuse', '1969-04-27', '5541663775', 'Back Surgery', 1),
13 ('Alfre Hopper', '333 McConnell Ave, Syracuse', '1994-12-06', '3412345711', null, 2),
14 ('Joyce Bender', '227 Comstock, Syracuse', '1981-07-13', '3654002789', 'Diabetes', 3),
15 ('Claire Buckley', '159 Redford, Syracuse', '1971-06-15', '7678905510', 'Diabetes', 2),
16 ('Luci Moon', '950 Ellison St, Syracuse', '1969-04-27', '5541663775', 'Back Surgery', 4)
17
18 select * from Department
19
```

Messages

18:44:28 Started executing query at Line 1
(5 rows affected)
(5 rows affected)
Total execution time: 00:00:00.020

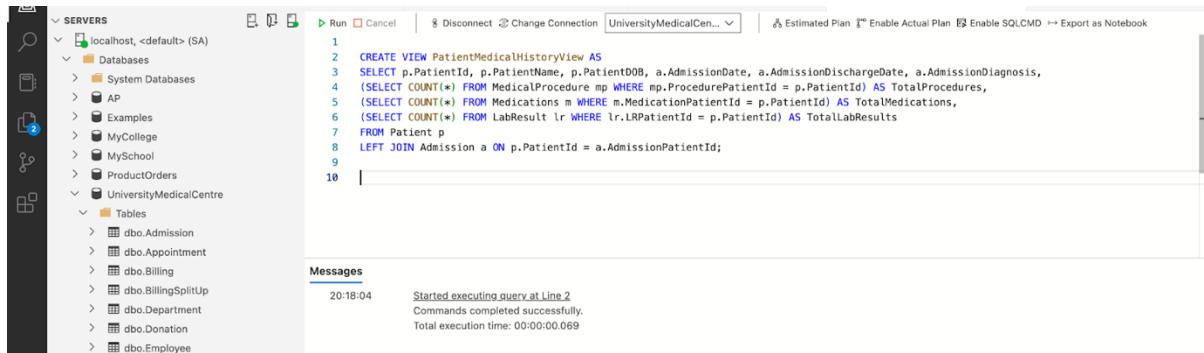
The provided SQL statements create and populate tables for the University Medical Centre database, covering different aspects of the center's operations like patient information, insurance, departments, employees, appointments, and more. The INSERT statements add sample data to the tables, allowing for 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:

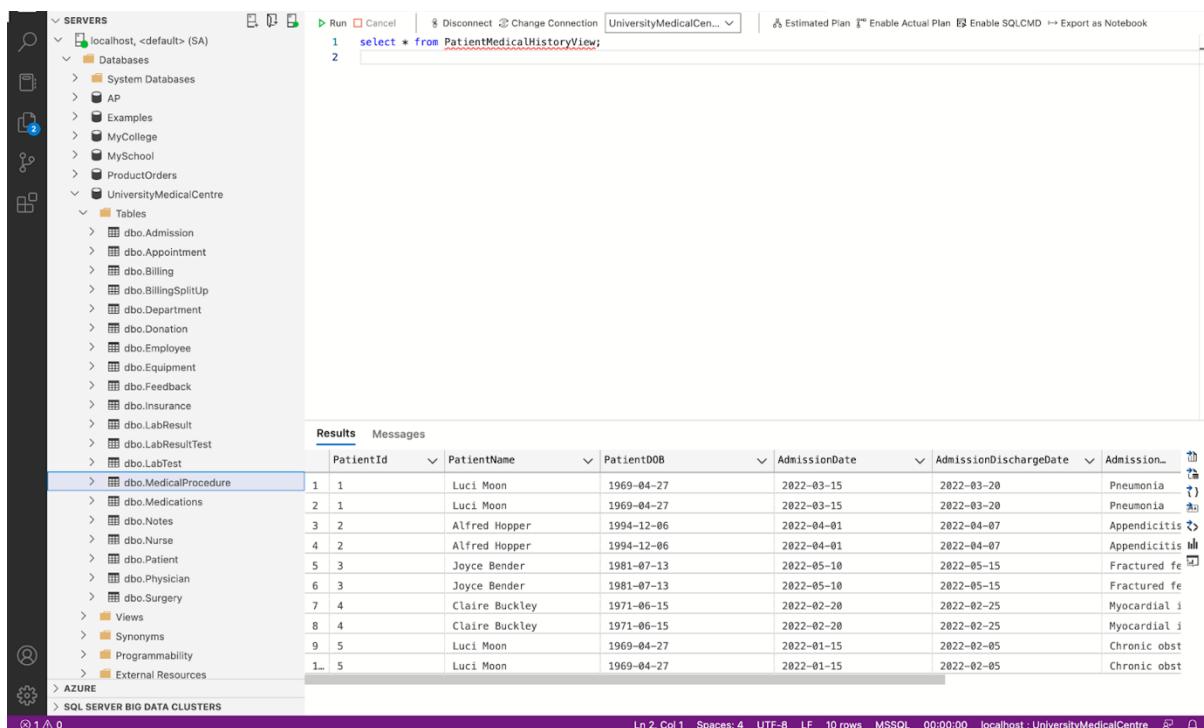
View 1: Patient Medical History Summary

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;
```

The screenshot shows the Object Explorer on the left with the 'UniversityMedicalCentre' database selected. The 'Views' node under 'UniversityMedicalCentre' is expanded, showing the newly created 'PatientDays' view. The 'Messages' pane at the bottom right shows the execution results: 'Started executing query at Line 18', 'Commands completed successfully.', and 'Total execution time: 00:00:00.118'.

```
Select * from PatientDays;
```

The screenshot shows the Object Explorer on the left with the 'UniversityMedicalCentre' database selected. The 'Views' node under 'UniversityMedicalCentre' is expanded, showing the 'PatientDays' view. The 'Results' pane at the bottom right displays the output of the query, showing the patient names and their stay durations:

	Name	NumberOfDays
1	Luci Moon	5
2	Alfred Hopp...	6
3	Joyce Bender	5
4	Claire Buck...	5
5	Luci Moon	21
6	Luci Moon	5
7	Alfred Hopp...	6
8	Joyce Bender	5
9	Claire Buck...	5
10	Luci Moon	21

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
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, under 'localhost, <default> (SA) / SERVERS / UniversityMedicalCentre', the 'Views' node is expanded, showing 'Top3PaidEmployees'. In the main results pane, the T-SQL code for creating the view is displayed. The 'Messages' pane at the bottom shows the execution log: 'Started executing query at Line 32', 'Commands completed successfully.', and 'Total execution time: 00:00:00.059'.

```
Select * from Top3PaidEmployees
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, under 'localhost, <default> (SA) / SERVERS / UniversityMedicalCentre', the 'Views' node is expanded, showing 'Top3PaidEmployees'. In the main results pane, the T-SQL code for selecting from the view is displayed. Below the results pane, the 'Results' tab is selected, showing a table with three rows of data. The table has columns 'Name' and 'Salary'. The data is as follows:

Name	Salary
Zack Hall	880000.00
April Ludgate	670000.00
Jackson Ave.	40000.00

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;
```

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure under 'UniversityMedicalCentre'. The 'Tables' node is expanded, showing various tables like 'Admission', 'Appointment', 'Billing', etc., with 'dbo.Employee' selected. The main pane contains the T-SQL code for creating the 'departmentalcounts' view. The code includes a SELECT statement with COUNT(DISTINCT) functions for PhysicianID, NurseID, EmployeeID, and EquipmentID, grouped by DepartmentName. Below the code, the 'Messages' pane shows the execution log: 'Started executing query at Line 6', 'Commands completed successfully.', and 'Total execution time: 00:00:00.109'.

```
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
```

```
SELECT * FROM departmentalcounts
```

The screenshot shows the execution of the 'departmentalcounts' view. The 'Results' pane at the bottom displays the output as a table with columns: Department_, physician_count, nurse_count, employee_count, and equipment_count. The data shows five departments: Burn Care, Emergency Dep., General Surge., Gynecology, and Orthopedics, each with their respective counts of physicians, nurses, employees, and equipment.

Department_	physician_count	nurse_count	employee_count	equipment_count
Burn Care	1	1	0	1
Emergency Dep.	2	1	3	1
General Surge.	1	1	2	2
Gynecology	1	0	1	1
Orthopedics	0	0	1	1

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)
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, the 'Servers' node is expanded, showing 'localhost, <default> (SA)'. Under this, 'Databases' is selected, showing 'System Databases', 'AP', 'Examples', 'MyCollege', 'MySchool', 'ProductOrders', and 'UniversityMedicalCentre'. The 'UniversityMedicalCentre' database is currently selected. In the center pane, there is a 'Scripting' window with the following T-SQL code:

```
-- TRIGGER
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
```

Below the script window, the 'Messages' pane displays the execution results:

```
20:34:14 Started executing query at Line 12
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.071
```

```
Select * from Patient
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' pane active. The previous T-SQL script is still visible in the center pane. The results of the 'Select * from Patient' query are displayed in a grid:

PatientId	PatientName	PatientAddress	PatientDOB	PatienceContactNumber	PatientMedicalHistory
1	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery
2	Alfred Hopper	333 McConnell Ave, Syracuse	1994-12-06	3412345711	NULL
3	Joyce Bender	227 Comstock, Syracuse	1981-07-13	3654002789	Diabetes
4	Claire Buckley	159 Redford, Syracuse	1971-06-15	7678905510	Diabetes
5	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery
6	Liz Moon	950 ELLISON ST, SYRACUSE,NY	1969-08-27	5578663775	Leg Surgery

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)
```

The screenshot shows the SSMS interface with the Object Explorer on the left and the Query Editor on the right. In the Object Explorer, the 'localhost, <default> (SA)' server node is expanded, showing databases like 'System Databases', 'AP', 'Examples', 'MySchool', 'ProductOrders', and 'UniversityMedicalCentre'. The 'Tables' node under 'UniversityMedicalCentre' is also expanded. In the Query Editor, the code for the trigger and a test insert statement are written. The message pane at the bottom shows the execution status: 'Started executing query at Line 22', 'Commands completed successfully.', and 'Total execution time: 00:00:00.081'.

```
select * from Patient;
```

The screenshot shows the SSMS interface with the Object Explorer on the left and the Results pane on the right. The results of the 'select * from Patient' query are displayed in a table. The columns are PatientId, PatientName, PatientAddress, PatientDOB, PatienceContactNumber, PatientMedicalHistory, and PatientStatus. The data shows seven patients with their respective details.

	PatientId	PatientName	PatientAddress	PatientDOB	PatienceContactNumber	PatientMedicalHistory	PatientStatus
1	1	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery	1
2	2	Alfred Hopper	333 McConnell Ave, Syracuse	1994-12-06	3412345711	NULL	2
3	3	Joyce Bender	227 Constock, Syracuse	1981-07-13	3654002789	Diabetes	3
4	4	Claire Buckley	159 Redford, Syracuse	1971-06-15	7678905510	Diabetes	2
5	5	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery	4
6	6	Liz Moon	950 ELLISON ST, SYRACUSE, NY	1969-08-27	5578663775	Leg Surgery	1
7	7	Jack Dan	950 ELLISON ST, AZ	1969-08-27	5578663775	Leg Surgery	1

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 SetDefaultEquipmentName
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)

```

The screenshot shows the Object Explorer on the left with the path: Examples > MyCollege > MySchool > ProductOrders > UniversityMedicalCentre > Tables > dbo.Employee. The script pane on the right contains the trigger creation script. The messages pane at the bottom shows the command was started at 22:17:40 and completed successfully.

```

9    create trigger SetDefaultEquipmentName
10   on Equipment instead of insert
11   as begin
12       Declare @EquipmentCompany varchar(60)
13       Declare @EquipmentName varchar(60)
14       Declare @EquipmentPurchaseDate Date
15       Declare @EquipmentDeptID int
16       select @EquipmentName = EquipmentName, @EquipmentCompany = EquipmentCompany,
17             @EquipmentPurchaseDate = EquipmentPurchaseDate,
18             @EquipmentDeptID = EquipmentDeptID
19       from inserted;
20       IF @EquipmentCompany = '' set @EquipmentCompany = @EquipmentName + ' :Generic';
21       Insert into Equipment(EquipmentName, EquipmentCompany, EquipmentPurchaseDate,
22                           EquipmentDeptID)
23       values(@EquipmentName, @EquipmentCompany, @EquipmentPurchaseDate, @EquipmentDeptID)
24   end;
25
26   INSERT into Equipment values
27   ('Stethoscope','','2022-03-20',1)
28
29   select * from Equipment
30

```

Messages

22:17:40 Started executing auen.dbo.[Line 10]

Commands completed successfully.

Total execution time: 00:00:00.039

```
Select * from Equipment
```

The screenshot shows the Object Explorer on the left with the path: Examples > MyCollege > MySchool > ProductOrders > UniversityMedicalCentre > Tables > dbo.Employee. The script pane on the right contains the trigger creation script. The results pane at the bottom shows the output of the SELECT statement.

```

4   alter table Equipment
5   alter column EquipmentCompany varchar(60) null
6
7   create trigger SetDefaultEquipmentName
8   on Equipment instead of insert
9   as begin
10      Declare @EquipmentCompany varchar(60)
11      Declare @EquipmentName varchar(60)
12      Declare @EquipmentPurchaseDate Date
13      Declare @EquipmentDeptID int
14      select @EquipmentName = EquipmentName, @EquipmentCompany = EquipmentCompany,
15            @EquipmentPurchaseDate = EquipmentPurchaseDate,
16            @EquipmentDeptID = EquipmentDeptID
17      from inserted;
18      IF @EquipmentCompany = '' set @EquipmentCompany = @EquipmentName + ' :Generic';
19      Insert into Equipment(EquipmentName, EquipmentCompany, EquipmentPurchaseDate,
20                           EquipmentDeptID)
21      values(@EquipmentName, @EquipmentCompany, @EquipmentPurchaseDate, @EquipmentDeptID)
22
23   INSERT into Equipment values
24   ('Stethoscope','','2022-03-20',1)
25
26   select * from Equipment
27

```

EquipmentID	EquipmentName	EquipmentCompany	EquipmentPurchaseDate	EquipmentDeptID
1	Stethoscope	Medtronic	2022-03-20	1
2	Sphygmomanometer	Johnson & Johnson	2022-03-20	2
3	Stretcher	Abbott Laboratories	2021-03-18	3
4	Scalpel	Abbott Laboratories	2021-09-10	4
5	Syringe	Cardinal Health	2020-09-20	5
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)
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, under 'UniversityMedicalCentre' > 'Tables' > 'dbo.Employee', the 'FeedbackNotes_Insert_Update' trigger is selected. In the main pane, the trigger definition is displayed. Below it, the test insert query is shown. The 'Messages' pane at the bottom shows the execution results: 'Started executing query at Line 78', '(6 rows affected)', '(1 row affected)', and 'Total execution time: 00:00:00.048'.

```
select * from Feedback
```

The screenshot shows the SQL Server Management Studio interface. The same trigger and test insert are visible. Below them, the results of the 'select * from Feedback' query are shown in a grid. The data consists of six rows, each with a FeedbackID (1-6), a FeedbackNotes entry (all starting with 'Er. Hall was very thorough during my exam.'), and FeedbackPatientId and FeedbackPhysicianId both set to 1.

FeedbackID	FeedbackNotes	FeedbackPatientId	FeedbackPhysicianID
1	Dr. Hall was very thorough during my exam.	1	1
2	Nurse Jones was very friendly.	2	2
3	The office staff was very helpful.	3	3
4	The wait time was a bit longer than expected.	4	4
5	Dr. Hall explained everything clearly.	5	5
6	Er. Hall was very thorough during my exam.	1	1

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
);
```

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure, including servers, databases, tables, and functions. In the center, the Query Editor window contains the T-SQL code for creating the function. The code defines a function named fnBillingRange that takes two decimal parameters (@MinCharges and @MaxCharges) and returns a table of billing records where the total charges fall within that range. Below the code, a message indicates the query was started at 23:14:14 and completed successfully in 0:00:00.044. At the bottom of the window, there is a 'Messages' section.

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

The screenshot shows the SQL Server Management Studio interface. The Query Editor window contains the same T-SQL code as the previous screenshot, but it is now followed by a single command: 'select * from fnBillingRange(500, 2000)'. The results pane on the right displays the output of this query, which is a table with four columns: BillID, BillInsuranceID, BillPatientId, and TotalCharges. The data shows six rows of billing records that meet the specified criteria.

BillID	BillInsuranceID	BillPatientId	TotalCharges
1	3	5	1250.5
2	5	1	800
3	5	4	1900
4	3	5	1250.5
5	5	1	800
6	4	4	1900

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
);
```

The screenshot shows the Object Explorer on the left with the 'UniversityMedicalCentre' database selected. Under the 'Tables' node, 'Employee' is currently selected. On the right, the 'Scripting' tab is open, displaying the T-SQL code for creating the function. The code is identical to the one provided above. Below the code, the 'Messages' pane shows the execution log: 'Started executing query at Line 21', 'Commands completed successfully.', and 'Total execution time: 00:00:00.026'.

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

The screenshot shows the same environment as the previous one. The 'Employee' table is still selected in the Object Explorer. The 'Results' pane on the right displays the output of the executed query. The result set contains a single row with the following data:

MedicationID	MedicationInfo	MedicationName	MedicationDosage	MedicationFrequency	ProcedureID	ProcedureInfo
1	For pain relief	Ibuprofen	200	3	7	Appendectomy

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.LRTesID = lt.TestID
    WHERE lr.LRPatientID = @patientID
);

```

The screenshot shows the SSMS interface with the 'UniversityMedicalCentre' database selected. The 'Tables' node under 'dbo' is expanded, and the 'Employee' table is selected. The main query editor window contains the T-SQL code for creating the function. The 'Messages' pane at the bottom right shows the execution log: 'Started executing query at Line 49', 'Commands completed successfully.', and 'Total execution time: 00:00:00.049'.

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

The screenshot shows the SSMS interface with the 'UniversityMedicalCentre' database selected. The 'Tables' node under 'dbo' is expanded, and the 'Employee' table is selected. The main query editor window contains the T-SQL code for creating the function followed by a select statement. The 'Results' pane at the bottom shows the output of the query, which is a single row:

LRID	LRPatientId	TestName	LRTInformation
1	2	Basic metabolic panel	Positive

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
);
```

The screenshot shows the SSMS interface with the following details:

- Object Explorer:** Shows the database structure under 'UniversityMedicalCentre'. The 'dbo.Employee' table is selected.
- Query Editor:** Contains the T-SQL code for creating the function.
- Messages Pane:** Displays the following output:


```
23:21:30 Started executing query at Line 65
Commands completed successfully.
Total execution time: 00:00:00.028
```

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

The screenshot shows the SSMS interface with the following details:

- Object Explorer:** Shows the database structure under 'UniversityMedicalCentre'. The 'dbo.Employee' table is selected.
- Query Editor:** Contains the T-SQL code for calling the function with a parameter value of 1.
- Results Pane:** Displays the output of the query:

	AppointmentID	AppointmentDate	AppointmentReason	PatientName	EmployeeName
1	3	2023-05-25 10:00:00.000	Follow-up for femur injury	Luci Moon	Derek Shepherd

Transactions:

Transaction 1: Add New Patient and Appointment

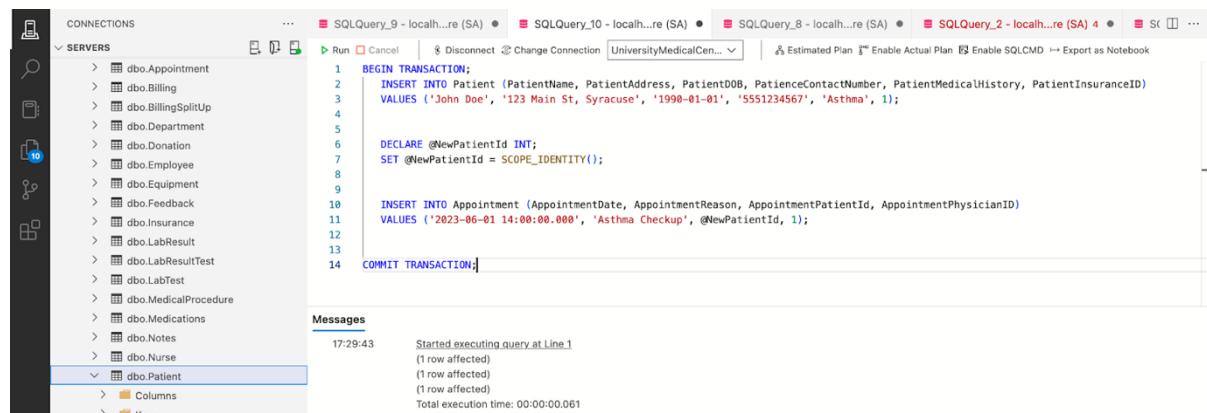
This transaction inserts a new patient record and creates an appointment for that patient. It starts by adding a patient with their details and then uses the newly created patient ID to insert an appointment record for that patient. After executing, the transaction is committed.

```
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);

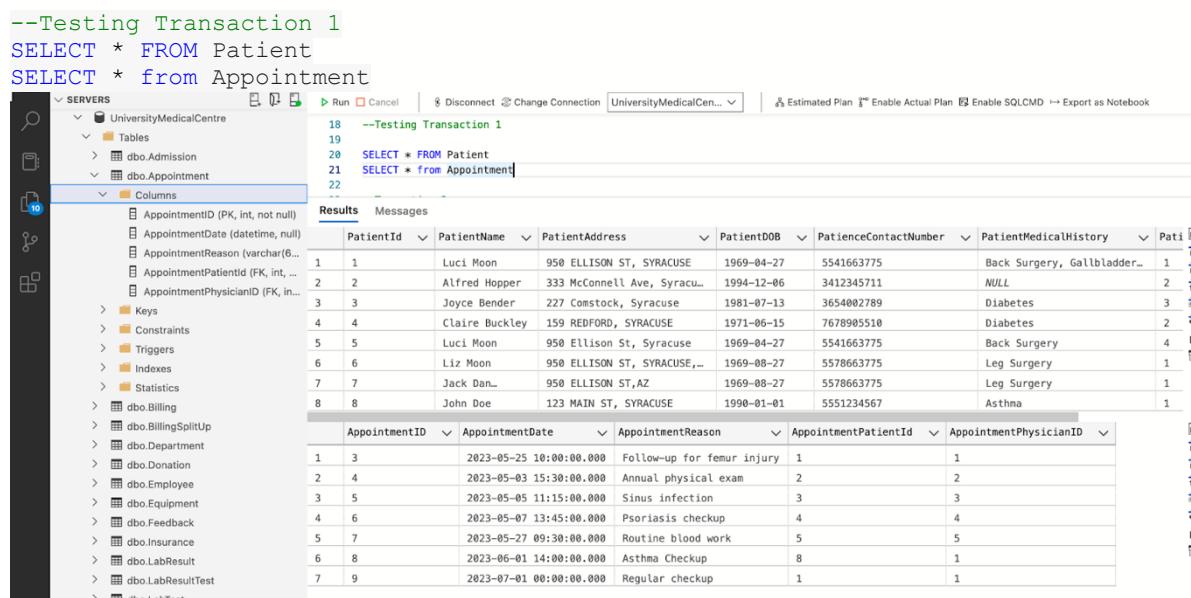
COMMIT TRANSACTION;
```



```
1 BEGIN TRANSACTION;
2     INSERT INTO Patient (PatientName, PatientAddress, PatientDOB,
3     PatienceContactNumber, PatientMedicalHistory, PatientInsuranceID)
4     VALUES ('John Doe', '123 Main St, Syracuse', '1990-01-01', '5551234567',
5     'Asthma', 1);
6
7     DECLARE @NewPatientId INT;
8     SET @NewPatientId = SCOPE_IDENTITY();
9
10
11     INSERT INTO Appointment (AppointmentDate, AppointmentReason, AppointmentPatientId, AppointmentPhysicianID)
12     VALUES ('2023-06-01 14:00:00.000', 'Asthma Checkup', @NewPatientId, 1);
13
14 COMMIT TRANSACTION;
```

Messages

17:29:43 Started executing query at Line 1
(1 row affected)
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.061



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

PatientId	PatientName	PatientAddress	PatientDOB	PatienceContactNumber	PatientMedicalHistory	Pati
1	Luci Moon	950 ELLISON ST, SYRACUSE	1969-04-27	5541663775	Back Surgery, Gallbladder...	1
2	Alfred Hopper	333 McConnell Ave, Syracu...	1994-12-06	3412345711	NULL	2
3	Joyce Bender	227 Comstock, Syracuse	1981-07-13	3654002789	Diabetes	3
4	Claire Buckley	159 REDFORD, SYRACUSE	1971-06-15	7678905510	Diabetes	2
5	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery	4
6	Liz Moon	950 ELLISON ST, SYRACUSE,...	1969-08-27	5578663775	Leg Surgery	1
7	Jack Dan...	950 ELLISON ST, AZ	1969-08-27	5578663775	Leg Surgery	1
8	John Doe	123 MAIN ST, SYRACUSE	1990-01-01	551234567	Asthma	1

AppointmentID	AppointmentDate	AppointmentReason	AppointmentPatientId	AppointmentPhysicianID
1	2023-05-25 10:00:00.000	Follow-up for femur injury	1	1
2	2023-05-03 15:30:00.000	Annual physical exam	2	2
3	2023-05-05 11:15:00.000	Sinus infection	3	3
4	2023-05-07 13:45:00.000	Psoriasis checkup	4	4
5	2023-05-27 09:30:00.000	Routine blood work	5	5
6	2023-06-01 14:00:00.000	Asthma Checkup	8	1
7	2023-07-01 00:00:00.000	Regular checkup	1	1

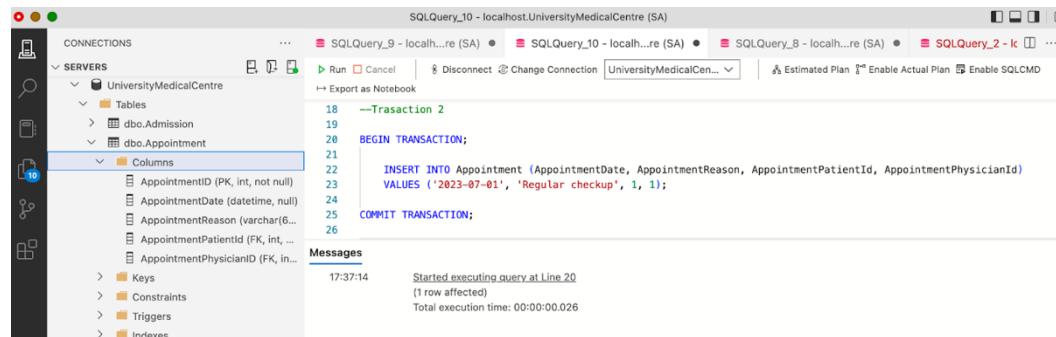
Transaction 2: Add New Appointment

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;
```



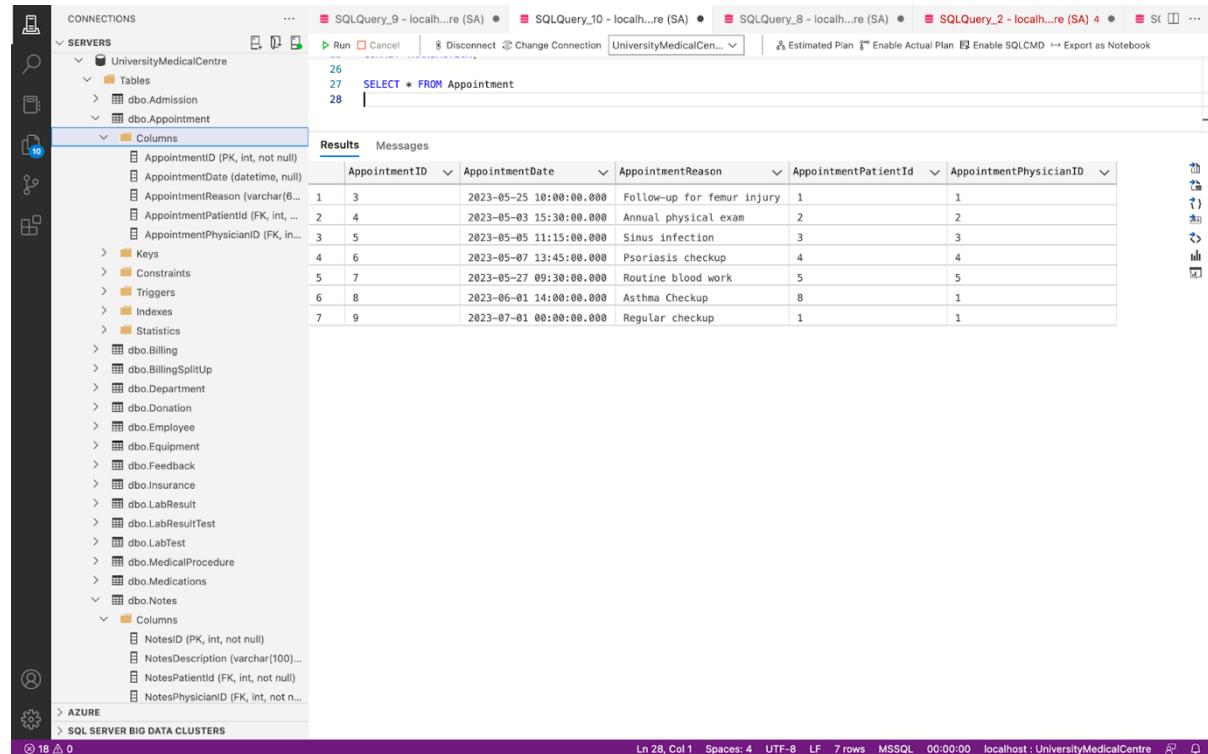
The screenshot shows a SQL Server Management Studio window with the following content:

```
--Trasaction 2
18
19
20 BEGIN TRANSACTION;
21
22 INSERT INTO Appointment (AppointmentDate, AppointmentReason, AppointmentPatientId, AppointmentPhysicianId)
23 VALUES ('2023-07-01', 'Regular checkup', 1, 1);
24
25 COMMIT TRANSACTION;
26
```

Messages pane:

- Started executing query at Line 20
- (1 row affected)
- Total execution time: 00:00:00.026

--Testing Transaction 1



The screenshot shows a SQL Server Management Studio window with the following content:

```
CONNECTIONS ... SQLQuery_9 - localhost.UniversityMedicalCentre (SA) • SQLQuery_10 - localhost.UniversityMedicalCentre (SA) • SQLQuery_8 - localhost.UniversityMedicalCentre (SA) • SQLQuery_2 - localhost.UniversityMedicalCentre (SA) • ...
26
27 SELECT * FROM Appointment
28
```

Results pane:

	AppointmentID	AppointmentDate	AppointmentReason	AppointmentPatientId	AppointmentPhysicianID
1	3	2023-05-25 10:00:00.000	Follow-up for femur injury	1	1
2	4	2023-05-03 15:30:00.000	Annual physical exam	2	2
3	5	2023-05-05 11:15:00.000	Sinus infection	3	3
4	6	2023-05-07 13:45:00.000	Psoriasis checkup	4	4
5	7	2023-05-27 09:30:00.000	Routine blood work	5	5
6	8	2023-06-01 14:00:00.000	Asthma Checkup	8	1
7	9	2023-07-01 00:00:00.000	Regular checkup	1	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;
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the 'Tables' node under 'dbo.Appointment' is expanded, showing the 'Columns' node. The script pane contains the T-SQL code for Transaction 3. The results pane shows the execution log:

```
--TRANSACTION 3
34 BEGIN TRANSACTION;
35 UPDATE Patient
36 SET PatientInsuranceID = 2
37 WHERE PatientId = 2;
38
39 DECLARE @BillToUpdate INT;
40 SELECT @BillToUpdate = BillID FROM Billing WHERE BillPatientId = 2;
41
42 UPDATE Billing
43 SET BillInsuranceID = 2
44 WHERE BillID = @BillToUpdate;
45
46 COMMIT TRANSACTION;
```

Messages

```
17:48:03 Started executing query at Line 36
(1 row affected)
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.043
```

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

The screenshot shows the SQL Server Management Studio interface. The 'Tables' node under 'dbo' is expanded, showing the 'Patient' and 'Billing' tables. The results pane displays the data for both tables.

Patient Table Results:

PatientId	PatientName	PatientAddress	PatientDOB	PatientContactNumber	PatientMedicalHistory	Pati
1	Luci Moon	950 ELLISON ST, SYRACUSE	1969-04-27	5541663775	Back Surgery, Gallbladder...	1
2	Alfred Hopper	333 MCCONNELL AVE, SYRACU...	1994-12-06	3412345711	NULL	2
3	Joyce Bender	227 Constock, Syracuse	1981-07-13	3654002789	Diabetes	3
4	Claire Buckley	159 REDFORD, SYRACUSE	1971-06-15	7678905510	Diabetes	2
5	Luci Moon	950 Ellison St, Syracuse	1969-04-27	5541663775	Back Surgery	4
6	Liz Moon	950 ELLISON ST, SYRACUSE,...	1969-08-27	5578663775	Leg Surgery	1
7	Jack Dan...	950 ELLISON ST, AZ	1969-08-27	5578663775	Leg Surgery	1
8	John Doe	123 MAIN ST, SYRACUSE	1990-01-01	5551234567	Asthma	1

Billing Table Results:

BillID	TotalCharges	BillInsuranceID	BillPatientId
1	1250.5	3	5
2	800	5	1
3	2350.75	1	2
4	450.25	2	3
5	1900	4	4
6	1250.5	3	5
7	800	5	1
8	2350.75	2	2
9	450.25	2	3
10	1900	4	4

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;
```

```
--Transaction 4
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.', 1, 1);

COMMIT TRANSACTION;
```

Messages

Started executing query at Line 60
 (1 row affected)
 (7 rows affected)
 (1 row affected)
 Total execution time: 00:00:00.036

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

MedicationID	MedicationInfo	MedicationName	MedicationDosage	MedicationFrequency	MedicationPatientId	MedicationPhysicianID
1	For pain relief	Ibuprofen	200	3	1	1
2	For high blood pressure	Lisinopril	2	1	2	2
3	For anxiety	Alprazolam	0.5	2	3	3
4	For heartburn	Omeprazole	20	1	4	4
5	For migraines	Sumatriptan	1	2	5	5
6	For allergies	Loratadine	10	1	1	1
7	For allergies	Loratadine	10	1	1	1

FeedbackID	FeedbackNotes	FeedbackPatientId	FeedbackPhysicianID
1	Dr. Hall was very thorough	1	1
2	Nurse Jones was very friendly	2	2
3	The office staff was very helpful	3	3
4	The wait time was a bit long	4	4
5	Dr. Hall explained everything clearly	5	5
6	Er. Hall was very thorough	1	1
7	Dr. Shepherd was very helpful	1	1

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
```

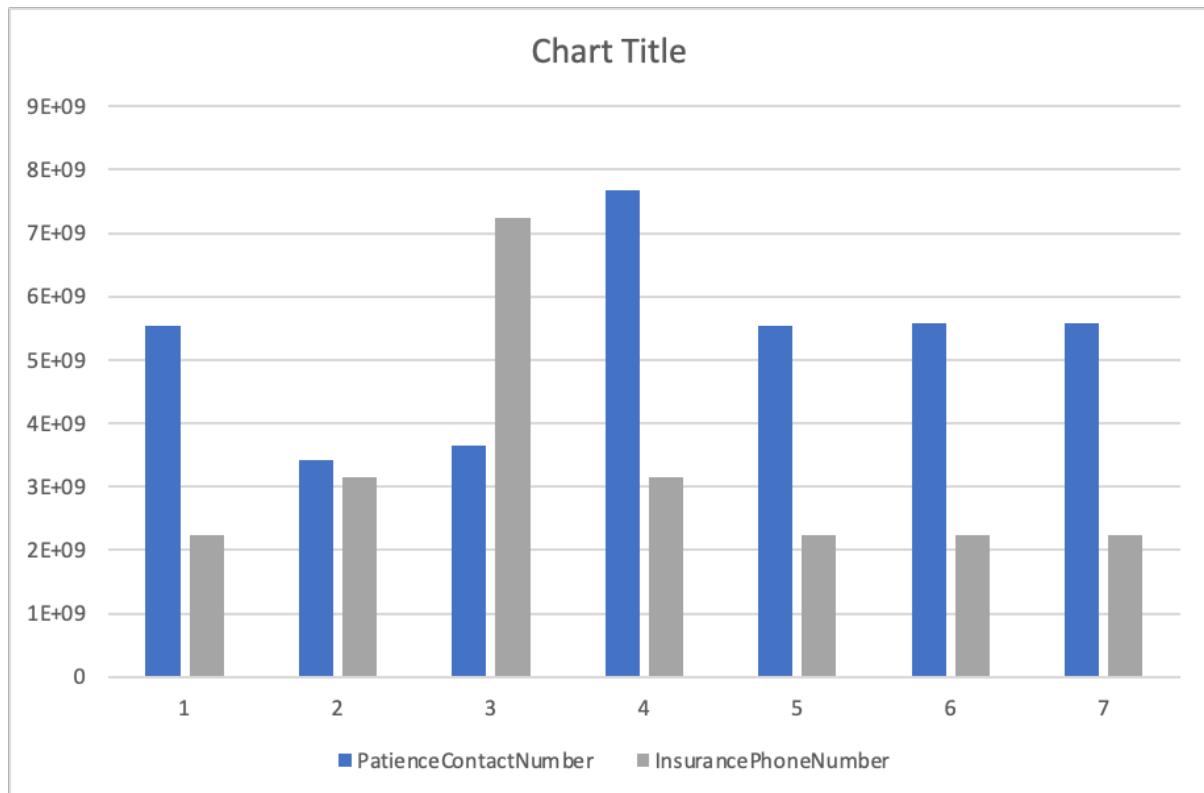
The screenshot shows the SSMS interface with the following details:

- Servers:** A tree view on the left showing database objects like dbo.Appointment, dbo.Billing, etc.
- Code:** The T-SQL script for creating the stored procedure.
- Results:** A table showing patient details and insurance coverage.

	PatientId	PatientName	PatientDOB	PatienceContactNumber	InsuranceCompany	InsurancePhoneNumber	InsuranceCoverage
1	1	Luci Moon	1969-04-27	5541663775	Aetna	2247891300	Full coverage
2	2	Alfred Hopper	1994-12-06	3412345711	Anthem Blue Cross	3147891300	Only surgery
3	3	Joyce Bender	1981-07-13	3654002789	United Health	7247451300	Only Pharmacy
4	4	Claire Buckley	1971-06-15	7678905510	Anthem Blue Cross	3147891300	Only surgery
5	5	Luci Moon	1969-04-27	5541663775	Aetna	2247891234	50% coverage
6	6	Liz Moon	1969-08-27	5578663775	Aetna	2247891300	Full coverage
7	7	Jack Dan...	1969-08-27	5578663775	Aetna	2247891300	Full coverage

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

```

Servers

```

13  --Procedure 2
14
15  CREATE PROCEDURE dbo.DepartmentRevenue
16  AS
17  BEGIN
18      SELECT
19          d.DepartmentID, d.DepartmentName, SUM(b.TotalCharges) AS Revenue
20      FROM Department d
21      JOIN Employee e ON d.DepartmentID = e.EmployeeDeptId
22      JOIN Physician ph ON e.EmployeeID = ph.PhysicianEmpId
23      JOIN Admission a ON ph.PhysicianID = a.AdmissionPhysicianID
24      JOIN Billing b ON a.AdmissionPatientId = b.BillPatientId
25      GROUP BY d.DepartmentID, d.DepartmentName
26
27  END
28  EXEC dbo.DepartmentRevenue

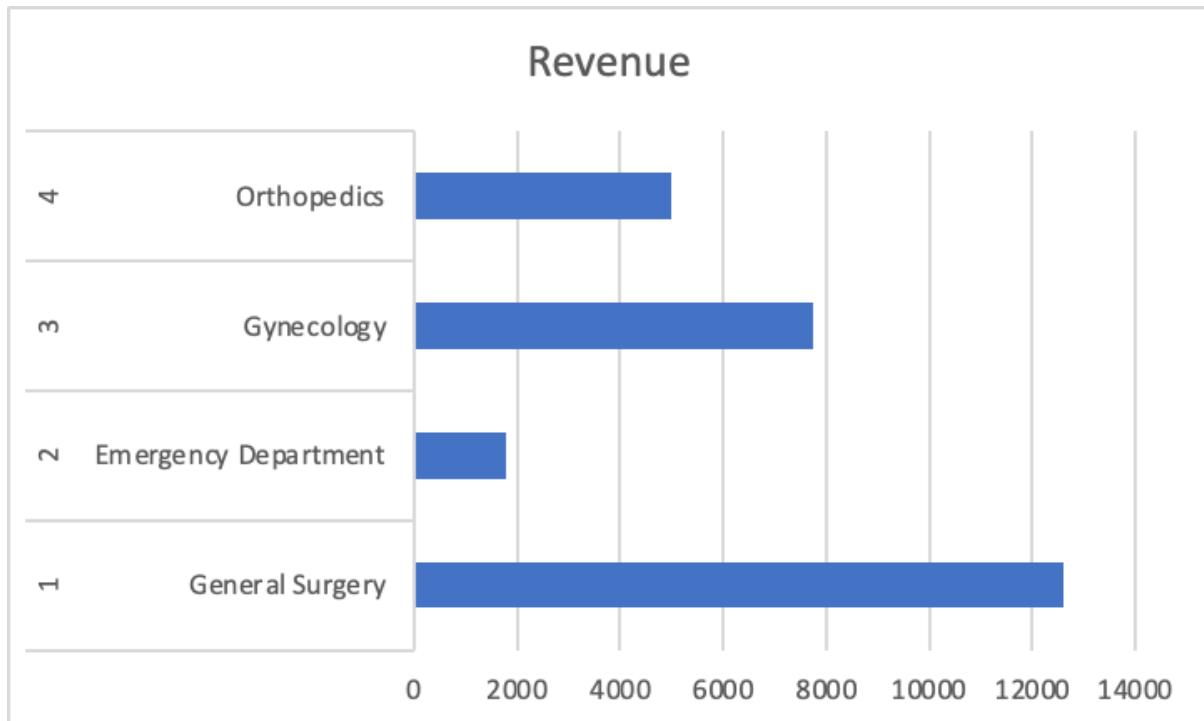
```

Results Messages

	DepartmentID	DepartmentName	Revenue
1	1	General Surgery	12683
2	2	Emergency Depar...	1881
3	3	Gynecology	7768
4	4	Orthopedics	5002

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.

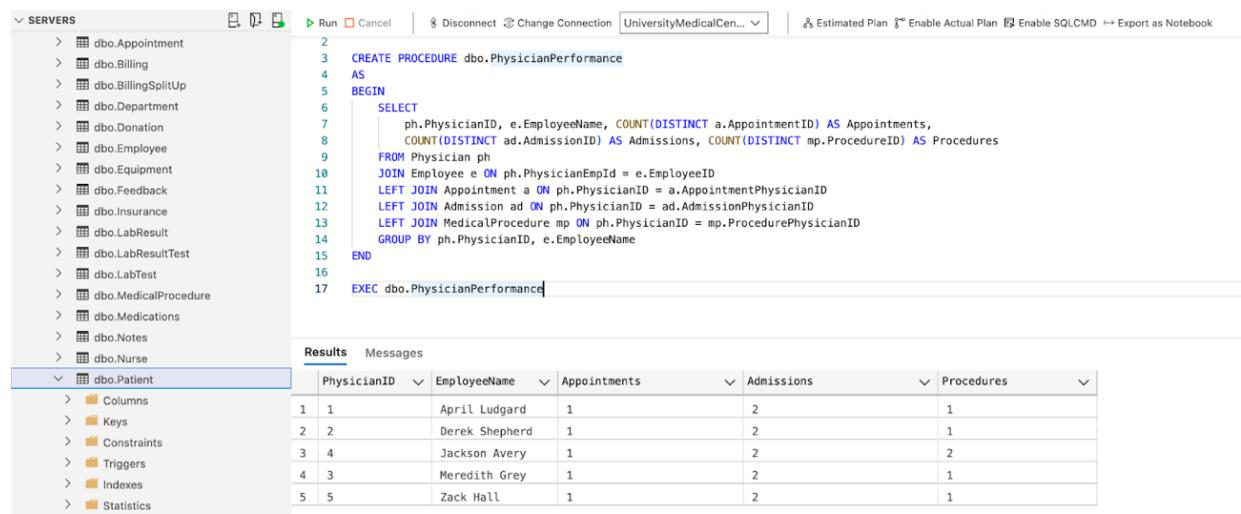


PROCEDURE 3: Physician Performance Report

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
```

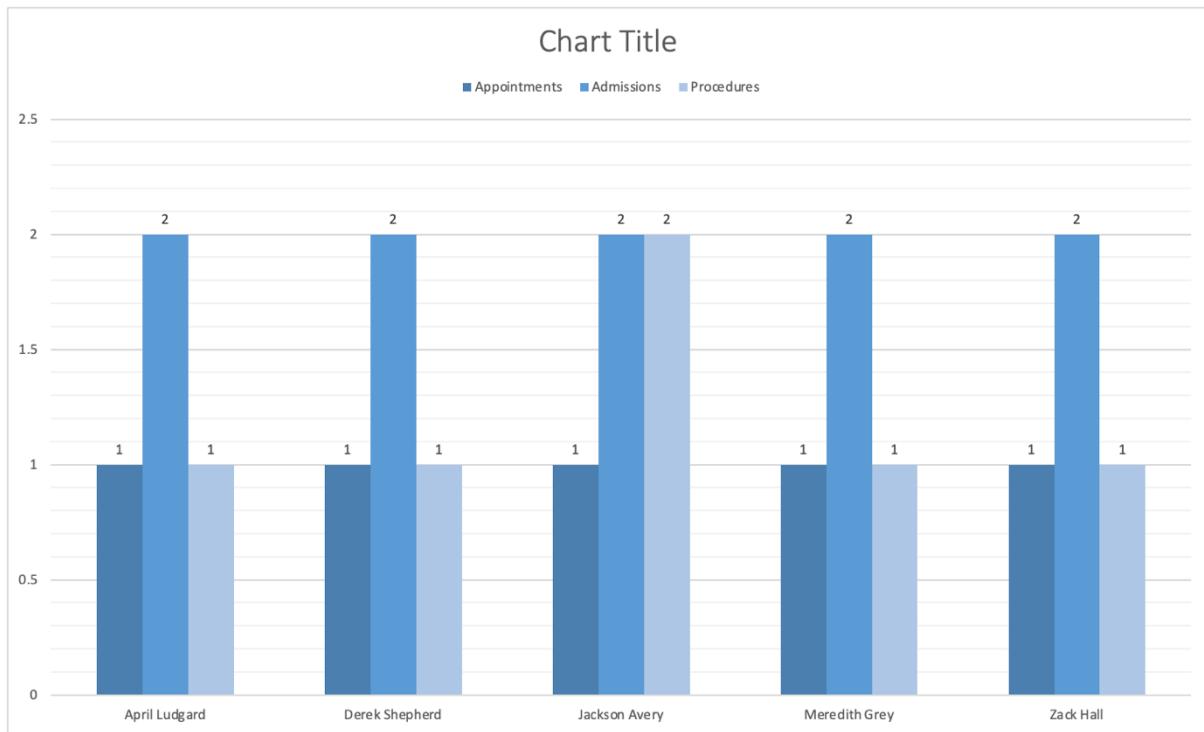


The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays database objects like Appointment, Billing, Department, etc., and the dbo.Patients folder. The main pane shows the T-SQL code for the stored procedure. The code includes a CREATE PROCEDURE statement, a BEGIN block with a SELECT query that joins Physician, Employee, Appointment, Admission, and MedicalProcedure tables, and a GROUP BY clause. Below the code, the EXEC command is shown. The bottom pane shows the Results grid with the following data:

	PhysicianID	EmployeeName	Appointments	Admissions	Procedures
1	1	April Ludgard	1	2	1
2	2	Derek Shepherd	1	2	1
3	4	Jackson Avery	1	2	2
4	3	Meredith Grey	1	2	1
5	5	Zack Hall	1	2	1

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

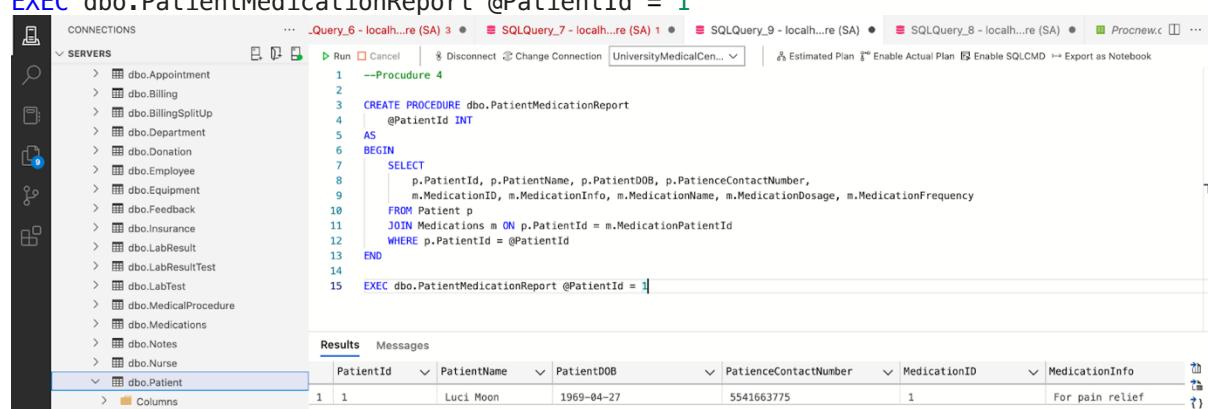
This stored procedure lists all medications prescribed to a specific patient. It selects the patient's details and medication information by joining the Patient and Medications tables and filtering by patient ID.

```

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

```

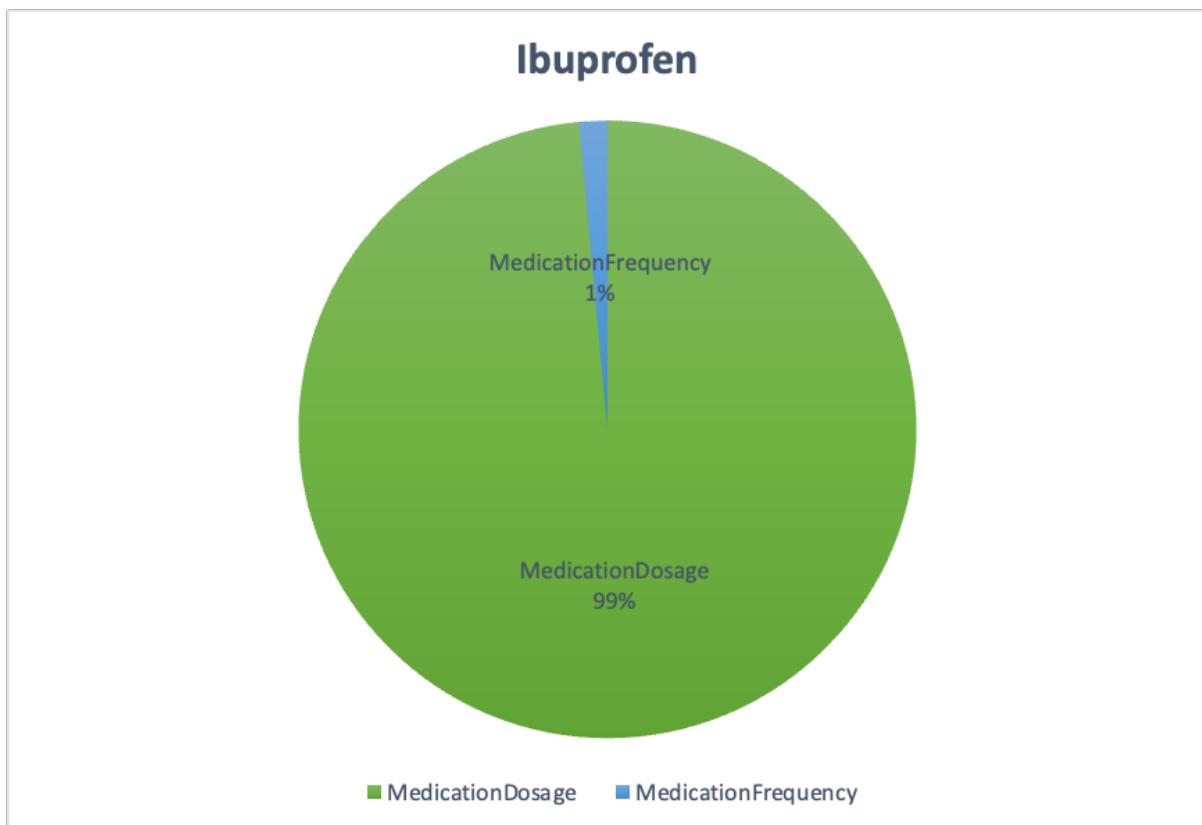
```
EXEC dbo.PatientMedicationReport @PatientId = 1
```



PatientId	PatientName	PatientDOB	PatienceContactNumber	MedicationID	MedicationInfo
1	Luci Moon	1969-04-27	5541663775	1	For pain relief

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
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, the 'Tables' node under the 'UniversityMedicalCentre' database is expanded, showing tables like 'dbo.Admission', 'dbo.Appointment', and 'dbo.Billing'. In the center pane, a query window displays the following T-SQL script:

```
CREATE ROLE ReadOnly;
GO

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

The 'Messages' pane at the bottom shows the execution results:

```
14:21:41 Started executing query at Line 1
14:21:41 Commands completed successfully.
14:21:41 Started executing query at Line 3
14:21:41 Commands completed successfully.
Total execution time: 00:00:00.081
```

Assigning role:

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

CREATE USER ReadOnlyUser FOR LOGIN ReadOnlyUser;
GO

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

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, the 'Tables' node under the 'UniversityMedicalCentre' database is expanded, showing tables like 'dbo.Admission', 'dbo.Appointment', 'dbo.Billing', etc. In the center pane, a query window displays the following T-SQL script:

```
32
33 -- ReadOnly user
34 CREATE LOGIN ReadOnlyUser WITH PASSWORD = 'C0mpl3xP@ssw0rd!';
35 GO
36
37 CREATE USER ReadOnlyUser FOR LOGIN ReadOnlyUser;
38 GO
39
40 EXEC sp_addrolemember 'db_datareader', 'ReadOnlyUser';
41 GO
```

The 'Messages' pane at the bottom shows the execution results:

```
14:32:40 Started executing query at Line 34
14:32:40 Commands completed successfully.
14:32:40 Started executing query at Line 36
14:32:40 Commands completed successfully.
14:32:40 Started executing query at Line 39
14:32:40 Commands completed successfully.
Total execution time: 00:00:00.082
```

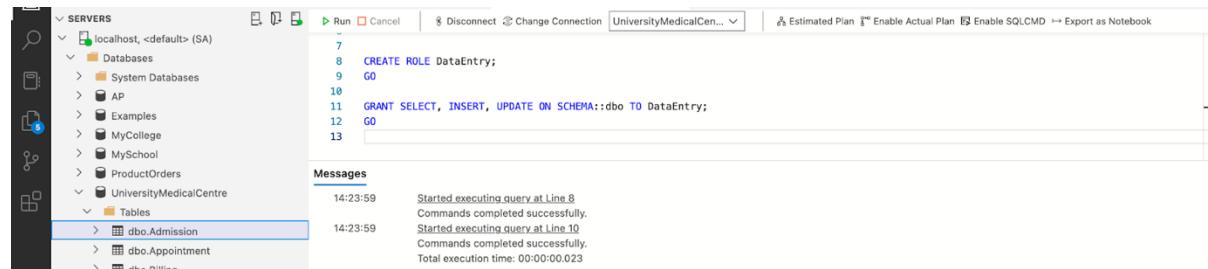
2. DataEntry User:

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
```



The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, under 'Tables' of the 'UniversityMedicalCentre' database, the 'Tables' node is expanded, and the 'dbo.Admission' table is selected. In the main query window, the following T-SQL script is being run:

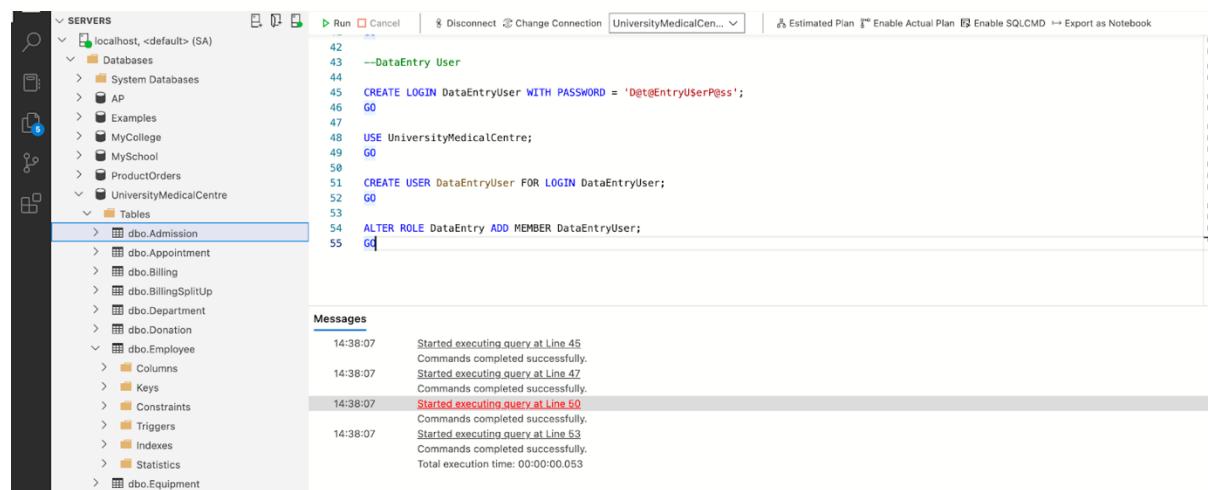
```
7
8 CREATE ROLE DataEntry;
9 GO
10
11 GRANT SELECT, INSERT, UPDATE ON SCHEMA::dbo TO DataEntry;
12 GO
13
```

The 'Messages' pane at the bottom displays the execution results:

- Line 8: Started executing query at Line 8. Commands completed successfully.
- Line 11: Started executing query at Line 10. Commands completed successfully.
- Total execution time: 00:00:00.023

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
```



The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, under 'Tables' of the 'UniversityMedicalCentre' database, the 'Tables' node is expanded, and the 'dbo.Admission' table is selected. In the main query window, the following T-SQL script is being run:

```
42
43 --DataEntry User
44
45 CREATE LOGIN DataEntryUser WITH PASSWORD = 'D@t@EntryUserP@ss';
46 GO
47
48 USE UniversityMedicalCentre;
49 GO
50
51 CREATE USER DataEntryUser FOR LOGIN DataEntryUser;
52 GO
53
54 ALTER ROLE DataEntry ADD MEMBER DataEntryUser;
55 GO
```

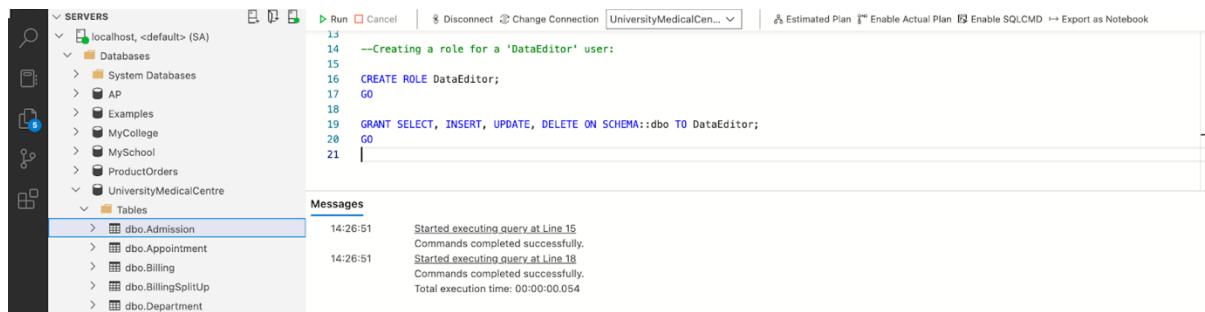
The 'Messages' pane at the bottom displays the execution results:

- Line 45: Started executing query at Line 45. Commands completed successfully.
- Line 47: Started executing query at Line 47. Commands completed successfully.
- Line 50: Started executing query at Line 50. Commands completed successfully.
- Line 53: Started executing query at Line 53. Commands completed successfully.
- Total execution time: 00:00:00.053

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
```



The screenshot shows the SQL Server Management Studio interface. In the Object Explorer on the left, under 'Tables' in the 'UniversityMedicalCentre' database, several tables like 'dbo.Admission', 'dbo.Appointment', etc., are listed. The 'Messages' pane at the bottom shows the execution of the T-SQL script. The output includes the creation of the role, granting permissions, and the creation of the login and user, all completed successfully in 0:00:00.054.

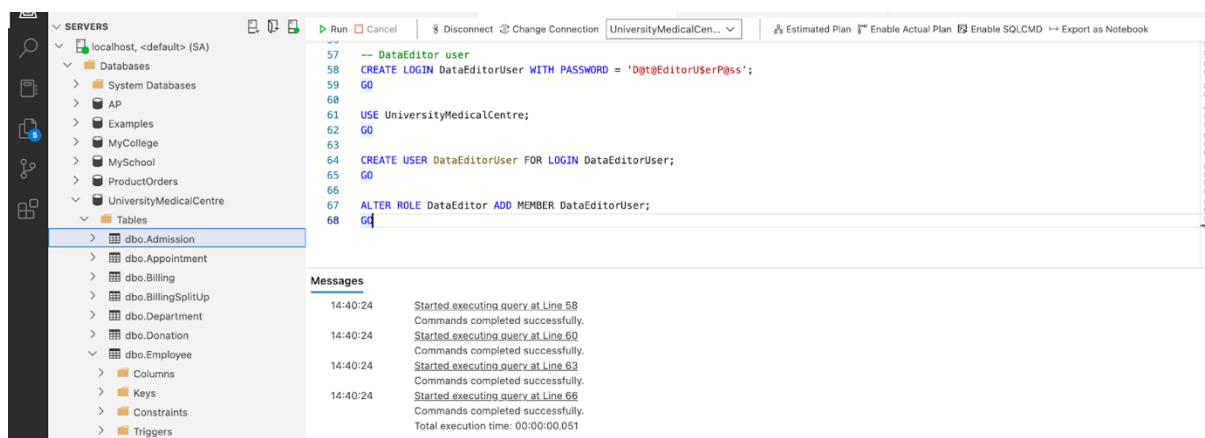
```
13
14  --Creating a role for a 'DataEditor' user:
15
16 CREATE ROLE DataEditor;
17 GO
18
19 GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO DataEditor;
20 GO
21 |
```

Messages

```
14:26:51 Started executing query at Line 15
14:26:51 Commands completed successfully.
14:26:51 Started executing query at Line 18
14:26:51 Commands completed successfully.
Total execution time: 00:00:00.054
```

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
```



The screenshot shows the SQL Server Management Studio interface. Under 'Tables' in the 'UniversityMedicalCentre' database, various objects like 'dbo.Employee' and its sub-objects 'Columns', 'Keys', 'Constraints', and 'Triggers' are listed. The 'Messages' pane shows the execution of the T-SQL script, which creates the login, user, and adds the user to the role, all completed successfully in 0:00:00.051.

```
57  -- DataEditor user
58 CREATE LOGIN DataEditorUser WITH PASSWORD = 'D@t@EditorU$erP@ss';
59 GO
60
61 USE UniversityMedicalCentre;
62 GO
63
64 CREATE USER DataEditorUser FOR LOGIN DataEditorUser;
65 GO
66
67 ALTER ROLE DataEditor ADD MEMBER DataEditorUser;
68 GO
```

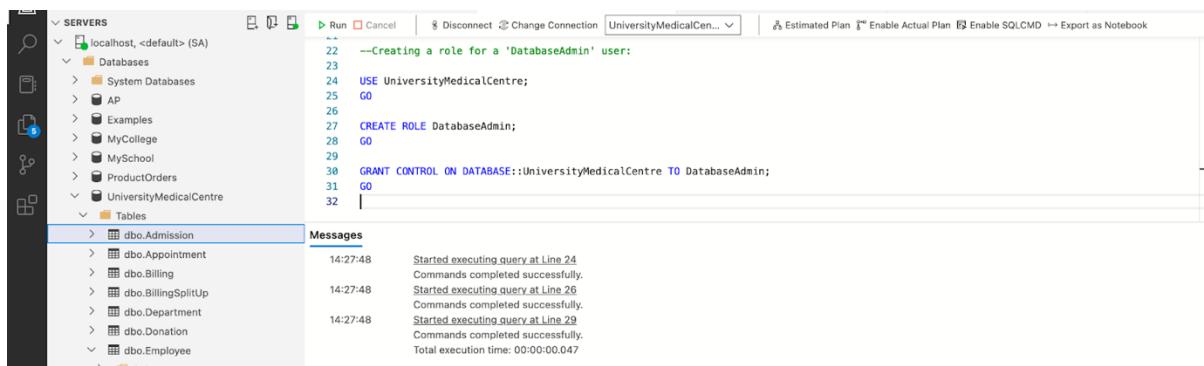
Messages

```
14:40:24 Started executing query at Line 58
14:40:24 Commands completed successfully.
14:40:24 Started executing query at Line 60
14:40:24 Commands completed successfully.
14:40:24 Started executing query at Line 63
14:40:24 Commands completed successfully.
14:40:24 Started executing query at Line 66
14:40:24 Commands completed successfully.
Total execution time: 00:00:00.051
```

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
```



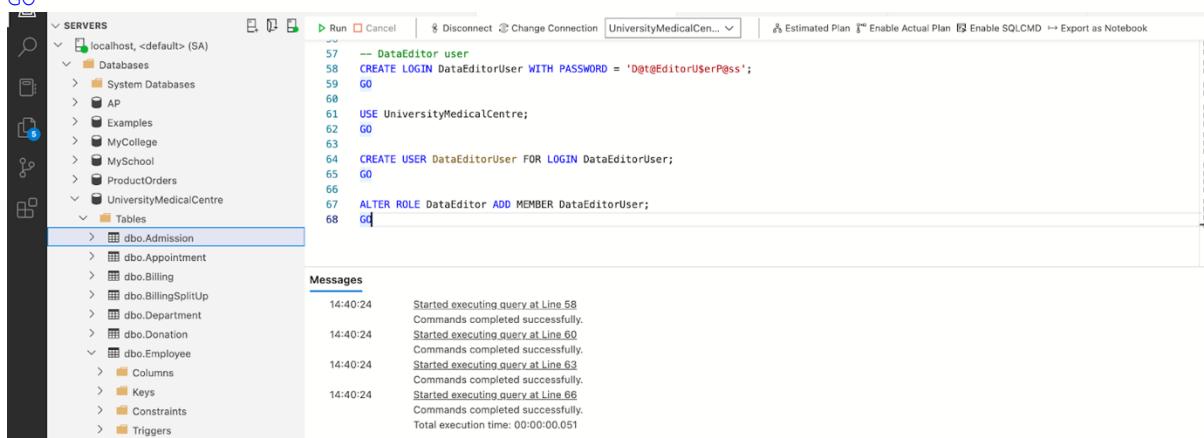
The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure under 'UniversityMedicalCentre'. In the center, the 'Query Editor' window contains the T-SQL script for creating the 'DatabaseAdmin' role and granting it control over the database. On the right, the 'Messages' pane shows the execution log with four entries indicating successful command completion and a total execution time of 0:00:00.047.

```
22 --Creating a role for a 'DatabaseAdmin' user:
23
24 USE UniversityMedicalCentre;
25 GO
26
27 CREATE ROLE DatabaseAdmin;
28 GO
29
30 GRANT CONTROL ON DATABASE::UniversityMedicalCentre TO DatabaseAdmin;
31 GO
32 |
```

Time	Message
14:27:48	Started executing query at Line 24
14:27:48	Commands completed successfully.
14:27:48	Started executing query at Line 26
14:27:48	Commands completed successfully.
14:27:48	Started executing query at Line 29
14:27:48	Commands completed successfully.
	Total execution time: 00:00:00.047

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
```



The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure under 'UniversityMedicalCentre'. In the center, the 'Query Editor' window contains the T-SQL script for creating the 'DataEditorUser' login, creating the 'DataEditorUser' user, and adding 'DataEditorUser' to the 'DataEditor' role. On the right, the 'Messages' pane shows the execution log with five entries indicating successful command completion and a total execution time of 0:00:00.051.

```
57 -- DataEditor user
58 CREATE LOGIN DataEditorUser WITH PASSWORD = 'D@t@EditorU$erP@ss';
59 GO
60
61 USE UniversityMedicalCentre;
62 GO
63
64 CREATE USER DataEditorUser FOR LOGIN DataEditorUser;
65 GO
66
67 ALTER ROLE DataEditor ADD MEMBER DataEditorUser;
68 GO
```

Time	Message
14:40:24	Started executing query at Line 58
14:40:24	Commands completed successfully.
14:40:24	Started executing query at Line 60
14:40:24	Commands completed successfully.
14:40:24	Started executing query at Line 63
14:40:24	Commands completed successfully.
14:40:24	Started executing query at Line 66
14:40:24	Commands completed successfully.
	Total execution time: 00:00:00.051

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.