

# Medical Center

CSE581 – Project 2



**Abstract** - This project aims to design and implement a robust and efficient database management system (DBMS) for a medical institute using SQL. The database will cater to the various needs of the medical setting, encompassing patient management, physician and nurse schedules, medical procedures, medication prescriptions, lab results, billing, and more. The database design will involve creating several base tables such as Patients, Physicians, Nurses, Notes, Appointments, Admissions, Procedures, Medications, Lab Results, and Billing. These tables will capture essential information including patient details, medical history, insurance information, physician profiles, nurse information, notes made by healthcare professionals, appointment scheduling, patient admissions and discharges, medical procedures performed, prescribed medications, laboratory test results, and billing information. The completed DBMS project aims to provide a comprehensive platform for managing medical data efficiently and accurately.

By:

Tanmay Sameer Unhale  
SUID: 899172356  
[tunhale@syr.edu](mailto:tunhale@syr.edu)

## **Contents:**

Sr. No.	Topic	Pg. No.
1	Contents	1
2	Introduction	2
3	Project Phases	2
4	Design Phase	3
5	Database design	3
6	Tables and Columns	3
7	Potential Integrity and Security Threats	6
8	Relationships between tables	7
9	Implementation Phase	8
10	Creating tables	8
11	Inserting data into tables	15
12	Observing tables	22
13	Testing Phase	30
14	Views	30
15	Stored Procedures	34
16	User defined functions	40
17	Triggers	44
18	Transactions	48
19	Scripts	52
20	Business Report	56
21	Conclusion	60
22	Remark	60

## **Introduction:**

This Healthcare Management System titled “Medical Center” is a comprehensive project that aims to streamline and optimize various processes within a healthcare facility, leveraging a well-designed and efficient database. This project encompasses a range of functionalities, including patient management, physician coordination, nurse assignments, appointment scheduling, medical procedures, medication tracking, and billing management. By utilizing this integrated system, healthcare providers can enhance the overall patient experience, improve operational efficiency, and ensure accurate and timely healthcare delivery. The database forms the backbone of the Healthcare Management System, containing structured and organized data pertaining to patients, physicians, nurses, rooms, procedures, medications, specialities, and more. It facilitates seamless data retrieval, storage, and manipulation, enabling healthcare professionals to access critical information quickly and make informed decisions. Patients play a central role in the system, with their details, such as names, dates of birth, genders, and contact information, being recorded. Patient addresses and insurance information are also captured, allowing for efficient communication and insurance coverage management. Appointments can be scheduled between patients and physicians, with details such as appointment dates, times, and reasons being recorded. Physicians and nurses are an integral part of the system, with their respective details, including names, contact information, and specialties, being stored. Physicians can be associated with specific procedures and patients, and their licenses are recorded. Nurses are assigned to various departments, and their contact information is maintained for effective communication. The system facilitates the management of medical procedures, including surgeries, blood work, stitching, chemotherapy, and imaging scans. It also tracks medication details, such as names, manufacturers, dosages, and frequencies, ensuring accurate administration and monitoring. Moreover, patient notes, lab results, and billing information are stored, providing a holistic view of the patient's medical journey. Billing management is another crucial aspect of the project, with insurance details, total bills, insurance coverage, and patient payments being recorded. This enables efficient billing processing and ensures accurate financial management.

## **Project Phases:**

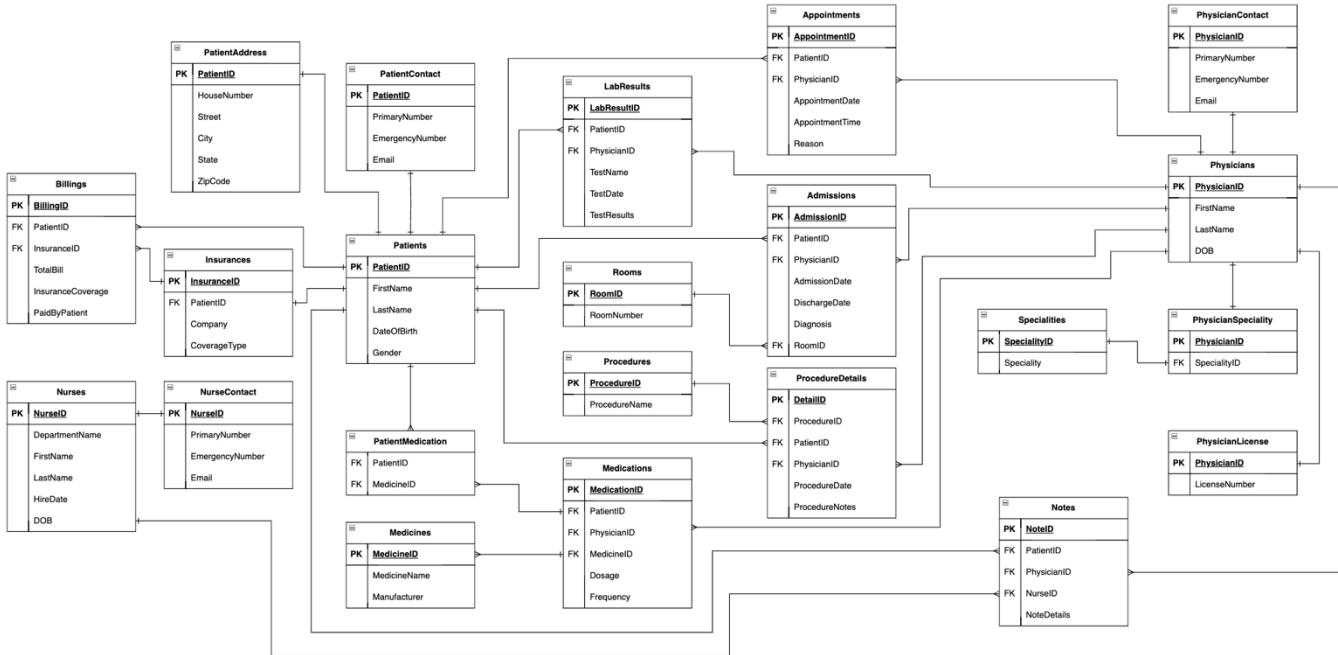
This project consists of three phases:

- 1.Design Phase
- 2.Implementation Phase
- 3.Testing Phase

# 1. Design Phase

## 1.1 Database Design

The ER-Diagram for the database is this:



There are 22 tables in this following database with one-to-one and one-to-many relationship. There is also a many-to-many relationship which is resolved using a linking table.

## 1.2 Tables and Columns

### 1. Patients Table

- PatientID (Primary Key)
- FirstName
- LastName
- DateOfBirth
- Gender

### 2. PatientContact Table:

- PatientID (Primary Key)
- PrimaryNumber
- EmergencyNumber
- Email

### 3. PatientAddress Table:

- PatientID (Primary Key)
- HouseNumber
- Street
- City
- State
- ZipCode

4. Insurances Table:

- InsuranceID (Primary Key)
- PatientID
- Company
- CoverageType

5. Billings Table:

- BillingID (Primary Key)
- PatientID
- InsuranceID
- TotalBill
- InsuranceCoverage
- PaidByPatient

6. Physicians Table:

- PhysicianID (Primary Key)
- FirstName
- LastName
- DOB

7. PhysicianContact Table:

- PhysicianID (Primary Key)
- PrimaryNumber
- EmergencyNumber
- Email

8. PhysicianLicense Table:

- PhysicianID (Primary Key)
- LicenseNumber

9. Specialities Table:

- SpecialityID (Primary Key)
- Speciality

10. PhysicianSpeciality Table:

- PhysicianID (Primary Key)
- SpecialityID

11. Nurses Table:

- NurseID (Primary Key)

- DepartmentName
- FirstName
- LastName
- HireDate
- DOB

12. NurseContact Table:

- NurseID (Primary Key)
- PrimaryNumber
- EmergencyNumber
- Email

13. Appointments Table:

- AppointmentID (Primary Key)
- PatientID
- PhysicianID
- AppointmentDate
- AppointmentTime
- Reason

14. Rooms Table:

- RoomID (Primary Key)
- RoomNumber

15. Admissions Table:

- AdmissionID (Primary Key)
- PatientID
- PhysicianID
- RoomID
- AdmissionDate
- DischargeDate
- Diagnosis

16. Procedures Table:

- ProcedureID (Primary Key)
- ProcedureName

17. ProcedureDetails Table:

- DetailID (Primary Key)
- ProcedureID
- PatientID
- PhysicianID
- ProcedureDate
- ProcedureNotes

18. Medicines Table:

- MedicineID (Primary Key)
- MedicineName
- Manufacturer

19. Medications Table:

- MedicationID (Primary Key)
- PatientID
- PhysicianID
- MedicineID
- Dosage
- Frequency

20. Notes Table:

- NoteID (Primary Key)
- PatientID
- PhysicianID
- NurseID
- NoteDetails

21. LabResults Table:

- LabResultID (Primary Key)
- PatientID
- PhysicianID
- TestName
- TestDate
- TestResults

22. PatientMedication Table:

- PatientID (Foreign Key)
- MedicineID (Foreign Key)

### **1.3 Potential Integrity and Security Threats:**

Possible integrity and security issues may be reduced by normalizing the tables. Referential integrity refers to the accurate upkeep of table relationships. This is employed to address integrity-related issues in database design. Every field in a base table designated as a foreign key must be capable of holding null values, values from a parent table's primary key, or values from a candidate key in order for referential integrity to be preserved in a database. This implies that every foreign key in a foreign key table must match a primary key in the connected database. Removing a record that contains a value that is referred to by a foreign key in another table would violate referential integrity if it were not preserved. Referential integrity is also included, changing the database design to minimize integrity and security problems while minimizing duplication. I've reached third normal form on my E/R diagram. I've divided the tables into several tables so that they can morph into the third normal form.

Here, the patient information is divided into Patients table, PatientAddress table and PatientContact table. The physician details are divided into Physicians table, PhysicianContact table, PhysicianLicense table and PhysicianSpeciality table. The nurse details are stored in Nurses table and NurseContact table.

#### **1.4 Relationships between table:**

These following are **one-to-one** relationships:

Patients and PatientContact – As there is one contact detail for same patient.

Patients and PatientAddress – As there is one address detail for same patient.

Patients and Insurances – As there is one insurance detail for same patient.

Physicians and PhysicianContact – As there is one contact detail for same physician.

Physicians and PhysicianSpeciality – As there is one speciality for same physician (assumption).

Physicians and PhysicianLicense – As there is one license details for the same physician.

PhysicianSpeciality and Specialities – As there is one speciality for every physician.

Nurses and NurseContact – As there is one contact detail for same nurse.

These following are **one-to-many** relationships:

Patients and Billings – As one patient can have multiple bills.

Insurances and Billings – As one insurance can be used in multiple bills.

Patients and Appointments – As one patient can have multiple appointments.

Patients and LabResults – As one patient can have multiple lab tests and lab results.

Patients and Admissions – As one patient can be admitted multiple times (assumption).

Patients and ProcedureDetails – As one patient can have multiple procedure.

Patients and Notes – As one patient can have multiple notes.

Rooms and Admission – As one room can be assigned to multiple admission.

Procedures and ProcedureDetails – As same one procedure can be used multiple time.

Physician and Appointments – As one physician can have multiple appointments.

Physician and ProcedureDetails – As one physician might have to perform multiple procedures.

Physician and Notes – As one physician can have multiple notes.

Physician and Medications – As one doctor can give multiple medicines to multiple patients.

Medications and Medicines – As one medicine can be prescribed multiple.

**Many-to-many** relationship:

Patients and Medications tables has a many to many relationship and PatientMedication table is used as a linking table here.

## 2. Implementation Phase

### 2.1 Creating all tables

**Code to create all tables:**

```
CREATE TABLE Patients
(
    PatientID INT NOT NULL IDENTITY(1,1),
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Gender VARCHAR(1) NOT NULL,
    PRIMARY KEY (PatientID)
);
```

```
CREATE TABLE PatientContact
(
    PatientID INT NOT NULL IDENTITY(1,1),
    PrimaryNumber VARCHAR(15) NOT NULL,
    EmergencyNumber VARCHAR(15) NOT NULL,
    Email VARCHAR(30) NOT NULL,
    PRIMARY KEY (PatientID)
);
```

```
CREATE TABLE PatientAddress
(
    PatientID INT NOT NULL IDENTITY(1,1),
    HouseNumber VARCHAR(4) NOT NULL,
    Street VARCHAR(20) NOT NULL,
    City VARCHAR(20) NOT NULL,
    State VARCHAR(20) NOT NULL,
    ZipCode VARCHAR(6) NOT NULL,
    PRIMARY KEY (PatientID)
);
```

```
CREATE TABLE Insurances
```

```
(  
    InsuranceID INT NOT NULL IDENTITY(1,1),  
    PatientID INT NOT NULL,  
    Company VARCHAR(30) NOT NULL,  
    CoverageType VARCHAR(20) NOT NULL,  
    PRIMARY KEY (InsuranceID),  
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)  
);
```

```
CREATE TABLE Billings  
(  
    BillingID INT NOT NULL IDENTITY(1,1),  
    PatientID INT NOT NULL,  
    InsuranceID INT NOT NULL,  
    TotalBill MONEY NOT NULL,  
    InsuranceCoverage MONEY NOT NULL,  
    PaidByPatient MONEY NOT NULL,  
    PRIMARY KEY (BillingID),  
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),  
    FOREIGN KEY (InsuranceID) REFERENCES Insurances(InsuranceID)  
);
```

```
CREATE TABLE Physicians  
(  
    PhysicianID INT NOT NULL IDENTITY(1,1),  
    FirstName VARCHAR(20) NOT NULL,  
    LastName VARCHAR(20) NOT NULL,  
    DOB DATE NOT NULL,  
    PRIMARY KEY (PhysicianID)  
);
```

```
CREATE TABLE PhysicianContact  
(  
    PhysicianID INT NOT NULL IDENTITY(1,1),  
    PrimaryNumber VARCHAR(15) NOT NULL,  
    EmergencyNumber VARCHAR(15) NOT NULL,  
    Email VARCHAR(30) NOT NULL,
```

```
    PRIMARY KEY (PhysicianID)
);

CREATE TABLE PhysicianLicense
(
    PhysicianID INT NOT NULL IDENTITY(1,1),
    LicenseNumber VARCHAR(7) NOT NULL,
    PRIMARY KEY (PhysicianID)
);

CREATE TABLE Specialities
(
    SpecialityID INT NOT NULL IDENTITY(1,1),
    Speciality VARCHAR(20) NOT NULL,
    PRIMARY KEY (SpecialityID)
);

CREATE TABLE PhysicianSpeciality
(
    PhysicianID INT NOT NULL IDENTITY(1,1),
    SpecialityID INT NOT NULL,
    PRIMARY KEY (PhysicianID),
    FOREIGN KEY (SpecialityID) REFERENCES Specialities(SpecialityID)
);

CREATE TABLE Nurses
(
    NurseID INT NOT NULL IDENTITY(1,1),
    DepartmentName VARCHAR(20) NOT NULL,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    HireDate DATE NOT NULL,
    DOB DATE NOT NULL,
    PRIMARY KEY (NurseID)
);

CREATE TABLE NurseContact
```

```
(  
    NurseID INT NOT NULL IDENTITY(1,1),  
    PrimaryNumber VARCHAR(15) NOT NULL,  
    EmergencyNumber VARCHAR(15) NOT NULL,  
    Email VARCHAR(30) NOT NULL,  
    PRIMARY KEY (NurseID)  
);
```

```
CREATE TABLE Appointments  
(  
    AppointmentID INT NOT NULL IDENTITY(1,1),  
    PatientID INT NOT NULL,  
    PhysicianID INT NOT NULL,  
    AppointmentDate DATE NOT NULL,  
    AppointmentTime TIME NOT NULL,  
    Reason VARCHAR(50) NOT NULL,  
    PRIMARY KEY (AppointmentID),  
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),  
    FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID)  
);
```

```
CREATE TABLE Rooms  
(  
    RoomID INT NOT NULL IDENTITY(1,1),  
    RoomNumber INT NOT NULL,  
    PRIMARY KEY (RoomID)  
);
```

```
CREATE TABLE Admissions  
(  
    AdmissionID INT NOT NULL IDENTITY(1,1),  
    PatientID INT NOT NULL,  
    PhysicianID INT NOT NULL,  
    RoomID INT NOT NULL,  
    AdmissionDate DATE NOT NULL,  
    DischargeDate DATE ,  
    Diagnosis VARCHAR(50) NOT NULL,
```

```
    PRIMARY KEY (AdmissionID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID),
    FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)
);
```

```
CREATE TABLE Procedures
(
    ProcedureID INT NOT NULL IDENTITY(1,1),
    ProcedureName VARCHAR(20) NOT NULL,
    PRIMARY KEY (ProcedureID)
);
```

```
CREATE TABLE ProcedureDetails
(
    DetailID INT NOT NULL IDENTITY(1,1),
    ProcedureID INT NOT NULL,
    PatientID INT NOT NULL,
    PhysicianID INT NOT NULL,
    ProcedureDate DATE NOT NULL,
    ProcedureNotes VARCHAR(50) NOT NULL,
    PRIMARY KEY (DetailID),
    FOREIGN KEY (ProcedureID) REFERENCES Procedures(ProcedureID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID),
);
```

```
CREATE TABLE Medicines
(
    MedicineID INT NOT NULL IDENTITY(1,1),
    MedicineName VARCHAR(30) NOT NULL,
    Manufacturer VARCHAR(30) NOT NULL,
    PRIMARY KEY (MedicineID)
);
```

```
CREATE TABLE Medications
(
```

```
MedicationID INT NOT NULL IDENTITY(1,1),
PatientID INT NOT NULL,
PhysicianID INT NOT NULL,
MedicineID INT NOT NULL,
Dosage VARCHAR(15) NOT NULL,
Frequency VARCHAR(15) NOT NULL,
PRIMARY KEY (MedicationID),
FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID),
FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID)
);
```

```
CREATE TABLE Notes
(
    NoteID INT NOT NULL IDENTITY(1,1),
    PatientID INT NOT NULL,
    PhysicianID INT NOT NULL,
    NurseID INT NOT NULL,
    NoteDetails VARCHAR(50) NOT NULL,
    PRIMARY KEY (NoteID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID),
    FOREIGN KEY (NurseID) REFERENCES Nurses(NurseID)
);
```

```
CREATE TABLE LabResults
(
    LabResultID INT NOT NULL IDENTITY(1,1),
    PatientID INT NOT NULL,
    PhysicianID INT NOT NULL,
    TestName VARCHAR(30) NOT NULL,
    TestDate DATE NOT NULL,
    TestResults VARCHAR(20) NOT NULL,
    PRIMARY KEY (LabResultID),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID)
);
```

```
CREATE TABLE PatientMedication
```

```
(  
    PatientID INT NOT NULL,  
    MedicineID INT NOT NULL,  
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),  
    FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID)  
);
```

### Screenshot of execution:

The screenshot shows the MySQL Workbench interface with the 'MedicalCenter' connection selected. The 'Output' tab displays the SQL code for creating the 'PatientMedication' table. The code includes the table definition with primary key constraints and foreign key references to the 'Patients' and 'Medicines' tables. Below the code, the 'Messages' section shows the execution log: 'Started executing query at Line 9', 'Commands completed successfully.', and 'Total execution time: 00:00:00.058'. The bottom status bar indicates the query was run at 6:00:45 PM.

```
227 PhysicianID INT NOT NULL,  
228 TestName VARCHAR(30) NOT NULL,  
229 TestDate DATE NOT NULL,  
230 TestResults VARCHAR(20) NOT NULL,  
231 PRIMARY KEY (LabResultID),  
232 FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),  
233 FOREIGN KEY (PhysicianID) REFERENCES Physicians(PhysicianID)  
234 );  
235  
236  
237 -- Linking Tables  
238 CREATE TABLE PatientMedication  
239 (  
240     PatientID INT NOT NULL,  
241     MedicineID INT NOT NULL,  
242     FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),  
243     FOREIGN KEY (MedicineID) REFERENCES Medicines(MedicineID)  
244 );  
245  
246  
247  
248
```

Messages

6:00:45 PM Started executing query at Line 9  
Commands completed successfully.  
Total execution time: 00:00:00.058

PROBLEMS 7 OUTPUT TERMINAL TASKS

MySQL Tools Service

AZURE

Ln 5, Col 1 Spaces: 4 UTF-8 LF { } SQL 0 rows Choose SQL Language 00:00:00 localhost:MedicalCenter

### Comment:

Now we have created all the tables in our database.

## **2.2 Inserting data into tables**

**Code to insert data:**

```
INSERT INTO Rooms --
```

```
(RoomNumber)
```

```
VALUES
```

```
(101),(102),(103),(104),(105),
```

```
(201),(202),(203),(204),(205);
```

```
INSERT INTO Procedures --
```

```
(ProcedureName)
```

```
VALUES
```

```
('Surgery'),('Blood-Work'),('Stitching'),('Chemotherapy'),
```

```
('X-Ray'),('CT-Scan'),('MRI'),('C-Section');
```

```
INSERT INTO Medicines --
```

```
(MedicineName,Manufacturer)
```

```
VALUES
```

```
('Paracetamol', 'Johnson & Johnson'),
```

```
('Ibuprofen', 'Pfizer'),
```

```
('Aspirin', 'Johnson & Johnson'),
```

```
('Amoxicillin', 'Novartis'),
```

```
('Atorvastatin', 'Pfizer'),
```

```
('Metformin', 'AstraZeneca'),
```

```
('Omeprazole', 'AstraZeneca'),
```

```
('Simvastatin', 'Merck'),
```

```
('Levothyroxine','Pfizer'),
```

```
('Losartan', 'Merck');
```

```
INSERT INTO Specialities --
```

```
(Speciality)
```

```
VALUES
```

```
('Cardiology'),('Dermatology'),('Orthopedics'),
```

```
('Gastroenterology'),('Pediatrics'),('Neurology'),
```

```
('Ophthalmology'),('Oncology'),('Nephrology');
```

```
INSERT INTO Patients --
```

```
(FirstName, LastName, DateOfBirth, Gender)
```

```
VALUES
```

```
('Emma', 'Johnson', '1995-06-15', 'F'),  
('Olivia', 'Smith', '1992-11-27', 'F'),  
('Noah', 'Williams', '1990-09-08', 'M'),  
('Liam', 'Brown', '1991-04-03', 'M'),  
('Ava', 'Jones', '1976-02-21', 'F'),  
('Sophia', 'Davis', '1974-08-12', 'F'),  
('Mason', 'Miller', '1978-12-05', 'M'),  
('Ethan', 'Taylor', '1973-07-30', 'M'),  
('Isabella', 'Anderson', '1958-10-17', 'F'),  
('Charlotte', 'Thomas', '1962-03-25', 'F');
```

```
INSERT INTO PatientContact --
```

```
(PrimaryNumber, EmergencyNumber, Email)
```

```
VALUES
```

```
('123-457-8900', '987-154-3210', 'emma.johnson@example.com'),  
('231-567-1901', '826-543-2102', 'olivia.smith@example.com'),  
('325-678-5012', '765-432-1598', 'noah.williams@example.com'),  
('456-729-0143', '614-311-0987', 'liam.brown@example.com'),  
('557-890-1034', '543-210-9806', 'ava.jones@example.com'),  
('678-901-2355', '432-109-8565', 'sophia.davis@example.com'),  
('779-012-3456', '371-098-7654', 'mason.miller@example.com'),  
('890-163-4667', '260-987-6533', 'ethan.taylor@example.com'),  
('901-234-1178', '229-876-5332', 'isabella.anderson@example.com'),  
('212-345-6189', '898-765-4221', 'charlotte.thomas@example.com');
```

```
INSERT INTO PatientAddress --
```

```
(HouseNumber, Street, City, State, ZipCode)
```

```
VALUES
```

```
('123', 'Main Street', 'New York City', 'NY', '10001'),  
('456', 'Maple Avenue', 'Buffalo', 'NY', '14201'),  
('321', 'Elm Street', 'Rochester', 'NY', '14601'),  
('987', 'Birch Avenue', 'Syracuse', 'NY', '13201'),  
('543', 'Willow Lane', 'Albany', 'NY', '12201'),  
('246', 'Oak Street', 'New York City', 'NY', '10002'),  
('789', 'Cedar Avenue', 'Buffalo', 'NY', '14202'),
```

```
('654', 'Elm Court', 'Rochester', 'NY', '14602'),  
('321', 'Birch Road', 'Syracuse', 'NY', '13202'),  
('876', 'Willow Lane', 'Albany', 'NY', '12202');
```

**INSERT INTO** Nurses --

```
(DepartmentName, FirstName, LastName, HireDate, DOB)
```

**VALUES**

```
('Surgical-Assistant', 'Emily', 'Johnson', '2022-01-15', '1990-05-12'),  
('Emergency', 'Michael', 'Smith', '2021-08-02', '1988-11-23'),  
('Labor-Delivery', 'Sophia', 'Anderson', '2022-03-10', '1992-07-08'),  
('Clinical-Nurse', 'James', 'Brown', '2021-05-29', '1991-02-17'),  
('Labor-Delivery', 'Olivia', 'Taylor', '2022-06-18', '1993-09-06'),  
('Emergency', 'Jacob', 'Martinez', '2023-02-08', '1994-04-29'),  
('Surgical-Assistant', 'Isabella', 'Davis', '2022-11-14', '1990-12-03');
```

**INSERT INTO** NurseContact

```
(PrimaryNumber, EmergencyNumber, Email)
```

**VALUES**

```
('987-654-3770', '113-456-3890', 'emily.johnson@example.com'),  
('836-543-2109', '234-597-8801', 'michael.smith@example.com'),  
('735-432-1098', '305-678-9002', 'sophia.anderson@example.com'),  
('654-351-0587', '456-739-0103', 'james.brown@example.com'),  
('543-210-9676', '527-890-1214', 'olivia.taylor@example.com'),  
('462-169-8765', '618-901-2245', 'jacob.martinez@example.com'),  
('361-098-6654', '719-012-3156', 'isabella.davis@example.com');
```

**INSERT INTO** Insurances

```
(PatientID, Company, CoverageType)
```

**VALUES**

```
(1,'Blue Cross Blue Shield',  'Full'),  
(2,'UnitedHealth Group',      'Full'),  
(3,'Blue Cross Blue Shield',  'Full'),  
(4,'Cigna',                  'Partial'),  
(5,'UnitedHealth Group',      'Partial'),  
(6,'Aetna',                  'Full'),  
(7,'Aetna',                  'Full'),  
(8,'Aetna',                  'Full'),
```

```
(9,'Cigna',          'Partial'),  
(10,'Cigna',         'Partial');
```

```
INSERT INTO Physicians --  
(FirstName, LastName, DOB)  
VALUES  
('John', 'Smith', '1980-06-15'),  
('Emily', 'Johnson', '1985-09-23'),  
('Michael', 'Brown', '1976-11-30'),  
('Sophia', 'Anderson', '1990-04-05'),  
('David', 'Williams', '1982-02-18'),  
('Olivia', 'Taylor', '1988-07-12');
```

```
INSERT INTO PhysicianContact --  
(PrimaryNumber, EmergencyNumber, Email)  
VALUES  
(‘315-287-6543’, ‘912-456-7890’, ‘john.smith@example.com’),  
(‘233-187-0123’, ‘345-654-3210’, ‘emily.johnson@example.com’),  
(‘564-937-6541’, ‘247-234-5678’, ‘michael.brown@example.com’),  
(‘918-087-0123’, ‘936-456-7890’, ‘sophia.anderson@example.com’),  
(‘123-644-3210’, ‘432-886-6541’, ‘david.williams@example.com’),  
(‘907-664-5688’, ‘180-487-6543’, ‘olivia.taylor@example.com’);
```

```
INSERT INTO PhysicianLicense --  
(LicenseNumber)  
VALUES  
(9876543), (4567890), (9870123),  
(6543210), (9876541), (2345678);
```

```
INSERT INTO PhysicianSpeciality --  
(SpecialityID)  
VALUES  
(2),(1),(6),(5),(3),(7);
```

```
INSERT INTO Appointments --  
(PatientID, PhysicianID, AppointmentDate, AppointmentTime, Reason)  
VALUES
```

```
(2, 5, '2023-05-15', '10:00:00', 'Routine check-up'),  
(4, 2, '2023-05-16', '14:30:00', 'Follow-up for medication'),  
(6, 1, '2023-05-17', '09:15:00', 'Initial consultation'),  
(8, 5, '2023-05-19', '15:00:00', 'Annual physical'),  
(10, 5, '2023-05-21', '12:00:00', 'Orthopedic evaluation');
```

**INSERT INTO** Admissions --

```
(PatientID, PhysicianID, RoomID, AdmissionDate, DischargeDate, Diagnosis)
```

**VALUES**

```
(1, 6, 1, '2023-05-10', '2023-05-15', 'Cataract'),  
(3, 2, 10, '2023-05-12', NULL, 'Heart Blockages'),  
(5, 5, 2, '2023-05-15', '2023-05-17', 'Fractured Arm'),  
(7, 3, 4, '2023-05-18', NULL, 'Epilepsy'),  
(9, 2, 6, '2023-05-20', '2023-05-23', 'Arrhythmias');
```

**INSERT INTO** ProcedureDetails --

```
(ProcedureID, PatientID, PhysicianID, ProcedureDate, ProcedureNotes)
```

**VALUES**

```
(1, 1, 6, '2023-05-12', 'Remove cataract using laser'),  
(6, 3, 2, '2023-05-13', 'CT-Scan to check details'),  
(5, 5, 5, '2023-05-16', 'X-Ray to see the fracture'),  
(7, 7, 3, '2023-05-19', 'MRI scan of the brain'),  
(7, 9, 2, '2023-05-22', 'MRI scan of the heart');
```

**INSERT INTO** Notes --

```
(PatientID, PhysicianID, NurseID, NoteDetails)
```

**VALUES**

```
(1, 6, 1, 'Eye recovery is good'),  
(3, 2, 2, 'Heart condition getting better'),  
(5, 5, 2, 'Vital signs are normal'),  
(7, 3, 4, 'Medications are not showing improvement'),  
(9, 2, 6, 'Performed diagnostic tests. Results pending');
```

**INSERT INTO** Medications --

```
(PatientID, PhysicianID, MedicineID, Dosage, Frequency)
```

**VALUES**

```
(1, 6, 3, '3 Tabs' , '3x - 3 Days'),
```

```
(1, 6, 5, '2 Tabs' , '1x - 4 Days'),  
(2, 5, 2, '20 ml' , '2x - 2 Days'),  
(3, 2, 8, '1 Tab' , '1x - 6 Days'),  
(3, 2, 6, '30 ml' , '2x - 3 Days'),  
(4, 2, 1, '4 Tabs' , '2x - 3 Days'),  
(5, 5, 2, '20 ml' , '3x - 3 Days'),  
(6, 1, 7, '10 ml' , '1x - 5 Days'),  
(7, 3, 4, '2 Tabs' , '3x - 5 Days'),  
(7, 3, 9, '5 ml' , '2x - 7 Days'),  
(7, 3, 10,'3 Tabs' , '1x - 7 Days'),  
(8, 5, 2, '15 ml' , '3x - 3 Days'),  
(9, 2, 1, '2 Tabs' , '2x - 4 Days'),  
(10,5, 5, '4 Tabs' , '1x - 5 Days');
```

INSERT INTO PatientMedication --

(PatientID, MedicineID)

VALUES

```
(1,3), (1,5), (2,2), (3,8), (3,6), (4,1), (5,2),  
(6,7), (7,4), (7,9), (7,10), (8,2), (9,1), (10,5);
```

INSERT INTO LabResults --

(PatientID, PhysicianID, TestName, TestDate, TestResults)

VALUES

```
(1, 6, 'Complete Blood Count', '2023-05-11', 'Normal'),  
(3, 2, 'Lipid Profile', '2023-05-12', 'Cholesterol normal'),  
(5, 5, 'Glucose Test', '2023-05-15', 'Normal blood sugar'),  
(7, 3, 'Thyroid Function Test', '2023-05-18', 'Thyroid normal'),  
(9, 2, 'Liver Function Test', '2023-05-19', 'Liver enzyme normal'),  
(4, 2, 'Urinalysis', '2023-05-15', 'No abnormalities'),  
(6, 1, 'Electrolyte Panel', '2023-05-16', 'Electrolyte normal');
```

INSERT INTO Billings --

(PatientID, InsuranceID, TotalBill, InsuranceCoverage, PaidByPatient)

VALUES

```
(1, 1, 500, 500, 0),  
(2, 2, 1000, 1000, 0),  
(3, 3, 750, 750, 0),
```

```
(4, 4, 1200, 1000, 200),
(5, 5, 900, 700, 200),
(6, 6, 650, 650, 0),
(7, 7, 800, 800, 0),
(8, 8, 1100, 1100, 0),
(9, 9, 950, 550, 400),
(10, 10, 700, 500, 200);
```

### Screenshot of execution:

```

CONNECTIONS ... Create_Table.sql - localhost...er (sa) Insert_Data.sql - localhost...er (sa) X
▼ SERVERS ...
> localhost, <default> (root)
> localhost, <default> (sa)
> Databases
> Security
> Server Objects

Users > tamay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
Run Cancel ⚡ Disconnect Change Connection MedicalCenter Estimated Plan Enable Actual Plan Parse Enable SQLCMD Export as Notebook
224 (5, 5, 'Glucose Test', '2023-05-15', 'Normal blood sugar'),
225 (7, 3, 'Thyroid Function Test', '2023-05-18', 'Thyroid normal'),
226 (9, 2, 'Liver Function Test', '2023-05-19', 'Liver enzyme normal'),
227 (4, 2, 'Urinalysis', '2023-05-15', 'No abnormalities'),
228 (6, 1, 'Electrolyte Panel', '2023-05-16', 'Electrolyte normal');
229
230 INSERT INTO Billings --
231 (PatientID, InsuranceID, TotalBill, InsuranceCoverage, PaidByPatient)
232 VALUES
233 (1, 1, 500, 500, 0),
234 (2, 2, 1000, 1000, 0),
235 (3, 3, 750, 750, 0),
236 (4, 4, 1200, 1000, 200),
237 (5, 5, 900, 700, 200),
238 (6, 6, 650, 650, 0),
239 (7, 7, 800, 800, 0),
240 (8, 8, 1100, 1100, 0),
241 (9, 9, 950, 550, 400),
242 (10, 10, 700, 500, 200);

Messages
Started executing query at Line 1
(10 rows affected)
(8 rows affected)
(10 rows affected)
(9 rows affected)
(10 rows affected)
(10 rows affected)
(7 rows affected)
(7 rows affected)
(10 rows affected)
(8 rows affected)
(6 rows affected)
(6 rows affected)
(6 rows affected)
(6 rows affected)
(5 rows affected)
(5 rows affected)
(5 rows affected)
(5 rows affected)
(14 rows affected)
(14 rows affected)
(10 rows affected)
Total execution time: 00:00:00.088

PROBLEMS OUTPUT TERMINAL TASKS
MySQL Tools Service
Ln 244, Col 1 Spaces: 4 UTF-8 LF ( SQL 0 rows Choose SQL Language 00:00:00 localhost : MedicalCenter

```

### Comment:

Now the data is inserted in the database.

## 2.3 Observing tables

### 1. Patients Table

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel | ⚙ Disconnect ⚙ Change Connection MedicalCenter ▾ | ⚙ Estimate
247 |
248
249 SELECT * FROM Patients;
250
```

Results Messages

PatientID	FirstName	LastName	DateOfBirth	Gender
1	Emma	Johnson	1995-06-15	F
2	Olivia	Smith	1992-11-27	F
3	Noah	Williams	1990-09-08	M
4	Liam	Brown	1991-04-03	M
5	Ava	Jones	1976-02-21	F
6	Sophia	Davis	1974-08-12	F
7	Mason	Miller	1978-12-05	M
8	Ethan	Taylor	1973-07-30	M
9	Isabella	Anderson	1958-10-17	F
10	Charlotte	Thomas	1962-03-25	F

### 2. PatientContact Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel | ⚙ Disconnect ⚙ Change Connection MedicalCenter ▾ | ⚙ Estimated Plan E
247 |
248
249 SELECT * FROM PatientContact;
250
```

Results Messages

PatientID	PrimaryNumber	EmergencyNumber	Email
1	123-457-8900	987-154-3210	emma.johnson@example.com
2	231-567-1901	826-543-2102	olivia.smith@example.com
3	325-678-5012	765-432-1598	noah.williams@example.com
4	456-729-0143	614-311-0987	liam.brown@example.com
5	557-890-1834	543-210-9806	ava.jones@example.com
6	678-901-2355	432-109-8565	sophia.davis@example.com
7	779-012-3456	371-098-7654	mason.miller@example.com
8	890-163-4667	260-987-6533	ethan.taylor@example.com
9	901-234-1178	229-876-5332	isabella.anderson@example.com
10	212-345-6189	898-765-4221	charlotte.thomas@example.com

### 3. PatientAddress Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel | ⚙ Disconnect ⚙ Change Connection MedicalCenter ▾ | ⚙ Estimated Plan E
247 |
248
249 SELECT * FROM PatientAddress;
250
```

Results Messages

PatientID	HouseNumber	Street	City	State	ZipCode
1	123	Main Street	New York City	NY	10001
2	456	Maple Avenue	Buffalo	NY	14201
3	321	Elm Street	Rochester	NY	14601
4	987	Birch Avenue	Syracuse	NY	13201
5	543	Willow Lane	Albany	NY	12201
6	246	Oak Street	New York City	NY	10002
7	789	Cedar Avenue	Buffalo	NY	14202
8	654	Elm Court	Rochester	NY	14602
9	321	Birch Road	Syracuse	NY	13202
10	876	Willow Lane	Albany	NY	12202

#### 4. Insurances Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter ▾ ⌂ Esti
247
248
249   SELECT * FROM Insurances;
250

Results Messages


|    | InsuranceID | PatientID | Company                | CoverageType |
|----|-------------|-----------|------------------------|--------------|
| 1  | 1           | 1         | Blue Cross Blue Shield | Full         |
| 2  | 2           | 2         | UnitedHealth Group     | Full         |
| 3  | 3           | 3         | Blue Cross Blue Shield | Full         |
| 4  | 4           | 4         | Cigna                  | Partial      |
| 5  | 5           | 5         | UnitedHealth Group     | Partial      |
| 6  | 6           | 6         | Aetna                  | Full         |
| 7  | 7           | 7         | Aetna                  | Full         |
| 8  | 8           | 8         | Aetna                  | Full         |
| 9  | 9           | 9         | Cigna                  | Partial      |
| 10 | 10          | 10        | Cigna                  | Partial      |


```

#### 5. Billings Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter ▾ ⌂ Estimated Plan ⌂ Enable Actual Plan ✓ P
247
248
249   SELECT * FROM Billings;
250

Results Messages


|    | BillingID | PatientID | InsuranceID | TotalBill | InsuranceCoverage | PaidByPatient |
|----|-----------|-----------|-------------|-----------|-------------------|---------------|
| 1  | 1         | 1         | 1           | 500.00    | 500.00            | 0.00          |
| 2  | 2         | 2         | 2           | 1000.00   | 1000.00           | 0.00          |
| 3  | 3         | 3         | 3           | 750.00    | 750.00            | 0.00          |
| 4  | 4         | 4         | 4           | 1200.00   | 1000.00           | 200.00        |
| 5  | 5         | 5         | 5           | 900.00    | 700.00            | 200.00        |
| 6  | 6         | 6         | 6           | 650.00    | 650.00            | 0.00          |
| 7  | 7         | 7         | 7           | 800.00    | 800.00            | 0.00          |
| 8  | 8         | 8         | 8           | 1100.00   | 1100.00           | 0.00          |
| 9  | 9         | 9         | 9           | 950.00    | 550.00            | 400.00        |
| 10 | 10        | 10        | 10          | 700.00    | 500.00            | 200.00        |


```

#### 6. Physicians Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter ▾
248
249   SELECT * FROM Physicians;
250

Results Messages


|   | PhysicianID | FirstName | LastName | DOB        |
|---|-------------|-----------|----------|------------|
| 1 | 1           | John      | Smith    | 1980-06-15 |
| 2 | 2           | Emily     | Johnson  | 1985-09-23 |
| 3 | 3           | Michael   | Brown    | 1976-11-30 |
| 4 | 4           | Sophia    | Anderson | 1990-04-05 |
| 5 | 5           | David     | Williams | 1982-02-18 |
| 6 | 6           | Olivia    | Taylor   | 1988-07-12 |


```

## 7. PhysicianContact Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan E

248
249 SELECT * FROM PhysicianContact;
250
```

Results Messages

	PhysicianID	PrimaryNumber	EmergencyNumber	Email
1	1	315-287-6543	912-456-7890	john.smith@example.com
2	2	233-187-0123	345-654-3210	emily.johnson@example.com
3	3	564-937-6541	247-234-5678	michael.brown@example.com
4	4	918-087-0123	936-456-7890	sophia.anderson@example.com
5	5	123-644-3210	432-886-6541	david.williams@example.com
6	6	907-664-5688	180-487-6543	olivia.taylor@example.com

## 8. PhysicianLicense Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan E

248
249 SELECT * FROM PhysicianLicense;
250
```

Results Messages

	PhysicianID	LicenseNumber
1	1	9876543
2	2	4567890
3	3	9870123
4	4	6543210
5	5	9876541
6	6	2345678

## 9. Specialities Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan E

248
249 SELECT * FROM Specialities;
250
```

Results Messages

	SpecialityID	Speciality
1	1	Cardiology
2	2	Dermatology
3	3	Orthopedics
4	4	Gastroenterology
5	5	Pediatrics
6	6	Neurology
7	7	Ophthalmology
8	8	Oncology
9	9	Nephrology

## 10. PhysicianSpeciality Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > `Insert_Data.sql`

Run Cancel Disconnect Change Connection MedicalCenter

248  
249 `SELECT * FROM PhysicianSpeciality;`  
250

**Results** Messages

	PhysicianID	SpecialityID
1	1	2
2	2	1
3	3	6
4	4	5
5	5	3
6	6	7

## 11. Nurses Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > `Insert_Data.sql`

Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan Enabled

248  
249 `SELECT * FROM Nurses;`  
250

**Results** Messages

	NurseID	DepartmentName	FirstName	LastName	HireDate	DOB
1	1	Surgical-Assistant	Emily	Johnson	2022-01-15	1990-05-12
2	2	Emergency	Michael	Smith	2021-08-02	1988-11-23
3	3	Labor-Delivery	Sophia	Anderson	2022-03-10	1992-07-08
4	4	Clinical-Nurse	James	Brown	2021-05-29	1991-02-17
5	5	Labor-Delivery	Olivia	Taylor	2022-06-18	1993-09-06
6	6	Emergency	Jacob	Martinez	2023-02-08	1994-04-29
7	7	Surgical-Assistant	Isabella	Davis	2022-11-14	1990-12-03

## 12. NurseContact Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > `Insert_Data.sql`

Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan

248  
249 `SELECT * FROM NurseContact;`  
250

**Results** Messages

	NurseID	PrimaryNumber	EmergencyNumber	Email
1	1	987-654-3770	113-456-3890	emily.johnson@example.com
2	2	836-543-2109	234-597-8801	michael.smith@example.com
3	3	735-432-1098	305-678-9002	sophia.anderson@example.c...
4	4	654-351-0587	456-739-0103	james.brown@example.com
5	5	543-210-9676	527-890-1214	olivia.taylor@example.com
6	6	462-169-8765	618-901-2245	jacob.martinez@example.com
7	7	361-098-6654	719-012-3156	isabella.davis@example.com

### 13. Appointments Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⚙ Disconnect ☰ Change Connection MedicalCenter ▾ | ⚑ Estimated Plan ⚒ Enable Actual Plan ✓ Parse ⚓ Enable SQLCI
```

248  
249 `SELECT * FROM Appointments;`  
250

Results Messages

	AppointmentID	PatientID	PhysicianID	AppointmentDate	AppointmentTime	Reason
1	1	2	5	2023-05-15	10:00:00	Routine check-up
2	2	4	2	2023-05-16	14:30:00	Follow-up for medication
3	3	6	1	2023-05-17	09:15:00	Initial consultation
4	4	8	5	2023-05-19	15:00:00	Annual physical
5	5	10	5	2023-05-21	12:00:00	Orthopedic evaluation

### 14. Rooms Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⚙ Disconnect ☰ Change Connection MedicalCenter ▾
```

248  
249 `SELECT * FROM Rooms;`  
250

Results Messages

	RoomID	RoomNumber
1	1	101
2	2	102
3	3	103
4	4	104
5	5	105
6	6	201
7	7	202
8	8	203
9	9	204
10	10	205

### 15. Admissions Table:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Insert_Data.sql
▶ Run □ Cancel ⚙ Disconnect ☰ Change Connection MedicalCenter ▾ | ⚑ Estimated Plan ⚒ Enable Actual Plan ✓ Parse ⚓ Enable SC
```

248  
249 `SELECT * FROM Admissions;`  
250

Results Messages

	AdmissionID	PatientID	PhysicianID	RoomID	AdmissionDate	DischargeDate	Diagnosis
1	1	1	6	1	2023-05-10	2023-05-15	Cataract
2	2	3	2	10	2023-05-12	NULL	Heart Blockages
3	3	5	5	2	2023-05-15	2023-05-17	Fractured Arm
4	4	7	3	4	2023-05-18	NULL	Epilepsy
5	5	9	2	6	2023-05-20	2023-05-23	Arrhythmias

### 16. Procedures Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⚙ Disconnect ⚙ Change Connection MedicalCenter

```
248
249  SELECT * FROM Procedures;
250
```

Results Messages

	ProcedureID	ProcedureName
1	1	Surgery
2	2	Blood-Work
3	3	Stitching
4	4	Chemotherapy
5	5	X-Ray
6	6	CT-Scan
7	7	MRI
8	8	C-Section

### 17. ProcedureDetails Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⚙ Disconnect ⚙ Change Connection MedicalCenter

Estimated Plan  Enable Actual Plan  Parse  En

```
248
249  SELECT * FROM ProcedureDetails;
250
```

Results Messages

	DetailID	ProcedureID	PatientID	PhysicianID	ProcedureDate	ProcedureNotes
1	1	1	1	6	2023-05-12	Remove cataract using las...
2	2	6	3	2	2023-05-13	CT-Scan to check details
3	3	5	5	5	2023-05-16	X-Ray to see the fracture
4	4	7	7	3	2023-05-19	MRI scan of the brain
5	5	7	9	2	2023-05-22	MRI scan of the heart

### 18. Medicines Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⚙ Disconnect ⚙ Change Connection MedicalCenter

```
248
249  SELECT * FROM Medicines;
250
```

Results Messages

	MedicineID	MedicineName	Manufacturer
1	1	Paracetamol	Johnson & Johnson
2	2	Ibuprofen	Pfizer
3	3	Aspirin	Johnson & Johnson
4	4	Amoxicillin	Novartis
5	5	Atorvastatin	Pfizer
6	6	Metformin	AstraZeneca
7	7	Omeprazole	AstraZeneca
8	8	Simvastatin	Merck
9	9	Levothyroxine	Pfizer
10	10	Losartan	Merck

### 19. Medications Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter Estimated Plan ⌂ Enable Act

248  
249 `SELECT * FROM Medications;`  
250

Results Messages

MedicationID	PatientID	PhysicianID	MedicineID	Dosage	Frequency
1	1	6	3	3 Tabs	3x - 3 Days
2	2	1	6	2 Tabs	1x - 4 Days
3	3	2	5	20 ml	2x - 2 Days
4	4	3	2	1 Tab	1x - 6 Days
5	5	3	2	30 ml	2x - 3 Days
6	6	4	2	4 Tabs	2x - 3 Days
7	7	5	5	20 ml	3x - 3 Days
8	8	6	1	10 ml	1x - 5 Days
9	9	7	3	2 Tabs	3x - 5 Days
10	10	7	3	5 ml	2x - 7 Days
11	11	7	3	3 Tabs	1x - 7 Days
12	12	8	5	15 ml	3x - 3 Days
13	13	9	2	2 Tabs	2x - 4 Days
14	14	10	5	4 Tabs	1x - 5 Days

### 20. Notes Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter Estimated Plan ⌂ E

248  
249 `SELECT * FROM Notes;`  
250

Results Messages

NoteID	PatientID	PhysicianID	NurseID	NoteDetails
1	1	6	1	Eye recovery is good
2	2	3	2	Heart condition getting b...
3	3	5	2	Vital signs are normal
4	4	7	3	Medications are not showi...
5	5	9	2	Performed diagnostic test...

### 21. LabResults Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Insert\_Data.sql

Run Cancel ⌂ Disconnect ⌂ Change Connection MedicalCenter Estimated Plan ⌂ Enable Actual Plan Parse

248  
249 `SELECT * FROM LabResults;`  
250

Results Messages

LabResultID	PatientID	PhysicianID	TestName	TestDate	TestResults
1	1	6	Complete Blood Count	2023-05-11	Normal
2	2	3	Lipid Profile	2023-05-12	Cholesterol normal
3	3	5	Glucose Test	2023-05-15	Normal blood sugar
4	4	7	Thyroid Function Test	2023-05-18	Thyroid normal
5	5	9	Liver Function Test	2023-05-19	Liver enzyme normal
6	6	4	Urinalysis	2023-05-15	No abnormalities
7	7	6	Electrolyte Panel	2023-05-16	Electrolyte normal

## 22. PatientMedication Table:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > `Insert_Data.sql`

Run Cancel Disconnect Change Connection MedicalCenter

```
248
249  SELECT * FROM PatientMedication;
250
```

Results Messages

	PatientID	MedicineID
1	1	3
2	1	5
3	2	2
4	3	8
5	3	6
6	4	1
7	5	2
8	6	7
9	7	4
10	7	9
11	7	10
12	8	2
13	9	1
14	10	5

Here we can see all the values in table that were inserted.

## 3. Testing Phase

Here in this section we will be creating Views, Stored Procedures, Functions, Triggers, Transaction and Scripts to test how everything is working with this database model.

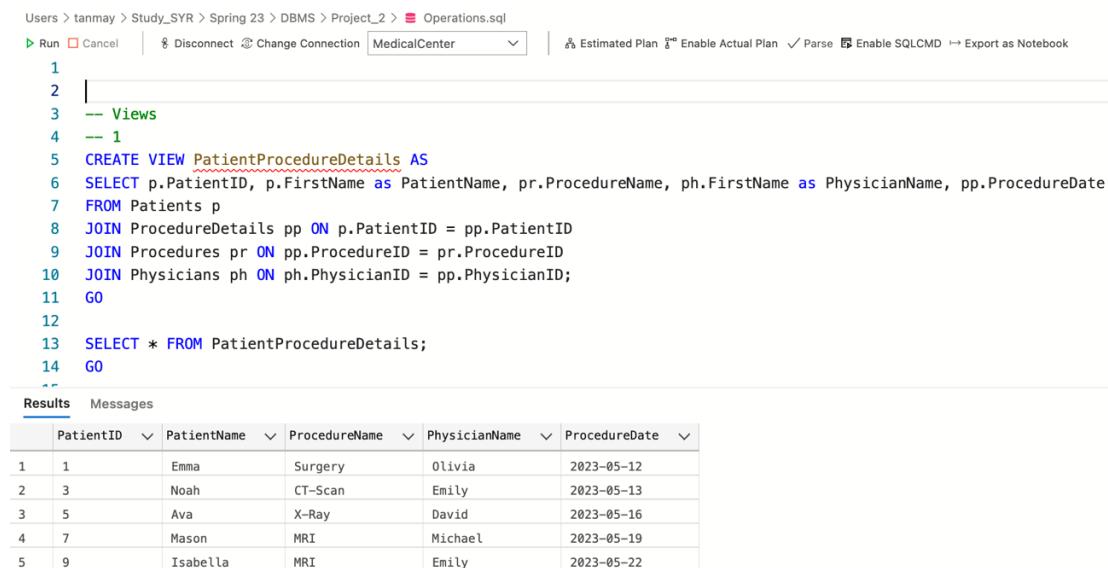
### 3.1 Views

#### 1. PatientProcedureDetails

**Code:**

```
CREATE VIEW PatientProcedureDetails AS
SELECT p.PatientID, p.FirstName as PatientName, pr.ProcedureName, ph.FirstName as PhysicianName,
pp.ProcedureDate
FROM Patients p
JOIN ProcedureDetails pp ON p.PatientID = pp.PatientID
JOIN Procedures pr ON pp.ProcedureID = pr.ProcedureID
JOIN Physicians ph ON ph.PhysicianID = pp.PhysicianID;
GO
SELECT * FROM PatientProcedureDetails;
GO
```

**Execution:**



The screenshot shows the SQL Server Management Studio interface. The query window contains the SQL code for creating the view. The results tab displays a table with the following data:

	PatientID	PatientName	ProcedureName	PhysicianName	ProcedureDate
1	1	Emma	Surgery	Olivia	2023-05-12
2	3	Noah	CT-Scan	Emily	2023-05-13
3	5	Ava	X-Ray	David	2023-05-16
4	7	Mason	MRI	Michael	2023-05-19
5	9	Isabella	MRI	Emily	2023-05-22

**Comment:**

This view gives the details about the patients who are having a procedure along with the physician's name along with the procedure date.

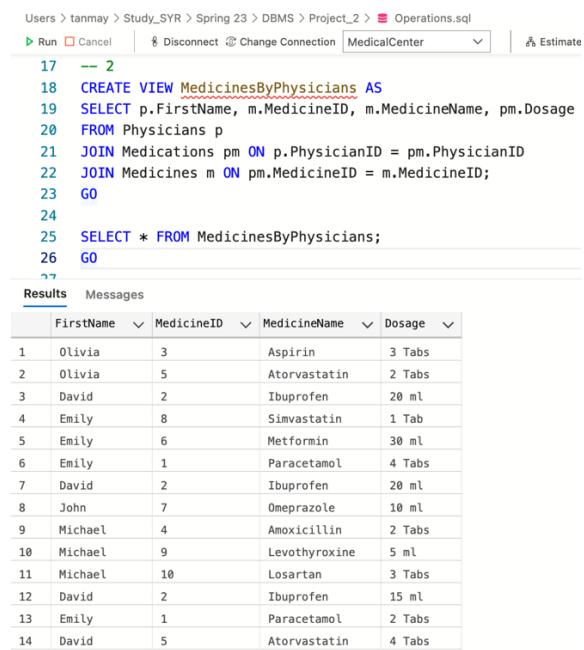
## 2. MedicinesByPhysicians

**Code:**

```
CREATE VIEW MedicinesByPhysicians AS
SELECT p.FirstName, m.MedicineID, m.MedicineName, pm.Dosage
FROM Physicians p
JOIN Medications pm ON p.PhysicianID = pm.PhysicianID
JOIN Medicines m ON pm.MedicineID = m.MedicineID;
GO

SELECT * FROM MedicinesByPhysicians;
GO
```

**Execution:**



The screenshot shows a SQL query editor window with the following details:

- Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar buttons: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimate.
- Code area:

```
17  -- 2
18  CREATE VIEW MedicinesByPhysicians AS
19  SELECT p.FirstName, m.MedicineID, m.MedicineName, pm.Dosage
20  FROM Physicians p
21  JOIN Medications pm ON p.PhysicianID = pm.PhysicianID
22  JOIN Medicines m ON pm.MedicineID = m.MedicineID;
23  GO
24
25  SELECT * FROM MedicinesByPhysicians;
26  GO
27
```
- Results tab: Displays the output of the query, which is a table with columns FirstName, MedicineID, MedicineName, and Dosage. The data is as follows:

	FirstName	MedicineID	MedicineName	Dosage
1	Olivia	3	Aspirin	3 Tabs
2	Olivia	5	Atorvastatin	2 Tabs
3	David	2	Ibuprofen	20 ml
4	Emily	8	Simvastatin	1 Tab
5	Emily	6	Metformin	30 ml
6	Emily	1	Paracetamol	4 Tabs
7	David	2	Ibuprofen	20 ml
8	John	7	Omeprazole	10 ml
9	Michael	4	Amoxicillin	2 Tabs
10	Michael	9	Levothyroxine	5 ml
11	Michael	10	Losartan	3 Tabs
12	David	2	Ibuprofen	15 ml
13	Emily	1	Paracetamol	2 Tabs
14	David	5	Atorvastatin	4 Tabs

**Comment:**

This view gives the details about the medicines given by physicians along with the dosage that was prescribed to the patients.

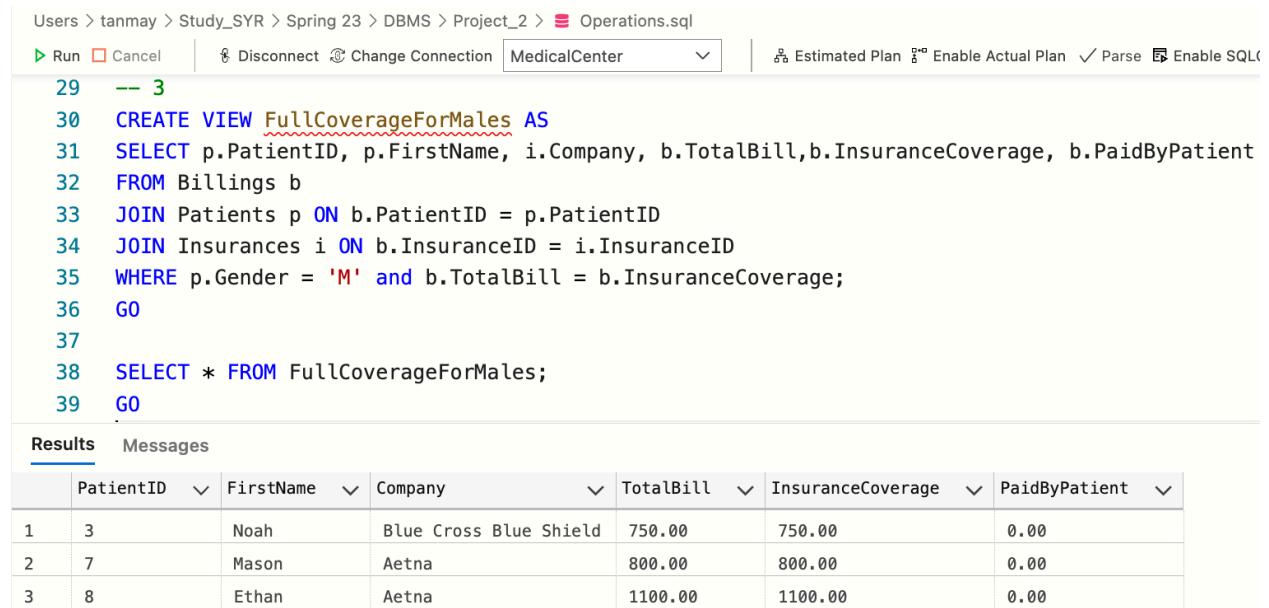
### 3. FullCoverageForMales

Code:

```
CREATE VIEW FullCoverageForMales AS
SELECT p.PatientID, p.FirstName, i.Company, b.TotalBill,b.InsuranceCoverage, b.PaidByPatient
FROM Billings b
JOIN Patients p ON b.PatientID = p.PatientID
JOIN Insurances i ON b.InsuranceID = i.InsuranceID
WHERE p.Gender = 'M' and b.TotalBill = b.InsuranceCoverage;
GO

SELECT * FROM FullCoverageForMales;
GO
```

Execution:



The screenshot shows the SQL query being run in SSMS. The code is identical to the one above. The results pane shows the following data:

	PatientID	FirstName	Company	TotalBill	InsuranceCoverage	PaidByPatient
1	3	Noah	Blue Cross Blue Shield	750.00	750.00	0.00
2	7	Mason	Aetna	800.00	800.00	0.00
3	8	Ethan	Aetna	1100.00	1100.00	0.00

Comments:

This view gives the billing details for 'Male' patients for whom the insurance company covered the entire medical expense.

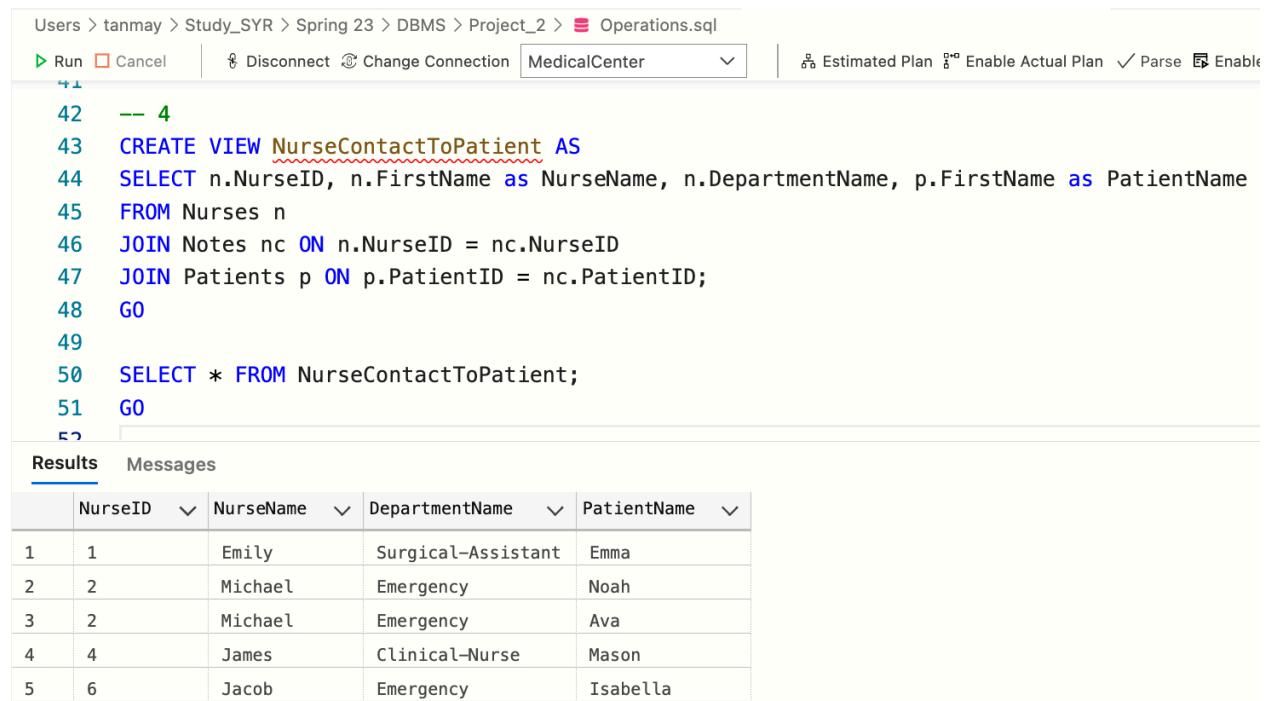
#### 4. NurseContactToPatient

Code:

```
CREATE VIEW NurseContactToPatient AS
SELECT n.NurseID, n.FirstName as NurseName, n.DepartmentName, p.FirstName as PatientName
FROM Nurses n
JOIN Notes nc ON n.NurseID = nc.NurseID
JOIN Patients p ON p.PatientID = nc.PatientID;
GO

SELECT * FROM NurseContactToPatient;
GO
```

Execution:



```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Operations.sql
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan Enable Actual Plan Parse Enable
+-
42 -- 4
43 CREATE VIEW NurseContactToPatient AS
44 SELECT n.NurseID, n.FirstName as NurseName, n.DepartmentName, p.FirstName as PatientName
45 FROM Nurses n
46 JOIN Notes nc ON n.NurseID = nc.NurseID
47 JOIN Patients p ON p.PatientID = nc.PatientID;
48 GO
49
50 SELECT * FROM NurseContactToPatient;
51 GO
52
```

Results Messages

	NurseID	NurseName	DepartmentName	PatientName
1	1	Emily	Surgical-Assistant	Emma
2	2	Michael	Emergency	Noah
3	2	Michael	Emergency	Ava
4	4	James	Clinical-Nurse	Mason
5	6	Jacob	Emergency	Isabella

Comment:

This view gives the nurse details like their name and department for the patients they were assigned to.

## **3.2 Stored Procedures**

### **1. InsertPatientDetails**

**Code:**

```
CREATE PROCEDURE InsertPatientDetails
(
    @FirstName VARCHAR(20),@LastName VARCHAR(20),@DateOfBirth DATE,@Gender VARCHAR(1),
    @PNumber VARCHAR(15),@ENumber VARCHAR(15),@Email VARCHAR(30),
    @HouseNum VARCHAR(4),@Street VARCHAR(20),@City VARCHAR(20),@State
    VARCHAR(20),@ZipCode VARCHAR(6)
)
AS
BEGIN
    INSERT INTO Patients (FirstName, LastName, DateOfBirth, Gender)
    VALUES (@FirstName, @LastName, @DateOfBirth, @Gender)
    INSERT INTO PatientContact(PrimaryNumber, EmergencyNumber, Email)
    VALUES (@PNumber, @ENumber, @Email)
    INSERT INTO PatientAddress (HouseNumber, Street, City, State, ZipCode)
    VALUES (@HouseNum, @Street, @City, @State, @ZipCode)
END
GO

EXEC InsertPatientDetails
    @FirstName = 'John', @LastName = 'Jill', @DateOfBirth = '1990-12-12', @Gender = 'M',
    @PNumber = '773-456-7898', @ENumber = '985-654-3217', @Email = 'john.jill@example.com',
    @HouseNum = '993', @Street = 'Main Street', @City = 'New York', @State = 'NY', @ZipCode = '15645';
GO

SELECT TOP 1 * FROM Patients ORDER BY PatientID DESC
SELECT TOP 1 * FROM PatientContact ORDER BY PatientID DESC;
SELECT TOP 1 * FROM PatientAddress ORDER BY PatientID DESC;
GO
```

## Execution:

```
Users > tanmay > Study_SYR > Spring 23 > DBMS > Project_2 > Operations.sql
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan Enable Actual Plan Parse Enable SQLCMD Export a...
56  -- Procedures
57  -- 1
58  CREATE PROCEDURE InsertPatientDetails
59  (
60      @FirstName VARCHAR(20), @LastName VARCHAR(20), @DateOfBirth DATE, @Gender VARCHAR(1),
61      @PNumber VARCHAR(15), @ENumber VARCHAR(15), @Email VARCHAR(30),
62      @HouseNum VARCHAR(4), @Street VARCHAR(20), @City VARCHAR(20), @State VARCHAR(20), @ZipCode VARCHAR(6)
63  )
64  AS
65  BEGIN
66      INSERT INTO Patients (FirstName, LastName, DateOfBirth, Gender)
67      VALUES (@FirstName, @LastName, @DateOfBirth, @Gender)
68      INSERT INTO PatientContact(PrimaryNumber, EmergencyNumber, Email)
69      VALUES (@PNumber, @ENumber, @Email)
70      INSERT INTO PatientAddress (HouseNumber, Street, City, State, ZipCode)
71      VALUES (@HouseNum, @Street, @City, @State, @ZipCode)
72  END
73  GO
74
75  EXEC InsertPatientDetails
76  @FirstName = 'John', @LastName = 'Jill', @DateOfBirth = '1990-12-12', @Gender = 'M',
77  @PNumber = '773-456-7898', @ENumber = '985-654-3217', @Email = 'john.jill@example.com',
78  @HouseNum = '993', @Street = 'Main Street', @City = 'New York', @State = 'NY', @ZipCode = '15645';
79  GO
80
81  SELECT TOP 1 * FROM Patients ORDER BY PatientID DESC
82  SELECT TOP 1 * FROM PatientContact ORDER BY PatientID DESC;
83  SELECT TOP 1 * FROM PatientAddress ORDER BY PatientID DESC;
84  GO
```

Results Messages

	PatientID	FirstName	LastName	DateOfBirth	Gender
1	10	Charlotte	Thomas	1962-03-25	F

	PatientID	PrimaryNumber	EmergencyNumber	Email
1	11	773-456-7898	985-654-3217	john.jill@example.com

	PatientID	HouseNumber	Street	City	State	ZipCode
1	11	993	Main Street	New York	NY	15645

## Comment:

This stored procedure is used to insert patient details into Patients, PatientAddress and PatientContact table. The SELECT query which is used displays the latest values that was inserted in the table. Here we can see that PatientID is now 11 after inserting new values (initially there were only 10 rows in this table).

## 2. UpdatePhysicianContact

Code:

```
CREATE PROCEDURE UpdatePhysicianContact
(
    @PhysicianID INT,
    @PNumber VARCHAR(15),
    @ENumber VARCHAR(15),
    @Email VARCHAR(30)
)
AS
BEGIN
    UPDATE PhysicianContact
    SET PrimaryNumber = @PNumber,
        EmergencyNumber = @ENumber,
        Email = @Email
    WHERE PhysicianID = @PhysicianID
END
GO

EXEC UpdatePhysicianContact
    @PhysicianID = 4, @PNumber = '918-087-0222', @ENumber = '936-456-7899', @Email =
'sophia.anderson.new@example.com';
GO

SELECT * FROM PhysicianContact;
GO
```

## Execution:

The screenshot shows a SQL query window in SSMS. The code is a stored procedure named 'UpdatePhysicianContact' that updates the 'PhysicianContact' table. It takes four parameters: PhysicianID (INT), PNumber (VARCHAR(15)), ENumber (VARCHAR(15)), and Email (VARCHAR(30)). The procedure uses an UPDATE statement to set the PrimaryNumber, EmergencyNumber, and Email fields based on the provided parameters, where the PhysicianID matches the parameter. It then EXECutes the procedure with PhysicianID = 4 and provides sample values for PNumber, ENumber, and Email. Finally, it SELECTs all columns from the PhysicianContact table.

```
86 -- 2
87 CREATE PROCEDURE UpdatePhysicianContact
88 (
89     @PhysicianID INT,
90     @PNumber VARCHAR(15),
91     @ENumber VARCHAR(15),
92     @Email VARCHAR(30)
93 )
94 AS
95 BEGIN
96     UPDATE PhysicianContact
97     SET PrimaryNumber = @PNumber,
98         EmergencyNumber = @ENumber,
99         Email = @Email
100    WHERE PhysicianID = @PhysicianID
101 END
102 GO
103
104 EXEC UpdatePhysicianContact
105     @PhysicianID = 4, @PNumber = '918-087-0222', @ENumber = '936-456-7899', @Email = 'sophia.anderson.new@example.com';
106 GO
107
108 SELECT * FROM PhysicianContact;
109 GO
```

**Results** Messages

	PhysicianID	PrimaryNumber	EmergencyNumber	Email
1	1	315-287-6543	912-456-7890	john.smith@example.com
2	2	233-187-0123	345-654-3210	emily.johnson@example.com
3	3	564-937-6541	247-234-5678	michael.brown@example.com
4	4	918-087-0222	936-456-7899	sophia.anderson.new@exampl...
5	5	123-644-3210	432-886-6541	david.williams@example.com
6	6	907-664-5688	180-487-6543	olivia.taylor@example.com

## Comment:

In this procedure we update the physician's contact details. We can see that the details for physician with PhysicianID = 4 have been updated.

### 3. DeleteSpeciality

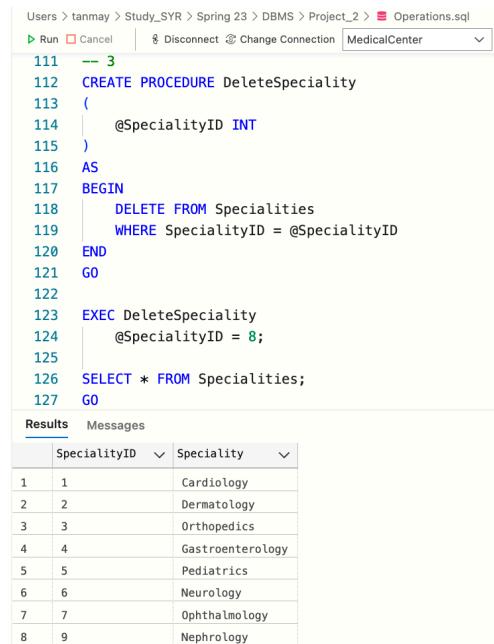
**Code:**

```
CREATE PROCEDURE DeleteSpeciality
(
    @SpecialityID INT
)
AS
BEGIN
    DELETE FROM Specialities
    WHERE SpecialityID = @SpecialityID
END
GO

EXEC DeleteSpeciality
    @SpecialityID = 8;

SELECT * FROM Specialities;
GO
```

**Execution:**



The screenshot shows the SQL Server Management Studio interface with the following details:

- Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter)
- Code pane:

```
111 -- 3
112 CREATE PROCEDURE DeleteSpeciality
113 (
114     @SpecialityID INT
115 )
116 AS
117 BEGIN
118     DELETE FROM Specialities
119     WHERE SpecialityID = @SpecialityID
120 END
121 GO
122
123 EXEC DeleteSpeciality
124     @SpecialityID = 8;
125
126 SELECT * FROM Specialities;
127 GO
```
- Results pane: Shows the output of the SELECT query, displaying the following data:

	SpecialityID	Speciality
1	1	Cardiology
2	2	Dermatology
3	3	Orthopedics
4	4	Gastroenterology
5	5	Pediatrics
6	6	Neurology
7	7	Ophthalmology
8	9	Nephrology

**Comment:**

This procedure deletes the speciality from the Specialities table.

#### 4. GetMedicineGivenDetails

**Code:**

```
CREATE PROCEDURE GetMedicineGivenDetails
(
    @MedicineID INT
)
AS
BEGIN
    SELECT m.MedicineID, m.MedicineName, me.Dosage, me.Frequency
    FROM Medicines m
    JOIN Medications me ON m.MedicineID = me.MedicineID
    WHERE m.MedicineID = @MedicineID
END
GO
EXEC GetMedicineGivenDetails
@MedicineID = 5;
EXEC GetMedicineGivenDetails
@MedicineID = 2;
```

**Execution:**

```
129 -- 4
130 CREATE PROCEDURE GetMedicineGivenDetails
131 (
132     @MedicineID INT
133 )
134 AS
135 BEGIN
136     SELECT m.MedicineID, m.MedicineName, me.Dosage, me.Frequency
137     FROM Medicines m
138     JOIN Medications me ON m.MedicineID = me.MedicineID
139     WHERE m.MedicineID = @MedicineID
140 END
141 GO
142
143 EXEC GetMedicineGivenDetails
144     @MedicineID = 5;
145
146 EXEC GetMedicineGivenDetails
147     @MedicineID = 2;
148 GO
```

Results

	MedicineID	MedicineName	Dosage	Frequency
1	5	Atorvastatin	2 Tabs	1x - 4 Days
2	5	Atorvastatin	4 Tabs	1x - 5 Days

	MedicineID	MedicineName	Dosage	Frequency
1	2	Ibuprofen	20 ml	2x - 2 Days
2	2	Ibuprofen	20 ml	3x - 3 Days
3	2	Ibuprofen	15 ml	3x - 3 Days

**Comment:**

This procedure displays the medicines that was prescribed to patients along with the dosage and frequency.

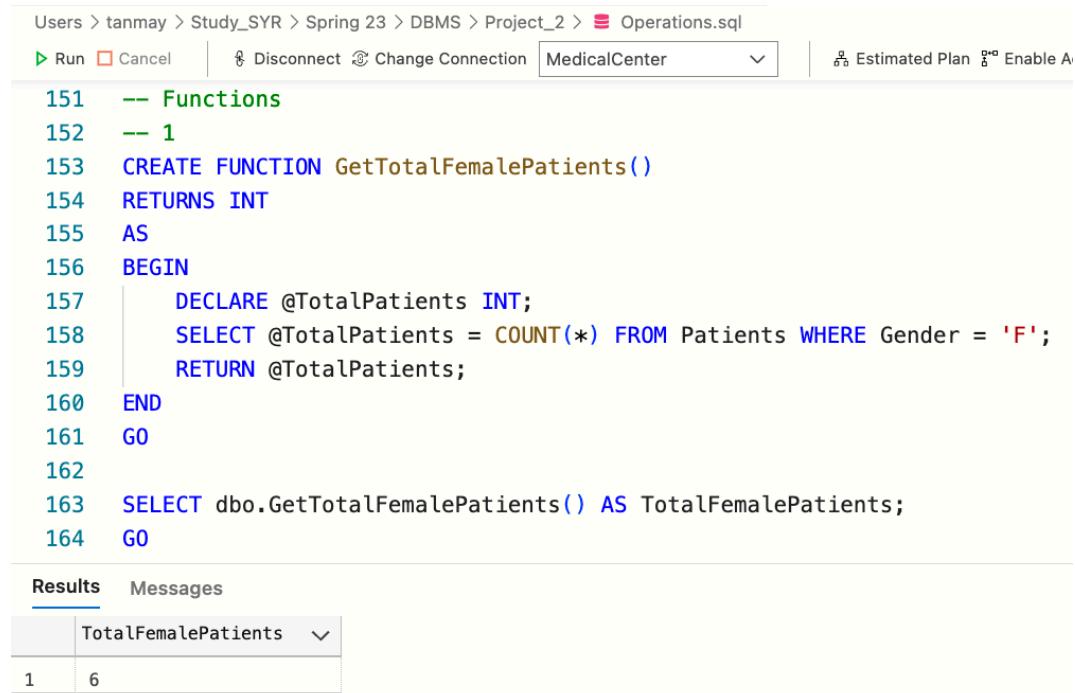
### 3.3 User-defined Functions

#### 1. GetTotalFemalePatients

**Code:**

```
CREATE FUNCTION GetTotalFemalePatients()
RETURNS INT
AS
BEGIN
    DECLARE @TotalPatients INT;
    SELECT @TotalPatients = COUNT(*) FROM Patients WHERE Gender = 'F';
    RETURN @TotalPatients;
END
GO
SELECT dbo.GetTotalFemalePatients() AS TotalFemalePatients;
GO
```

**Execution:**



The screenshot shows a SQL query editor interface with the following details:

- File Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar Buttons: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable A... (disabled).
- Code Area (Listed Lines):
  - 151 -- Functions
  - 152 -- 1
  - 153 CREATE FUNCTION GetTotalFemalePatients()
  - 154 RETURNS INT
  - 155 AS
  - 156 BEGIN
  - 157 DECLARE @TotalPatients INT;
  - 158 SELECT @TotalPatients = COUNT(\*) FROM Patients WHERE Gender = 'F';
  - 159 RETURN @TotalPatients;
  - 160 END
  - 161 GO
  - 162
  - 163 SELECT dbo.GetTotalFemalePatients() AS TotalFemalePatients;
  - 164 GO
- Results Tab: Shows a table with one row:

	TotalFemalePatients
1	6
- Messages Tab: Not visible.

**Comment:**

This function displays the total female patients in the database.

## 2. GetMedicineDosage

**Code:**

```
CREATE FUNCTION GetMedicineDosage(@MedicineID INT)
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @Dosage VARCHAR(50);
    SELECT @Dosage = Dosage FROM Medications WHERE MedicineID = @MedicineID;
    RETURN @Dosage;
END
GO
SELECT dbo.GetMedicineDosage(4) AS MedicineDosage;
GO
```

**Execution:**

The screenshot shows the SSMS interface with the following details:

- Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan.
- Code Editor (Listed Lines):
  - 167 -- 2
  - 168 CREATE FUNCTION GetMedicineDosage(@MedicineID INT)
  - 169 RETURNS VARCHAR(50)
  - 170 AS
  - 171 BEGIN
  - 172 DECLARE @Dosage VARCHAR(50);
  - 173 SELECT @Dosage = Dosage FROM Medications WHERE MedicineID = @MedicineID;
  - 174 RETURN @Dosage;
  - 175 END
  - 176 GO
  - 177
  - 178 SELECT dbo.GetMedicineDosage(4) AS MedicineDosage;
  - 179 GO
  - 180 |
- Results Tab: Shows a single row in the results grid:

MedicineDosage
1
- Messages Tab: Shows "2 Tabs".

**Comment:**

This function gives the medicine dosage prescribed for the selected medicine.

### 3. GetPatientAge

**Code:**

```
CREATE FUNCTION GetPatientAge(@PatientID INT)
RETURNS INT
AS
BEGIN
    DECLARE @Age INT;
    SELECT @Age = DATEDIFF(YEAR, DateOfBirth, GETDATE()) FROM Patients WHERE PatientID =
@PatientID;
    RETURN @Age;
END
GO
SELECT dbo.GetPatientAge(6) AS PatientAge;
GO
```

**Execution:**

The screenshot shows the SSMS interface with the following details:

- Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan, Parse, Enable SQLCMD, Export.
- Code Area (Lines 182-195):

```
-- 3
CREATE FUNCTION GetPatientAge(@PatientID INT)
RETURNS INT
AS
BEGIN
    DECLARE @Age INT;
    SELECT @Age = DATEDIFF(YEAR, DateOfBirth, GETDATE()) FROM Patients WHERE PatientID = @PatientID;
    RETURN @Age;
END
GO
SELECT dbo.GetPatientAge(6) AS PatientAge;
GO
```
- Results Tab: Shows a single row of results:

PatientAge
1
49

**Comment:**

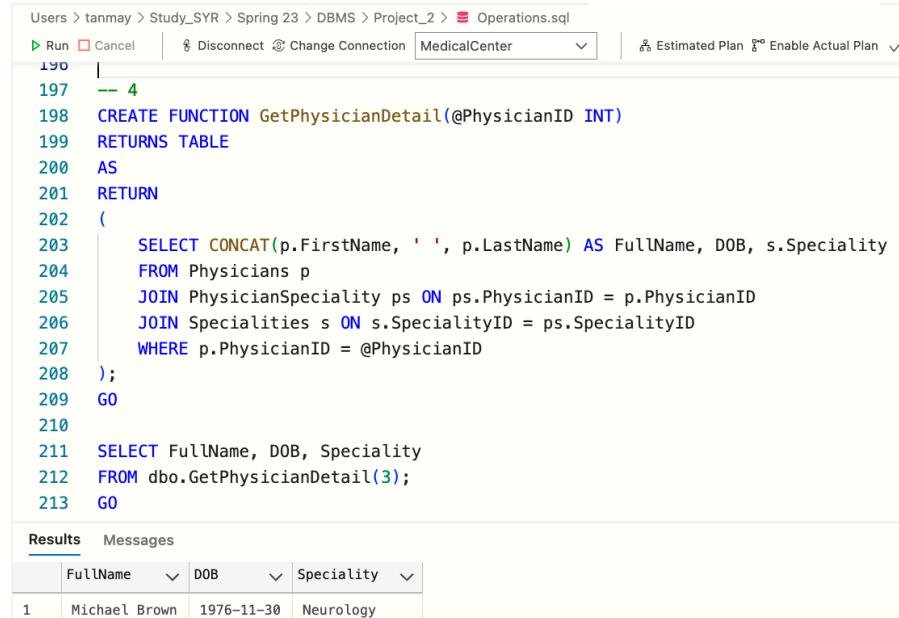
This function gives the current age for the patient based on their date of birth for the given PatientID.

#### 4. GetPhysicianDetails

**Code:**

```
CREATE FUNCTION GetPhysicianDetail(@PhysicianID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT CONCAT(p.FirstName, ' ', p.LastName) AS FullName, DOB, s.Speciality
    FROM Physicians p
    JOIN PhysicianSpeciality ps ON ps.PhysicianID = p.PhysicianID
    JOIN Specialities s ON s.SpecialityID = ps.SpecialityID
    WHERE p.PhysicianID = @PhysicianID
);
GO
SELECT FullName, DOB, Speciality
FROM dbo.GetPhysicianDetail(3);
GO
```

**Execution:**



The screenshot shows the SQL Server Management Studio interface with the following details:

- File Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar Buttons: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan.
- Code Area:

```
190
191
192 CREATE FUNCTION GetPhysicianDetail(@PhysicianID INT)
193 RETURNS TABLE
194 AS
195 RETURN
196 (
197     SELECT CONCAT(p.FirstName, ' ', p.LastName) AS FullName, DOB, s.Speciality
198     FROM Physicians p
199     JOIN PhysicianSpeciality ps ON ps.PhysicianID = p.PhysicianID
200     JOIN Specialities s ON s.SpecialityID = ps.SpecialityID
201     WHERE p.PhysicianID = @PhysicianID
202 );
203 GO
204
205
206
207
208
209
210
211 SELECT FullName, DOB, Speciality
212 FROM dbo.GetPhysicianDetail(3);
213 GO
```
- Results Tab: Shows the output of the query in a table format.

	FullName	DOB	Speciality
1	Michael Brown	1976-11-30	Neurology

**Comment:**

This function returns a table returning the full name, date of birth and speciality of a physician for a given PhysicianID.

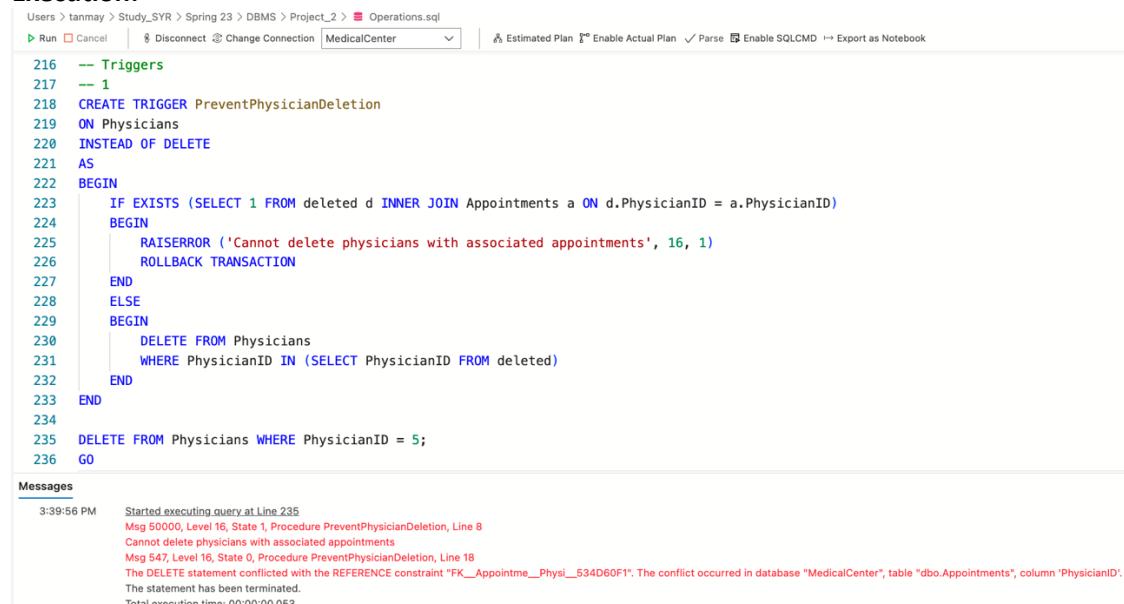
## 3.4 Triggers

### 1. PreventPhysicianDeletion

Code:

```
CREATE TRIGGER PreventPhysicianDeletion
ON Physicians
INSTEAD OF DELETE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Appointments a ON d.PhysicianID = a.PhysicianID)
        BEGIN
            RAISERROR ('Cannot delete physicians with associated appointments', 16, 1)
            ROLLBACK TRANSACTION
        END
    ELSE
        BEGIN
            DELETE FROM Physicians
            WHERE PhysicianID IN (SELECT PhysicianID FROM deleted)
        END
    END
DELETE FROM Physicians WHERE PhysicianID = 5;
```

#### Execution:



The screenshot shows the SSMS interface with the following details:

- File Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar Buttons: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan, Parse, Enable SQLCMD, Export as Notebook.
- Script Content (Lines 216-236):

```
216 -- Triggers
217 -- 1
218 CREATE TRIGGER PreventPhysicianDeletion
219 ON Physicians
220 INSTEAD OF DELETE
221 AS
222 BEGIN
223     IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Appointments a ON d.PhysicianID = a.PhysicianID)
224         BEGIN
225             RAISERROR ('Cannot delete physicians with associated appointments', 16, 1)
226             ROLLBACK TRANSACTION
227         END
228     ELSE
229         BEGIN
230             DELETE FROM Physicians
231             WHERE PhysicianID IN (SELECT PhysicianID FROM deleted)
232         END
233     END
234
235 DELETE FROM Physicians WHERE PhysicianID = 5;
236 GO
```
- Messages Log:
  - 3:39:56 PM Started executing query at Line 235
  - Msg 50000, Level 16, State 1, Procedure PreventPhysicianDeletion, Line 8
  - Cannot delete physicians with associated appointments
  - Msg 547, Level 16, State 0, Procedure PreventPhysicianDeletion, Line 18
  - The DELETE statement conflicted with the REFERENCE constraint "FK\_Appointment\_PhysicianID". The conflict occurred in database "MedicalCenter", table "dbo.Appointments", column 'PhysicianID'.
  - The statement has been terminated.
  - Total execution time: 00:00:00.053

#### Comment:

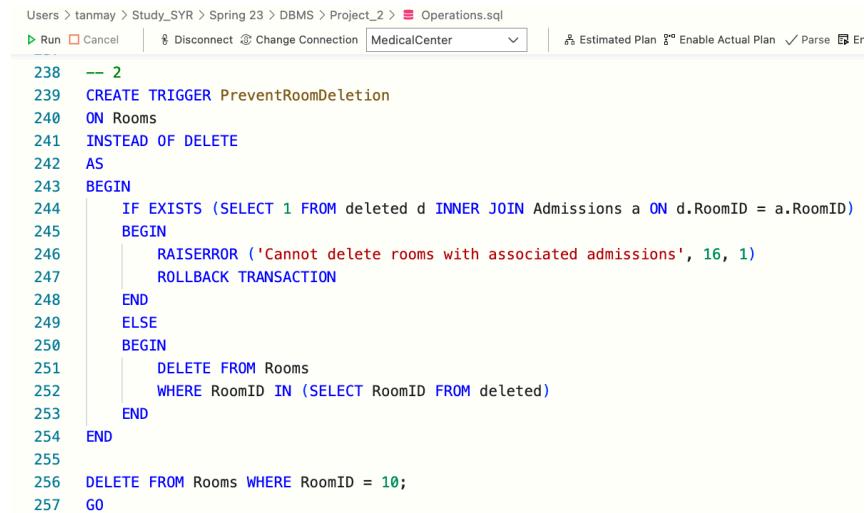
This trigger is used to prevent deletion of records from the Physicians table if they have associated records in the Appointments table.

## 2. PreventRoomDeletion

Code:

```
CREATE TRIGGER PreventRoomDeletion
ON Rooms
INSTEAD OF DELETE
AS
BEGIN
IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Admissions a ON d.RoomID = a.RoomID)
BEGIN
    RAISERROR ('Cannot delete rooms with associated admissions', 16, 1)
    ROLLBACK TRANSACTION
END
ELSE
BEGIN
    DELETE FROM Rooms
    WHERE RoomID IN (SELECT RoomID FROM deleted)
END
DELETE FROM Rooms WHERE RoomID = 10;
```

Execution:



The screenshot shows the SQL Server Management Studio interface with the following details:

- Path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar buttons: Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan, Parse, Er.
- Script pane content:

```
238 -- 2
239 CREATE TRIGGER PreventRoomDeletion
240 ON Rooms
241 INSTEAD OF DELETE
242 AS
243 BEGIN
244     IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Admissions a ON d.RoomID = a.RoomID)
245     BEGIN
246         RAISERROR ('Cannot delete rooms with associated admissions', 16, 1)
247         ROLLBACK TRANSACTION
248     END
249     ELSE
250     BEGIN
251         DELETE FROM Rooms
252         WHERE RoomID IN (SELECT RoomID FROM deleted)
253     END
254 END
255
256 DELETE FROM Rooms WHERE RoomID = 10;
257 GO
```
- Messages pane content:

3:41:43 PM Started executing query at Line 256  
Msg 50000, Level 16, State 1, Procedure PreventRoomDeletion, Line 8  
Cannot delete rooms with associated admissions  
Msg 3609, Level 16, State 1, Line 1  
The transaction ended in the trigger. The batch has been aborted.  
Total execution time: 00:00:00.032

Comment:

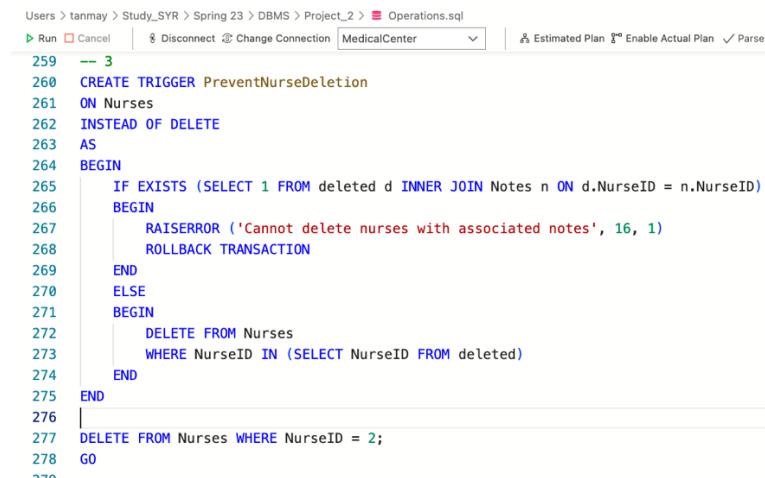
This trigger is used to prevent deletion of records from the Rooms table if they have associated records in the Admissions table.

### 3. PreventNurseDeletion

**Code:**

```
CREATE TRIGGER PreventNurseDeletion
ON Nurses
INSTEAD OF DELETE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Notes n ON d.NurseID = n.NurseID)
    BEGIN
        RAISERROR ('Cannot delete nurses with associated notes', 16, 1)
        ROLLBACK TRANSACTION
    END
    ELSE
    BEGIN
        DELETE FROM Nurses
        WHERE NurseID IN (SELECT NurseID FROM deleted)
    END
END
DELETE FROM Nurses WHERE NurseID = 2;
```

**Execution:**



The screenshot shows the SQL Server Management Studio interface with the following details:

- File path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql
- Toolbar buttons: Run, Cancel, Disconnect, Change Connection, MedicalCenter, Estimated Plan, Enable Actual Plan, Parse.
- Text area content:

```
259 -- 3
260 CREATE TRIGGER PreventNurseDeletion
261 ON Nurses
262 INSTEAD OF DELETE
263 AS
264 BEGIN
265     IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Notes n ON d.NurseID = n.NurseID)
266     BEGIN
267         RAISERROR ('Cannot delete nurses with associated notes', 16, 1)
268         ROLLBACK TRANSACTION
269     END
270     ELSE
271     BEGIN
272         DELETE FROM Nurses
273         WHERE NurseID IN (SELECT NurseID FROM deleted)
274     END
275 END
276
277 DELETE FROM Nurses WHERE NurseID = 2;
278 GO
279
```
- Messages tab:

Started executing query at Line 277  
Msg 50000, Level 16, State 1, Procedure PreventNurseDeletion, Line 8  
Cannot delete nurses with associated notes  
Msg 3609, Level 16, State 1, Line 1  
The transaction ended in the trigger. The batch has been aborted.  
Total execution time: 00:00:00.020

**Comment:**

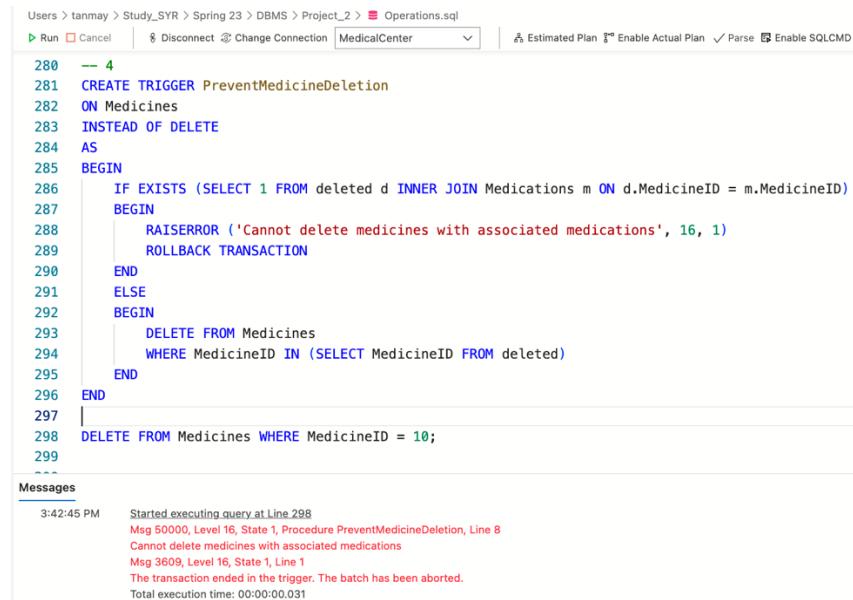
This trigger is used to prevent deletion of records from the Nurses table if they have associated records in the Notes table.

#### 4. PreventMedicineDeletion

##### Code:

```
CREATE TRIGGER PreventMedicineDeletion
ON Medicines
INSTEAD OF DELETE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Medications m ON d.MedicineID = m.MedicineID)
    BEGIN
        RAISERROR ('Cannot delete medicines with associated medications', 16, 1)
        ROLLBACK TRANSACTION
    END
    ELSE
    BEGIN
        DELETE FROM Medicines
        WHERE MedicineID IN (SELECT MedicineID FROM deleted)
    END
END
DELETE FROM Medicines WHERE MedicineID = 10;
```

##### Execution:



The screenshot shows the SQL Server Management Studio interface with the following details:

- Toolbar: Run, Cancel, Disconnect, Change Connection, Estimated Plan, Enable Actual Plan, Parse, Enable SQLCMD.
- Text Editor Content:

```
280 -- 4
281 CREATE TRIGGER PreventMedicineDeletion
282 ON Medicines
283 INSTEAD OF DELETE
284 AS
285 BEGIN
286     IF EXISTS (SELECT 1 FROM deleted d INNER JOIN Medications m ON d.MedicineID = m.MedicineID)
287     BEGIN
288         RAISERROR ('Cannot delete medicines with associated medications', 16, 1)
289         ROLLBACK TRANSACTION
290     END
291     ELSE
292     BEGIN
293         DELETE FROM Medicines
294         WHERE MedicineID IN (SELECT MedicineID FROM deleted)
295     END
296 END
297
298 DELETE FROM Medicines WHERE MedicineID = 10;
299 --
```
- Messages Log:

```
3:42:45 PM Started executing query at Line 298
Msg 50000, Level 16, State 1, Procedure PreventMedicineDeletion, Line 8
Cannot delete medicines with associated medications
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
Total execution time: 00:00:00.031
```

##### Comment:

This trigger is used to prevent deletion of records from the Medicines table if they have associated records in the Medications table.

## 3.5 Transactions

**Transaction 1:**

**Code:**

```
BEGIN TRANSACTION
```

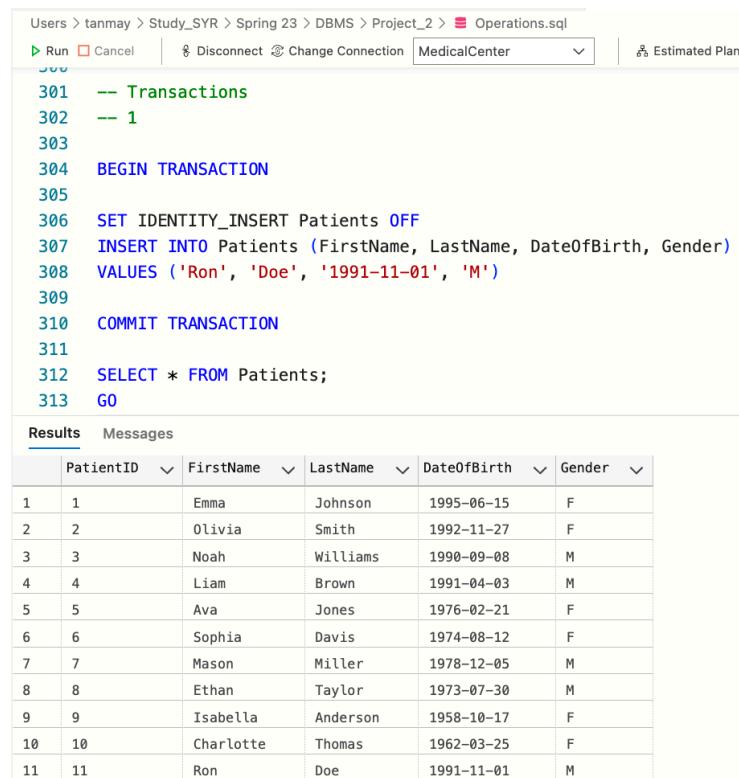
```
SET IDENTITY_INSERT Patients OFF
INSERT INTO Patients (FirstName, LastName, DateOfBirth, Gender)
VALUES ('Ron', 'Doe', '1991-11-01', 'M')
```

```
COMMIT TRANSACTION
```

```
SELECT * FROM Patients;
```

```
GO
```

**Execution:**



The screenshot shows a SQL query execution interface. At the top, there are buttons for 'Run' and 'Cancel', and connection information for 'MedicalCenter'. Below the buttons, the SQL code is displayed with line numbers from 301 to 313. The code includes a transaction block, an insert statement with a specific value ('Ron', 'Doe', '1991-11-01', 'M'), and a select statement. The 'Results' tab is selected at the bottom, showing a table with 11 rows of patient data.

	PatientID	FirstName	LastName	DateOfBirth	Gender
1	1	Emma	Johnson	1995-06-15	F
2	2	Olivia	Smith	1992-11-27	F
3	3	Noah	Williams	1990-09-08	M
4	4	Liam	Brown	1991-04-03	M
5	5	Ava	Jones	1976-02-21	F
6	6	Sophia	Davis	1974-08-12	F
7	7	Mason	Miller	1978-12-05	M
8	8	Ethan	Taylor	1973-07-30	M
9	9	Isabella	Anderson	1958-10-17	F
10	10	Charlotte	Thomas	1962-03-25	F
11	11	Ron	Doe	1991-11-01	M

**Comment:**

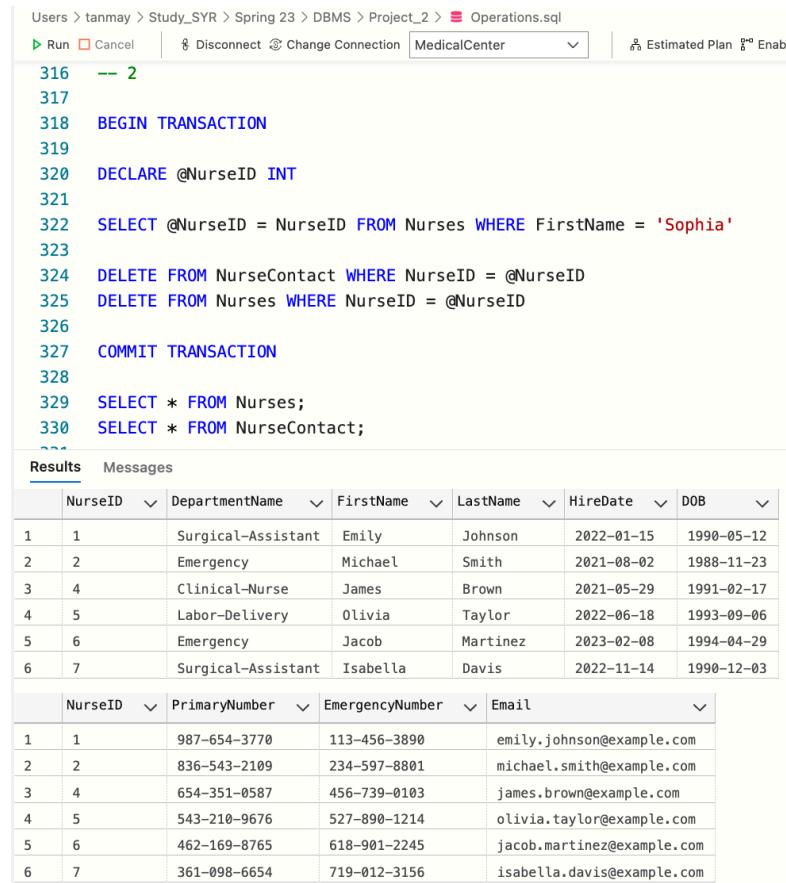
Transaction to insert value in Patients table.

## Transaction 2:

### Code:

```
BEGIN TRANSACTION  
DECLARE @NurseID INT  
SELECT @NurseID = NurseID FROM Nurses WHERE FirstName = 'Sophia'  
DELETE FROM NurseContact WHERE NurseID = @NurseID  
DELETE FROM Nurses WHERE NurseID = @NurseID  
COMMIT TRANSACTION  
  
SELECT * FROM Nurses;  
SELECT * FROM NurseContact;
```

### Execution:



The screenshot shows the execution of a transaction in SQL Server Management Studio. The code is as follows:

```
316 -- 2  
317  
318 BEGIN TRANSACTION  
319  
320 DECLARE @NurseID INT  
321  
322 SELECT @NurseID = NurseID FROM Nurses WHERE FirstName = 'Sophia'  
323  
324 DELETE FROM NurseContact WHERE NurseID = @NurseID  
325 DELETE FROM Nurses WHERE NurseID = @NurseID  
326  
327 COMMIT TRANSACTION  
328  
329 SELECT * FROM Nurses;  
330 SELECT * FROM NurseContact;
```

The results show the state of the Nurses and NurseContact tables after the transaction is committed.

	NurseID	DepartmentName	FirstName	LastName	HireDate	DOB
1	1	Surgical-Assistant	Emily	Johnson	2022-01-15	1990-05-12
2	2	Emergency	Michael	Smith	2021-08-02	1988-11-23
3	4	Clinical-Nurse	James	Brown	2021-05-29	1991-02-17
4	5	Labor-Delivery	Olivia	Taylor	2022-06-18	1993-09-06
5	6	Emergency	Jacob	Martinez	2023-02-08	1994-04-29
6	7	Surgical-Assistant	Isabella	Davis	2022-11-14	1990-12-03

	NurseID	PrimaryNumber	EmergencyNumber	Email
1	1	987-654-3770	113-456-3890	emily.johnson@example.com
2	2	836-543-2109	234-597-8801	michael.smith@example.com
3	4	654-351-0587	456-739-0103	james.brown@example.com
4	5	543-210-9676	527-890-1214	olivia.taylor@example.com
5	6	462-169-8765	618-901-2245	jacob.martinez@example.com
6	7	361-098-6654	719-012-3156	isabella.davis@example.com

### Comment:

Here the nurse details of 'Sophia' are deleted from Nurses and NurseContact table.

### Transaction 3:

#### Code:

```
BEGIN TRANSACTION  
DECLARE @PhysicianID INT  
SELECT @PhysicianID = PhysicianID FROM Physicians WHERE FirstName = 'John' AND LastName = 'Smith'  
UPDATE PhysicianContact  
SET PrimaryNumber = '315-287-9900', EmergencyNumber = '912-456-0087', Email =  
'john.smith.new@example.com'  
WHERE PhysicianID = @PhysicianID  
COMMIT TRANSACTION
```

```
SELECT * FROM PhysicianContact;
```

#### Execution:

The screenshot shows a SQL query window in SSMS. The title bar indicates the path: Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql. The toolbar includes Run, Cancel, Disconnect, Change Connection (set to MedicalCenter), Estimated Plan, Enable Actual Plan, Parse, Enable SQLCMD, and Export as Note. The code area contains numbered lines 333 through 343, which are the transaction script. The results tab is selected, showing a table with columns: PhysicianID, PrimaryNumber, EmergencyNumber, and Email. The data for six rows is as follows:

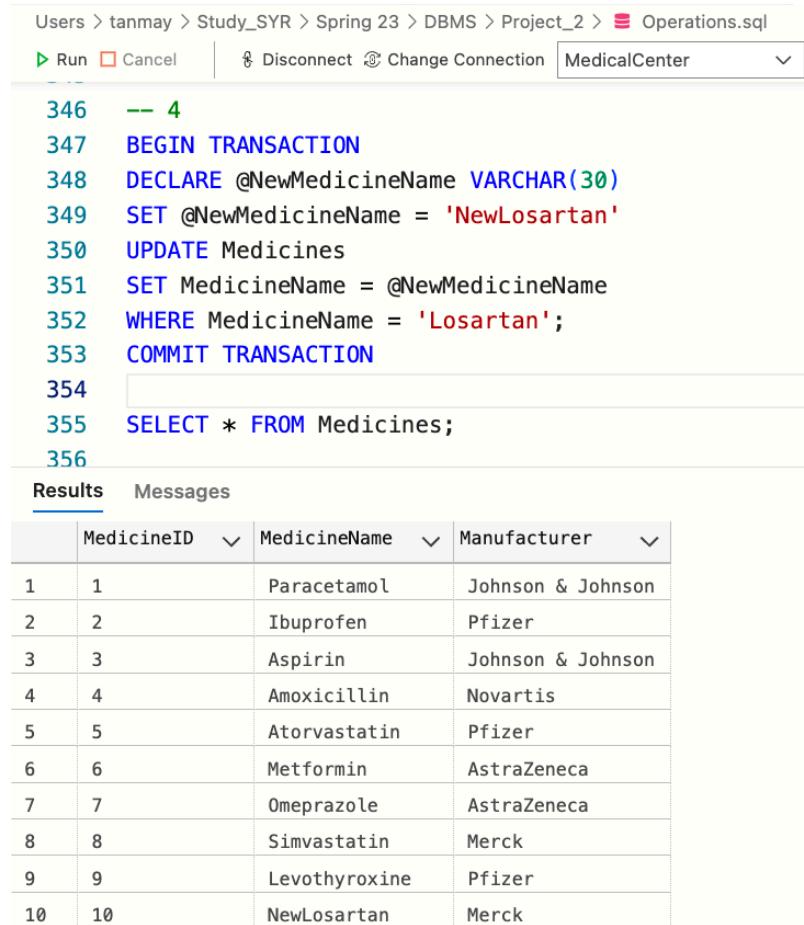
	PhysicianID	PrimaryNumber	EmergencyNumber	Email
1	1	315-287-9900	912-456-0087	john.smith.new@example.com
2	2	233-187-0123	345-654-3210	emily.johnson@example.com
3	3	564-937-6541	247-234-5678	michael.brown@example.com
4	4	918-087-0222	936-456-7899	sophia.anderson.new@example.com
5	5	123-644-3210	432-886-6541	david.williams@example.com
6	6	907-664-5688	180-487-6543	olivia.taylor@example.com

#### Comment:

Here we can see that the values in PhysicianContact tables have been updated.

**Transaction 4:****Code:**

```
BEGIN TRANSACTION  
DECLARE @NewMedicineName VARCHAR(30)  
SET @NewMedicineName = 'NewLosartan'  
UPDATE Medicines  
SET MedicineName = @NewMedicineName  
WHERE MedicineName = 'Losartan';  
COMMIT TRANSACTION  
  
SELECT * FROM Medicines;
```

**Execution:**

The screenshot shows a SQL query execution interface. At the top, there's a navigation bar with 'Operations.sql' selected. Below it, a toolbar has 'Run' and 'Cancel' buttons. The main area contains the SQL code for Transaction 4, with line numbers 346 through 356. Line 356 is a comment '-- 4'. The code includes a BEGIN TRANSACTION block, variable declarations, an UPDATE statement, and a COMMIT TRANSACTION. Below the code, there are two tabs: 'Results' and 'Messages'. The 'Results' tab is selected and displays a table with 10 rows of medicine data. The columns are MedicineID, MedicineName, and Manufacturer. The data shows 'NewLosartan' as the 10th entry.

	MedicineID	MedicineName	Manufacturer
1	1	Paracetamol	Johnson & Johnson
2	2	Ibuprofen	Pfizer
3	3	Aspirin	Johnson & Johnson
4	4	Amoxicillin	Novartis
5	5	Atorvastatin	Pfizer
6	6	Metformin	AstraZeneca
7	7	Omeprazole	AstraZeneca
8	8	Simvastatin	Merck
9	9	Levothyroxine	Pfizer
10	10	NewLosartan	Merck

**Comment:**

Here we can see that the medicine name has been updated.

## 3.6 Scripts

### Script 1:

#### Code:

```
CREATE LOGIN ReadOnlyUser WITH PASSWORD = 'ReadOnlyPassword450';
```

```
CREATE USER ReadOnlyUser FOR LOGIN ReadOnlyUser;
```

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

#### Execution:

The screenshot shows the SQL Server Management Studio interface. On the left, there's a script editor window with the following content:

```
358 -- Scripts
359 -- 1
360 CREATE LOGIN ReadOnlyUser WITH PASSWORD = 'ReadOnlyPassword450';
361 CREATE USER ReadOnlyUser FOR LOGIN ReadOnlyUser;
362
363 EXEC sp_addrolemember 'db_datareader', 'ReadOnlyUser';
```

Below the script editor is a 'Messages' pane showing the execution results:

```
4:27:30 PM Started executing query at Line 363
Commands completed successfully.
Total execution time: 00:00:00.006
```

To the right of the script editor is a 'User - ReadOnlyUser (Preview)' dialog box. This dialog contains several tabs: General, General (selected), Owned Schemas, Membership, and Help.

- General Tab:** Shows the user details:
  - Name: ReadOnlyUser
  - Default schema: dbo
  - Type: User with login
  - Login: ReadOnlyUser
- Owned Schemas:** A list of schemas with checkboxes:
  - db\_accessadmin
  - db\_backupoperator
  - db\_datareader
  - db\_databwriter
  - db\_ddladmin
  - db\_denydatareader
  - db\_denydatawriter
  - db\_owner
  - db\_securityadmin
  - dbo
  - guest
  - INFORMATION\_SCHEMA
  - sys
- Membership:** A list of roles with checkboxes:
  - db\_accessadmin
  - db\_backupoperator
  - db\_datareader
  - db\_databwriter
  - db\_ddladmin
  - db\_denydatareader
  - db\_denydatawriter
  - db\_owner
  - db\_securityadmin

At the bottom of the dialog are OK and Cancel buttons.

#### Comment:

Here we can see that the new user is created who has data-reader membership.

## Script 2:

### Code:

```
CREATE LOGIN ReadWriteUser WITH PASSWORD = 'ReadWritePassword972';
CREATE USER ReadWriteUser FOR LOGIN ReadWriteUser;
```

```
EXEC sp_addrolemember 'db_datawriter', 'ReadWriteUser';
```

### Execution:

The screenshot shows the SQL Server Management Studio interface. On the left, there are three tabs: 'Create\_Table.sql - localh...er (sa)', 'Insert\_Data.sql - localh...er (sa)', and 'Operations.sql - localh...er (sa)'. The 'Operations.sql' tab is active, containing the following T-SQL script:

```
366 -- 2
367 CREATE LOGIN ReadWriteUser WITH PASSWORD = 'ReadWritePassword972';
368 CREATE USER ReadWriteUser FOR LOGIN ReadWriteUser;
369
370 EXEC sp_addrolemember 'db_datawriter', 'ReadWriteUser';
371
```

In the center, a preview window titled 'User - ReadWriteUser (Preview)' displays the user configuration. The 'General' section shows:

- Name: ReadWriteUser
- Default schema: dbo
- Type: User with login
- Login: ReadWriteUser

The 'Owned Schemas' section lists system schemas with checkboxes, none of which are selected:

Selected	Name
<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_dareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin
<input type="checkbox"/>	dbo
<input type="checkbox"/>	guest
<input type="checkbox"/>	INFORMATION_SCHEMA
<input type="checkbox"/>	sys

The 'Membership' section shows the user's current roles. The 'db\_datawriter' role is checked, while others are unchecked:

Selected	Name
<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_dareader
<input checked="" type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin

At the bottom of the preview window, there are 'Help', 'OK', and 'Cancel' buttons. The status bar at the bottom of the screen shows 'Ln 372, Col 1 Spaces: 4 UTF-8 LF (6 SQL)'.

### Comment:

Here we can see that the new user is created who has data-writer membership.

### Script 3:

#### Code:

```
CREATE LOGIN AdminUser WITH PASSWORD = 'AdminPassword123';
```

```
CREATE USER AdminUser FOR LOGIN AdminUser;
```

```
EXEC sp_addrolemember 'db_owner', 'AdminUser';
```

#### Execution:

The screenshot shows the SSMS interface with three tabs open: 'Create\_Table.sql', 'Insert\_Data.sql', and 'Operations.sql'. The 'Operations.sql' tab contains the following T-SQL script:

```
373 -- 3
374 CREATE LOGIN AdminUser WITH PASSWORD = 'AdminPassword123';
375 CREATE USER AdminUser FOR LOGIN AdminUser;
376
377 EXEC sp_addrolemember 'db_owner', 'AdminUser';
378
```

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

```
4:26:19 PM Started executing query at Line 377.
Commands completed successfully.
Total execution time: 00:00:00.010
```

On the right, the 'User - AdminUser (Preview)' dialog is displayed with the following settings:

General	
Name	AdminUser
Default schema	dbo
Type	User with login
Login	AdminUser
Owned Schemas	
Selected	Name
<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_datareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin
<input type="checkbox"/>	dbo
<input type="checkbox"/>	guest
<input type="checkbox"/>	INFORMATION_SCHEMA
<input type="checkbox"/>	sys
Membership	
Selected	Name
<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_datareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input checked="" type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin

At the bottom of the dialog, there are 'Help', 'OK', and 'Cancel' buttons.

#### Comment:

Here we can see that the new user is created who has owner membership.

#### Script 4:

##### Code:

```
CREATE LOGIN AccessAdminUser WITH PASSWORD = 'AccessAdminPassword369';
CREATE USER AccessAdminUser FOR LOGIN AccessAdminUser;
```

```
EXEC sp_addrolemember 'db_accessadmin', 'AccessAdminUser';
```

##### Execution:

The screenshot shows the SSMS interface with three tabs open: Create\_Table.sql, Insert\_Data.sql, and Operations.sql. The Operations.sql tab contains the following T-SQL script:

```
380    -- 4
381 CREATE LOGIN AccessAdminUser WITH PASSWORD = 'AccessAdminPassword369';
382 CREATE USER AccessAdminUser FOR LOGIN AccessAdminUser;
383 EXEC sp_addrolemember 'db_accessadmin', 'AccessAdminUser';
385
```

The Messages pane shows the execution results:

```
4:26:48 PM Started executing query at Line 384
Commands completed successfully.
Total execution time: 00:00:00.009
```

A context menu is open over the 'AccessAdminUser' entry in the Object Explorer, displaying a preview window titled 'User - AccessAdminUser (Preview)'. The preview window contains the following details:

General	
Name	AccessAdminUser
Default schema	dbo
Type	User with login
Login	AccessAdminUser

Below the General section are two expandable sections: 'Owned Schemas' and 'Membership'.

Selected	Name
<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_datareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin
<input type="checkbox"/>	dbo
<input type="checkbox"/>	guest
<input type="checkbox"/>	INFORMATION_SCHEMA
<input type="checkbox"/>	sys

Selected	Name
<input checked="" type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_datareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin

At the bottom of the preview window are 'Help', 'OK', and 'Cancel' buttons. The status bar at the bottom of the screen shows 'Ln 383, Col 1 Spaces: 4 UTF-8 LF (SQL)'.

##### Comment:

Here we can see that the new user is created who has access-admin membership.

### **3.7 Business Report**

#### **1. Number of times a medicine was prescribed.**

**Code:**

```
SELECT m.MedicineName, COUNT(*) AS NumberOfTimesPrescribed
FROM Medications AS md
JOIN Medicines AS m ON md.MedicineID = m.MedicineID
GROUP BY m.MedicineName
ORDER BY NumberOfTimesPrescribed DESC;
```

**Data:**

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql

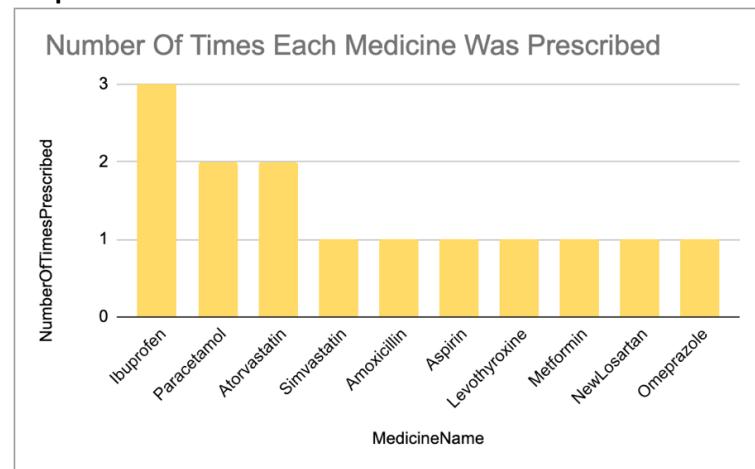
Run Cancel Disconnect Change Connection MedicalCenter Estimat

```
388 -- Visualization
389 -- Count of medicines prescribed
390 SELECT m.MedicineName, COUNT(*) AS NumberOfTimesPrescribed
391 FROM Medications AS md
392 JOIN Medicines AS m ON md.MedicineID = m.MedicineID
393 GROUP BY m.MedicineName
394 ORDER BY NumberOfTimesPrescribed DESC;
```

Results Messages

	MedicineName	NumberOfTimesPrescribed
1	Ibuprofen	3
2	Paracetamol	2
3	Atorvastatin	2
4	Simvastatin	1
5	Amoxicillin	1
6	Aspirin	1
7	Levothyroxine	1
8	Metformin	1
9	NewLosartan	1
10	Omeprazole	1

**Graph:**



**Comment:**

This bar graph shows the number of times each medicine in the database was prescribed to the patient. This shows us which medicine was prescribed more and helps us analyse what type of diseases do frequently occur.

## 2. Contribution of medicine manufacturers

**Code:**

```
SELECT m.Manufacturer, COUNT(*) AS Contributon  
FROM Medications AS md  
JOIN Medicines AS m ON md.MedicineID = m.MedicineID  
GROUP BY m.Manufacturer  
ORDER BY Contributon DESC;
```

**Data:**

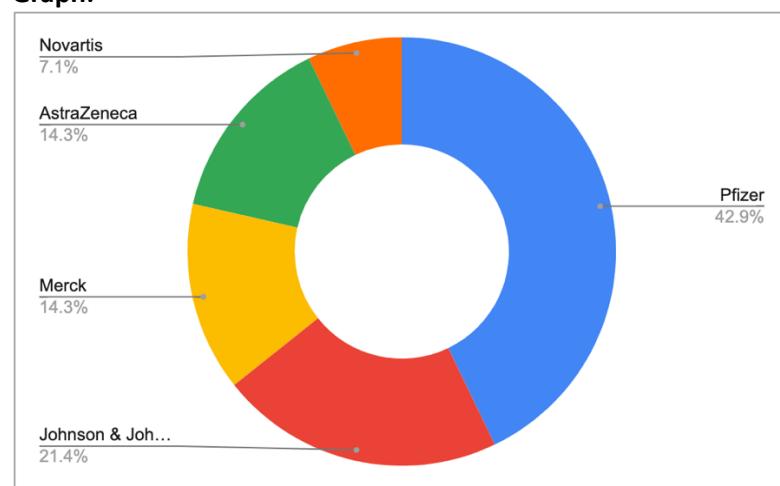
Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql

```
397 -- Contribution of manufacturers in medicine  
398 SELECT m.Manufacturer, COUNT(*) AS Contributon  
399 FROM Medications AS md  
400 JOIN Medicines AS m ON md.MedicineID = m.MedicineID  
401 GROUP BY m.Manufacturer  
402 ORDER BY Contributon DESC;  
403  
101
```

Results Messages

	Manufacturer	Contributon
1	Pfizer	6
2	Johnson & Johnson	3
3	Merck	2
4	AstraZeneca	2
5	Novartis	1

**Graph:**



**Comment:**

This donut graph shows the percentage of medicine manufacturers who have their medicines in the database.

### 3. Money covered by insurance companies

Code:

```
SELECT i.Company, SUM(b.InsuranceCoverage) AS TotalCoverage
FROM Billings AS b
JOIN Insurances as i ON i.InsuranceID = b.InsuranceID
GROUP BY i.Company
ORDER BY SUM(b.InsuranceCoverage) DESC;
```

Data:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql

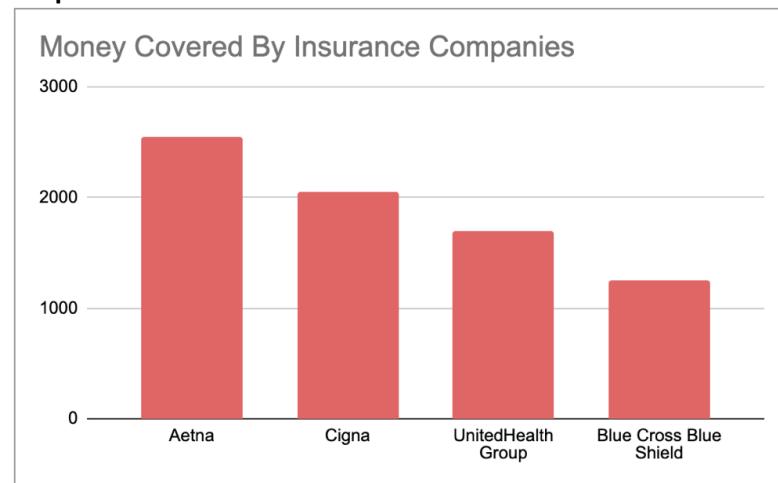
Run Cancel Disconnect Change Connection MedicalCenter Estimate

405 -- Money covered by insurance companies
406 SELECT i.Company, SUM(b.InsuranceCoverage) AS TotalCoverage
407 FROM Billings AS b
408 JOIN Insurances as i ON i.InsuranceID = b.InsuranceID
409 GROUP BY i.Company
410 ORDER BY SUM(b.InsuranceCoverage) DESC;
411
412

Results Messages

	Company	TotalCoverage
1	Aetna	2550.00
2	Cigna	2050.00
3	UnitedHealth Group	1700.00
4	Blue Cross Blue Shield	1250.00

Graph:



Comment:

This bar graph shows the amount of money paid by each insurance company in the database.

#### 4. Patient details

##### Code:

```
SELECT Gender, COUNT(Gender) AS GenderCount, AVG(DATEDIFF(YEAR, DateOfBirth,  
GETDATE())) AS AverageAge  
FROM Patients  
GROUP BY Gender;
```

##### Data:

Users > tanmay > Study\_SYR > Spring 23 > DBMS > Project\_2 > Operations.sql

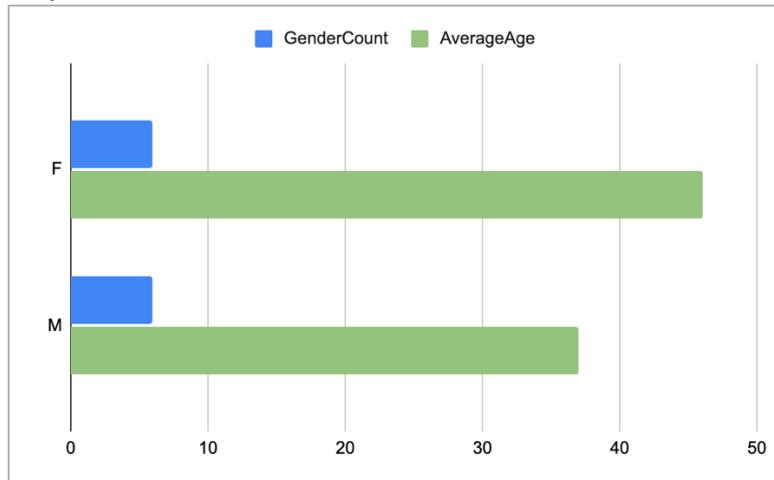
Run Cancel Disconnect Change Connection MedicalCenter Estimated Plan Enable Actual Plan Parse Enable SQLCMD Export as

```
422 --- Count of Gender
423 SELECT Gender, COUNT(Gender) AS GenderCount, AVG(DATEDIFF(YEAR, DateOfBirth, GETDATE())) AS AverageAge
424 FROM Patients
425 GROUP BY Gender;
426
427
```

Results Messages

	Gender	GenderCount	AverageAge
1	F	6	46
2	M	5	38

##### Graph:



##### Comment:

This bar graph shows the count of male and female patients along with the average age of each gender.

## **4. Conclusion**

The implemented database design provides a comprehensive and organized system for managing medical center records. It allows for the efficient storage, retrieval, and management of patient information, addresses, medical records, doctor details, and appointment records. By leveraging the relationships between the tables, medical center staff can easily access relevant information, streamline administrative tasks, and provide enhanced patient care. The database design promotes accuracy, reliability, and accessibility of data, facilitating effective decision-making and improving the overall operational efficiency of the medical center.

### **4.1 Remarks**

From this project I learnt many things such as designing a database and learnt to write complex views, functions, triggers, stored procedures, and scripts. I also learnt to make business reports.