DESIGN AND DEVELOPMENT OF HOSPITAL MANAGEMENT SYSTEM

By

Syed Ali Murtaza

1. INTRODUCTION

1. Introduction to the Assignment

In this assignment we are going to implement the design and development of Hospital management system. In this senerio we are tasked to provide a robust and efficient database system that caters the need of the hospital.This task is done using the triggers,Views, Stored Procedures, System functions and user define function and the select statement with the use of joins and sub queries as per the instruction of the instructor. This system also ensure database design , normalization to ensure data integrity and accessibility. Now below are the steps that I used to implement the database design and the logic behind them and technical implementations using T-SQL statements.

2. STEPS TO IMPLEMENT THE PROJECT

2.1 Understanding The Requirements:-

The requirements for designing the database is to store the information based on Patients, doctors, medical records which include (past appointments,diagnosis, medicine, medicine prescribed date, allergies), Appointments and departments.

2.2 Understanding The Clients Requirements:-

1. When the patient comes he should register with GB using his full name, insurance, date of birth, address etc.
2. They must create their username and password to allow them to sign in into the patients portal.
3. The system will store the information of patients.
4. The Patient will book an appointment through patients portal.
5. System will check for the doctor availibility and if possible than book an appointment.

6. The system will store the appointment details.
7. When patient arrive on an appointment than doctor can review his(patient) past appointments and details.
8. After appointment the doctor can update the medical record and the patient can leave a feedback about the doctor.
9. If the patient cancel the appointment than he can rebook his appointment
10. When the patient attend the appointment than his status must be change from pending to completed.
11. If the patient leaves the hospital, than hospital wants to retain his data but they should record the date when patient left.

3. DATABASE DESIGN AND NORMALIZATION

After understanding the requirement, the first step is to conceptualize the entities involved and their relationship between them. I identify the following tables which include Patients, PatientContactInfo, UserAccount, Doctors, DoctorsReviews, DoctorSpecialization, Specialization, MedicalHistory, Appointment, AppointmentStatus, DeletedAppointment and Department.

3.1 Patient Table:-

I design the Patient table as follow
PatientID INT:- I marked it as INTEGER and used it as a primary key for each patient to ensure uniqueness.

FullName NVARCHAR(20) not null:- this is the full name of the patient and it is marked as a nvarchar(allowing for Unicode character) max length 20 and it must be not null.

Address nvarchar(100) not null:- this is the full address of the patient and it is marked as a nvarchar(allowing for Unicode character) max length 100 and it must be not null.

DateOfBith date not null:- it is simply the date of birth of the patient and it must be not null.

insurance nvarchar(100) not null:- this is the insurance of the patient and it is marked as a nvarchar(allowing for Unicode character) max length 100 and it must be not null

DepartureDate date null:- this is the **DepartureDate** of the patient from hospital and it is null so that it would not disturb the records of patients who has not left the hospital.

```
Create table Patient(
PatientID int primary key,
FullName nvarchar(50) not null,
address nvarchar(250) not null,
DateOfBirth date not null,
insurance nvarchar(250) not null,
);
ALTER TABLE Patient
ADD DepartureDate DATE NULL;
```

The sample data for a above patients table is

```
475 INSERT INTO Patient (PatientID, FullName, Address, DateOfBirth, Insurance, DepartureDate, Age)
476 VALUES
477
478   (8, 'ALI MURTAZA', '901 Maple St, Ruralville', '1994-04-22', '567 Insurance', NULL, NULL);
479   (1, 'John Doe', '123 Main St, Cityville', '1980-05-15', 'ABC Insurance', NULL, NULL),
480   (2, 'Jane Smith', '456 Elm St, Townsville', '1975-10-20', 'XYZ Insurance', NULL, NULL),
481   (3, 'Alice Johnson', '789 Oak St, Villagetown', '1990-03-08', '123 Insurance', NULL, NULL),
482   (4, 'Michael Brown', '567 Pine St, Hamletville', '1988-07-03', '456 Insurance', NULL, NULL),
483   (5, 'Emily Williams', '890 Cedar St, Countryside', '1972-12-30', '789 Insurance', NULL, NULL),
484   (6, 'David Jones', '234 Birch St, Suburbia', '1995-09-18', '234 Insurance', NULL, NULL),
485   (7, 'Sarah Garcia', '901 Maple St, Ruralville', '1984-04-22', '567 Insurance', NULL, NULL);
486
```

Now after creating the patient table. I have also added a stored procedure with the name of PatientDepartureDate and its function is to store the departure date when the Patient left the hospital.

```
57 Create procedure PatientDepartureDate
58 @PatientID int,
59 @DepartureDate date
60 as
61 begin
62     update Patients
63     Set DepartureDate=@DepartureDate
64     where PatientID=@PatientID
65     end;
66
```

3.2 UserAccounts Table

UserID:- it is the primary key of the table.

PatientID:- it is the foreign key references the PatientID in the Patient table.

username nvarchar(200) unique not null:- this is the username of the patient and it is marked as a nvarchar(allowing for Unicode character) max length 50 and it must unique and not null.

passwordHash binary(64) not null:- this is the username of the patient and it is marked as a binary max length 64 and it is not null.

Salt not null: this is used for security measures and its functionality is defined in next stored procedure when we add a new patient that it will act as a security measure to ensure the strength of password.

```
39
40  -- Table creation
41  CREATE TABLE UserAccounts (
42      UserID INT PRIMARY KEY,
43      PatientID INT FOREIGN KEY REFERENCES Patient, -- Assuming there's a table named 'Patient'
44      username NVARCHAR(200) UNIQUE NOT NULL,
45      passwordHash binary(64) NOT NULL,
46      salt NVARCHAR(50) NOT NULL
47  );
48
```

And the sample data for above is

```
491  -- Inserting data for 7 users
492  INSERT INTO UserAccounts (UserID, PatientID, username, passwordHash, salt)
493  VALUES
494      (1, 1, 'john_doe', 0x6C7280B3C4E97A, @SaltPrefix + '1'),
495      (2, 2, 'jane_smith', 0x8F32A1B2C3D4E5, @SaltPrefix + '2'),
496      (3, 3, 'alice_johnson', 0xABDE87C4E3F6A9, @SaltPrefix + '3'),
497      (4, 4, 'bob_jackson', 0x923F6A2B4C1D7E, @SaltPrefix + '4'),
498      (5, 5, 'sarah_miller', 0x5C2E7F1A3B9D8C, @SaltPrefix + '5'),
499      (6, 6, 'michael_brown', 0x7A8F3B9D5C2E4F, @SaltPrefix + '6'),
500      (7, 7, 'emily_taylor', 0xD8C7E9A2B6F3D5, @SaltPrefix + '7');
501
```

Now after creating the user login table now I have create a stored procedure with the name AddPatientWithCredentials to ensure security for each user. In this stored procedure First I have pass several parameters like UserID, PatientID, Username, Password. And then I declare 2 variable salt and hashpassword respectively and Hashpassword variable uses HASHBYTES function with the algorithm SHA2_512 to generate a random salt and than hashpassword variable will concatenate the password and salt in this way it will create a secure hash password that cannot be easily reversed to its original password.

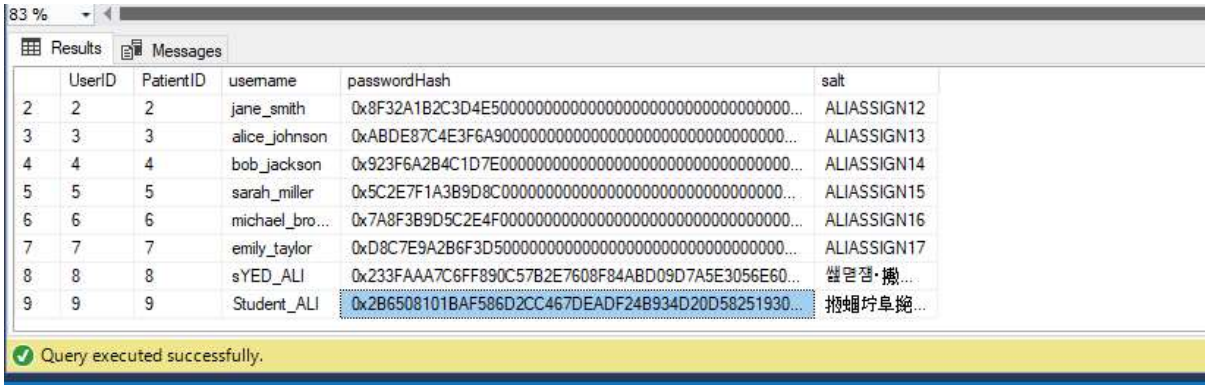
```
-- Stored procedure to add patients with hashed password and salt
CREATE PROCEDURE AddPatientWithCredentials
    @UserID INT,
    @PatientID INT,
    @Username NVARCHAR(200),
    @Password NVARCHAR(200)
AS
BEGIN
    DECLARE @Salt NVARCHAR(50);
    DECLARE @HashedPassword binary(200);

    -- Generate salt
    SET @Salt = HASHBYTES('SHA2_512', CONVERT(VARCHAR(36), NEWID()));

    -- Hash password with salt
    SET @HashedPassword = HASHBYTES('SHA2_512', @Password + @Salt);

    -- Add the user with hashed password and salt
    INSERT INTO UserAccounts (UserID, PatientID, username, passwordHash, salt)
    VALUES (@UserID, @PatientID, @Username, @HashedPassword, @Salt);
END;
```

Now lets check its execution first of all for creating the username and password first of all system will check that the table have already filled the patient table if Yes than he will be able to create user name and password. Now lets declare the paraments and execute the store procedure

```
    -- Execute the stored procedure to add a patient with credentials
DECLARE @UserID INT = 9; -- Assuming UserID
DECLARE @PatientID INT = 9; -- Assuming PatientID
DECLARE @Username NVARCHAR(200) = 'Student_ALI'; -- Assuming Username
DECLARE @Password NVARCHAR(200) = 'MANIBheheA3'; -- Assuming Password

EXEC AddPatientWithCredentials @UserID, @PatientID, @Username, @Password;
```

Now from this SS you can see that the password is Nvarchar but when we execute the UserLogin table

```


SELECT * FROM UserAccounts;

```



We can see that now the password is in hexadecimal number now it is not easy to crack the original password.

3.3 PatientContactInfo

ContactID:-this column is integer and it is the primary key which is unique to find the patients contact information.

PatientID:- it is also an integer column and is the foreign key referencing to Patient Table.

Email:- it is the email of Patients and it is optional for the patients.

Telephone:- it is marked as big int due to length of telephone number and it is also optional.

```
create table PatientContactInfo(
contactID int primary key,
PatientID int foreign key references Patient,
email nvarchar(200),
Telephone bigint,
);
```

3.4 RegisteredPatient:-

This database design also contains an extra feature that if a hospital wants to retrieve all the registered patients a stored procedure is used which takes entities from userAccount table, Patient table and Patient contact info table and then join them using inner joins. In this way hospital can see number of registered patients.

```
68  CREATE PROCEDURE RegisteredPatients
69  AS
70  BEGIN
71      SELECT
72          p.PatientID,
73          p.FullName,
74          p.address,
75          p.DateOfBirth,
76          p.insurance,
77          p.DepartureDate,
78          u.username,
79          c.email,
80          c.Telephone
81      FROM Patient p
82      inner JOIN UserAccounts u ON p.PatientID = u.PatientID
83      inner JOIN PatientContactInfo c ON p.PatientID = c.PatientID
84      ORDER BY p.PatientID;
85  END;
```

The hospital can see the registered Patients by executing this and in our case

Results | Messages

	PatientID	FullName	address	DateOfBirth	insurance	DepartureDate	username	email	Telephone
1	1	John Doe	123 Main St, Cityville	1980-05-15	ABC Insurance	NULL	john_doe	john_doe@example.com	1234567890
2	2	Jane Smith	456 Elm St, Townsville	1975-10-20	XYZ Insurance	NULL	jane_smith	jane_smith@example.com	9876543210
3	3	Alice Johnson	789 Oak St, Villagetown	1990-03-08	123 Insurance	NULL	alice_johnson	alice_johnson@example.com	1112223333
4	4	Michael Bro...	567 Pine St, Hamletville	1988-07-03	456 Insurance	NULL	bob_jackson	bob_jackson@example.com	4445556666
5	5	Emily Williams	890 Cedar St, Country...	1972-12-30	789 Insurance	NULL	sarah_miller	sarah_miller@example.com	7778889999
6	6	David Jones	234 Birch St, Suburbia	1995-09-18	234 Insurance	NULL	michael_bro...	michael_brown@example.c...	1231231234
7	7	Sarah Garcia	901 Maple St, Ruralville	1984-04-22	567 Insurance	NULL	emily_taylor	emily_taylor@example.com	4564564567

3.5 Doctor Table

DoctorID:- it is the primary key which is unique for each doctor and it is marked as INT.

FullName:- it is the full name of the doctor and it is nvarchar.
DepartmentID:- departmentID is an integer column that references the departmentID in the department table.
Email:-it is the email of doctor marked as nvarchar and it is optional.
Telephone:- it is the telephone number of doctor and it is marked as bigint.
Speciality:- it is the speciality of each doctor and it is compulsory so it is marked as not null.

```
Create table Doctors(
DoctorID int primary key,
FullName nvarchar(200) not null,
departmentID int FOREIGN KEY REFERENCES department(departmentID) NOT NULL,
email nvarchar(100),
telephone bigint,
Specialty nvarchar(100) not null,

);
```

3.6 Doctor's Specialization table

SpecializationID:- it is the primary key for each specialization

Specialization name:- it is the column that stores the name of the specialization and it is nvarchar

```
CREATE TABLE Specializations (
    SpecializationID INT IDENTITY(1,1) PRIMARY KEY,
    SpecializationName NVARCHAR(255) NOT NULL
);
```

3.7 Doctor Specializations

This table is used to handle many to many relationship between doctor and specialization.
In this table I made a composite key referencing both DoctorID and specializationID as primary key and than a DoctorID also referencing the DoctorID in Doctor table and specialization referencing the specialization key in specialization table.In this way each

doctor is associated with many specialization and each specialization is associated with many doctors.

```
CREATE TABLE DoctorSpecializations (
    DoctorID INT,
    SpecializationID INT,
    PRIMARY KEY (DoctorID, SpecializationID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID),
    FOREIGN KEY (SpecializationID) REFERENCES Specializations(SpecializationID)
);
```

3.8 Appointment table:-

AppointmentID:- it is the primary key which uniquely define each department and it is marked as integer column.

PatientID:-it is the foreign key references to PatientID in Patient column and it is marked as int.

DoctorID:- it is the foreign key references to DoctorID in Doctor column and it is marked as int.

AppointmentDate:- it is the appointmentDate colum which stores the date of appointment of doctor and patient and it is marked as date and it cannot be null. After that I also alter this colum and add the check constrain which checks the appointment date must be from future and not from past dates to make the database more efficient.

AppointmentTime:- it is the appointmentTime colum which stores the time of appointment of doctor and patient and it is marked as Time and it cannot be null.

```
CREATE TABLE Appointment (
   AppointmentID INT  PRIMARY KEY,
   PatientID INT FOREIGN KEY (PatientID) REFERENCES Patient(PatientID) not null,
   DoctorID INT FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)  not null,
   AppointmentDate DATE not null,
   AppointmentTime TIME not null,
);
Alter table appointment
add  CONSTRAINT CHECK_AppointmentDate CHECK (AppointmentDate >= CAST(GETDATE() AS DATE));

```

3.9 AppointmentStatus:-

This table is used to determine the status of the appointment whether the status is pending booked, available or cancelled however complete breakdown of logic is as

StatusID:- First I define the statusID which is the primary key and is used to identify each appointment which is 1st NF form

AppointmentID:- than I define the AppointmentID which is foreign key here referencing back to the appointment table to make a link between AppointmentStatus table and Appointment table.

Status:- than I define the Status column with a check that appointment Status cannot be out of these 4 status which are available pending booked and cancelled.

```
CREATE TABLE AppointmentStatus (
   StatusID INT IDENTITY(1,1) PRIMARY KEY,
   AppointmentID INT,
    Status VARCHAR(50) NOT NULL CHECK (Status IN ('Pending', 'Cancelled', 'Available','Completed')),
   FOREIGN KEY (AppointmentID) REFERENCES Appointment(AppointmentID)
);
```

Now as per the Client requirement that when the patient cancels the appointment than their status should be converted into available so that whenever he wants to again book the appointment he would do it easily. So for this purpose I created a trigger on on appointment status table that when the appointment status=Cancelled than it should be converted into available

```
138  CREATE TRIGGER TR_CancelledAppointment
139  ON AppointmentStatus
140  AFTER UPDATE
141  AS
142  BEGIN
143     IF UPDATE(Status)
144     BEGIN
145        DECLARE @CancelledStatus VARCHAR(50) = 'Cancelled';
146        UPDATE a
147        SET a.Status = 'Available'
148        FROM AppointmentStatus AS a
149        INNER JOIN inserted AS i ON a.AppointmentID = i.AppointmentID
150        WHERE i.Status = @CancelledStatus;
151     END
152  END;
153
```

Procedure to book an appointment:-

Here I have created a procedure to book an appointment. Let suppose a Patient comes to book an appointment the system will ask for PatientID ,DoctorID(patient will chose the doctor themselfs), AppointmentDate, Appointment time.

Than the system will check whether the appointment is available or not by looking into the Appointment table. If the appointment is available than the system will store the given appointment data into the Appointment table and also update the Status in the AppointmentStatus table to ‘Booked’. Otherwise gives the error that Doctor not available

```
Object Explorer   databaseassignpart...ary Computer (67))*
187 CREATE PROCEDURE BookAppointment
188   @PatientID INT,
189   @DoctorID INT,
190   @AppointmentDate DATE,
191   @AppointmentID INT,
192   @AppointmentTime TIME
193 AS
194 BEGIN
195   IF NOT EXISTS (
196     SELECT * FROM Appointment
197     WHERE DoctorID = @DoctorID
198     AND AppointmentDate = @AppointmentDate
199     AND AppointmentTime = @AppointmentTime
200   )
201   BEGIN
202     INSERT INTO Appointment (AppointmentID, PatientID, DoctorID, AppointmentDate, AppointmentTime)
203     VALUES (@AppointmentID, @PatientID, @DoctorID, @AppointmentDate, @AppointmentTime);
204
205     -- Assuming you have an AppointmentStatus table and logic implemented for it.
206     INSERT INTO AppointmentStatus (AppointmentID, Status)
207     VALUES (@AppointmentID, 'Booked');
208
209     SELECT 'Appointment booked successfully.' AS Message;
210   END
211   ELSE
212   BEGIN
213     SELECT 'Doctor not available at the selected time.' AS Message;
214   END
215 END;
```

Execution:-

Now we can verify its execution

```
618
619   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 236, @AppointmentDate = '2024-07-25', @AppointmentTime = '11:00:00';
1 %
Results  Messages
   Message
1  Appointment booked successfully.
```

When we re run the same command it will show error

```
618
619   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 236, @AppointmentDate = '2024-07-25', @AppointmentTime = '11:00:00';
Results  Messages
Message
Doctor not available at the selected time.
```

When we rerun the same querry by changing the appointmentID and appointment date it runs successfully

```
618
619   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 896, @AppointmentDate = '2024-08-25', @AppointmentTime = '11:00:00';
%
Results  Messages
Message
Appointment booked successfully.
```

When I rerun the same querry by change the appointmentID and appointment time it runs successfully

1

```
618
619   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 623, @AppointmentDate = '2024-08-25', @AppointmentTime = '09:00:00';
```

Results | Messages

Message
Appointment booked successfully.

When we rerun the same querry by only changing appointmentID than it will also shows error because at the same time and date only 1 appointment occur.

```
618
619   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 624, @AppointmentDate = '2024-08-25', @AppointmentTime = '09:00:00';
```

91 % | Results | Messages

	Message
1	Doctor not available at the selected time.

Sample Data:-

```
609   ----as a sample data we taken 7 appointments
610
611   EXEC BookAppointment @PatientID = 1, @DoctorID = 1, @AppointmentID = 433, @AppointmentDate = '2026-08-25', @AppointmentTime = '09:00:00';
612   EXEC BookAppointment @PatientID = 2, @DoctorID = 2, @AppointmentID = 434, @AppointmentDate = '2026-08-26', @AppointmentTime = '10:30:00';
613   EXEC BookAppointment @PatientID = 3, @DoctorID = 3, @AppointmentID = 435, @AppointmentDate = '2026-08-27', @AppointmentTime = '11:45:00';
614   EXEC BookAppointment @PatientID = 4, @DoctorID = 4, @AppointmentID = 436, @AppointmentDate = '2026-08-28', @AppointmentTime = '14:00:00';
615   EXEC BookAppointment @PatientID = 5, @DoctorID = 5, @AppointmentID = 437, @AppointmentDate = '2026-08-29', @AppointmentTime = '15:30:00';
616   EXEC BookAppointment @PatientID = 6, @DoctorID = 6, @AppointmentID = 438, @AppointmentDate = '2026-08-30', @AppointmentTime = '16:45:00';
617   EXEC BookAppointment @PatientID = 7, @DoctorID = 7, @AppointmentID = 439, @AppointmentDate = '2026-08-31', @AppointmentTime = '17:30:00';
618
```

3.10 Department table:-

departmentID:- it is the primary key which is unique for each department.

departmentName:- it is the department name of each department and it can not be null.

```
Create table department (
departmentID int primary key,
departmentName nvarchar(100) not null,
);
```

Sample data :-

1

```
INSERT INTO department (departmentID, departmentName)
VALUES
    (1, 'Cardiology'),
    (2, 'Neurology'),
    (3, 'Orthopedics'),
    (4, 'Oncology'),
    (5, 'Pediatrics'),
    (6, 'Internal Medicine'),
    (7, 'Surgery');
```

3.11 MedicalHistory Table:-

RecordID:- it is the primary key of each medical record and it is an integer column.

PatientID:- it is the foreign key referencing to PatientID in Patient table and it is an integer column.

DoctorID:- it is the foreign key referencing to DoctorID in Doctor table and it is an integer column.

AppointmentID:- it is the foreign key referencing to AppointmentID in Appointment table and it is an integer column.

Diagnoses:- it is the diagnose of the patient and this column is nvarchar and it cannot be null.

Allergies:- it is the allergies that patient might have and it can be null and this column is marked as nvarchar.

Medicines:- these are the medicines prescribe by the doctor and this column must be not null.

MedicinePrescribedDate:- it is the column that stores the date of prescribed medicine and it cannot be null.

RecordTimestamp:- it is the column that stores the timestamp of each medical record.

```
CREATE TABLE MedicalHistory (
    RecordID INT primary key,
    PatientID INT NOT NULL,
    DoctorID INT NOT NULL,
    AppointmentID INT NOT NULL,
    Diagnoses NVARCHAR(100) NOT NULL,
    Allergies NVARCHAR(100),
    Medicines NVARCHAR(500) not null,
    MedicinePrescribedDate DATE NOT NULL,
    RecordTimestamp DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID),
    FOREIGN KEY (AppointmentID) REFERENCES Appointment(AppointmentID)
);
```

3.12 DoctorReviews table:-

This is the table in which we record the reviews given by the patients after he/she attend the doctor. However how this table is used to store the patients reviews is defined in stored procedure which iam going to discuss later. But for now in this table I have defined

reviewId:- it is marked as int and act as a primary key for doctorReview table to adhere to 1st NF.

Review:- it is marked as nvarchar because it is the statement given by the patient about the doctor that how he/she treats in their appointed appointment.

AppointmentID:- it is the foreign key referencing AppointmentID in the appointment table to ensure normalization and adhere the link between the tables.

```
CREATE TABLE DoctorReviews(
  reviewID INT IDENTITY(1,1) PRIMARY KEY,
  Review NVARCHAR(200),
  AppointmentID INT FOREIGN KEY REFERENCES Appointment(AppointmentID)
);
```

3.13 DeletedAppointment Table:-

1

This table is used to store all the appointments whose status is completed. This will works when we apply trigger on appointmentStatus table that when the appointment status=Completed than it will be saved into DeletedAppointment table and delete from AppointmentStatus table. Here I have defined

DeletedStatusID;- it is marked as a identity(every times when it updates it will ID is changed by 1) and primary key.

AppointmentID:- it is marked a an integer and is used here as a foreign key referencing to the Appointment table to adhere normalization and to create a link between the 2 tables.

Status:- it is the status of appointment which is obviously completed.

DeletionDate:- it is the date when the appointment completed.

```
CREATE TABLE DeletedAppointment (
    DeletedStatusID INT IDENTITY(1,1) PRIMARY KEY,
    AppointmentID INT,
    Status VARCHAR(50),
    DeletionDate DATETIME DEFAULT GETDATE()
);
```

Store Procedure UpdateMedicalHistoryAndCompleteAppointment:-

Now I have created a stored procedure to update medical history. The logic behind this store procedure is that when the patient comes to doctor first of all system will check that the concern patient comes on his appointed appointment. If it is true than doctor should attend him otherwise it will shows Invalid Appointment. And than it would cross check the appointment with the appointment status table to check whether the appointment status is cancelled available or Booked. If the Appointment status='Booked ', than the doctor should see the patient and than update the Patient Medical history by prescribing medicines, diagnoses , Allergies etc etc.
When a medical record is updated, then it typically signifies that the appointmen is now completed, so the system will update the appointment status of the patient in the AppointmentStatus table to "COMPLETED".

```sql
CREATE PROCEDURE UpdateMedicalHistoryAndCompleteAppointment
 @DoctorID INT,
 @AppointmentID INT,
 @RecordID INT,
 @Diagnoses NVARCHAR(MAX),
 @Medicines NVARCHAR(MAX),
 @Allergies NVARCHAR(MAX),
 @Review NVARCHAR(200) = NULL -- Added parameter for review, default NULL if not provided
AS
BEGIN
 SET NOCOUNT ON;

 -- Verify if the appointment exists and is associated with the specified doctor
 IF NOT EXISTS (
  SELECT *
  FROM Appointment
  WHERE AppointmentID = @AppointmentID
   AND DoctorID = @DoctorID
 )
 BEGIN
  RAISERROR('Invalid appointment or doctor.', 16, 1);
  RETURN;
 END

 -- Check the current status of the appointment to ensure it is 'Booked'
 DECLARE @CurrentStatus NVARCHAR(50);
 SELECT @CurrentStatus = Status
 FROM AppointmentStatus
 WHERE AppointmentID = @AppointmentID;

 IF @CurrentStatus IS NULL OR @CurrentStatus <> 'Booked'
 BEGIN
  RAISERROR('Appointment is not in a valid state to update medical history.', 16, 1);
  RETURN;
 END

 -- Update the MedicalHistory
 INSERT INTO MedicalHistory (PatientID, RecordID, DoctorID, AppointmentID, Diagnoses, Allergies, Medicines, MedicinePrescribedDate)
 SELECT PatientID, @RecordID, @DoctorID, @AppointmentID, @Diagnoses, @Allergies, @Medicines, GETDATE()
 FROM Appointment
 WHERE AppointmentID = @AppointmentID;

 -- Update the AppointmentStatus to 'Completed'
 UPDATE AppointmentStatus
 SET Status = 'Completed'
 WHERE AppointmentID = @AppointmentID;
 -- Return a success message
 SELECT 'Medical history updated, appointment marked as completed' AS Message;
END;
GO
```

Sample data:-

```sql
--- sample data for updating medical history
INSERT INTO MedicalHistory (RecordID, PatientID, DoctorID, AppointmentID, Diagnoses, Allergies, Medicines, MedicinePrescribedDate)
VALUES
(1, 1, 1, 433, 'Type 2 Diabetes', 'Peanuts', 'Metformin', '2023-01-15'),
(2, 2, 2, 434, 'Hypertension', 'Penicillin', 'Lisinopril', '2023-02-20'),
(3, 3, 3, 435, 'Asthma', 'Dust', 'Albuterol', '2023-03-10'),
(4, 4, 4, 436, 'Breast Cancer', NULL, 'Tamoxifen', '2023-04-05'),
(5, 5, 5, 437, 'Osteoarthritis', 'Shellfish', 'Ibuprofen', '2023-05-12'),
(6, 6, 6, 438, 'Bronchitis', 'Mold', 'Amoxicillin', '2023-06-20'),
(7, 7, 7, 439, 'Migraine', 'None', 'Sumatriptan', '2023-07-08');
```

Execution:-

For the execution of above first of all I book an appointment

1

```
602
603  EXEC BookAppointment @PatientID = 2, @DoctorID = 4, @AppointmentID = 1237, @AppointmentDate = '2024-09-22', @AppointmentTime = '02:00:00';
604
605
```

Results | Messages

Message
Appointment booked successfully.

As you can see from above that we have book an appointment with AppointmentID=1237 and DoctorID=4;
Now we will execute the **UpdateMedicalHistoryAndCompleteAppointment** store procedure with the AppointmentID=1237 and DoctorID=4, so that it will execute successfully

```
593  -- Execute the stored procedure with test data
594  EXEC UpdateMedicalHistoryAndCompleteAppointment
595      @DoctorID = 4,
596      @AppointmentID =1237,
597      @RecordID = 25,
598      @Diagnoses = 'Gastroenterologists',
599      @Medicines = 'Test Medicine',
600      @Allergies = 'Test Allergy';
601
602
```

Results | Messages

Message
Medical history updated, appointment marked as co...

Yes now you can see that the medical history is updated successfully and now we would check the Appointment Status for AppointmentID=1237 must be changed to completed

7	7	871	Completed	2024-03-22 18:23:33.203
8	8	871	Completed	2024-03-23 16:28:53.847
9	9	1287	Completed	2024-03-23 16:30:31.813
10	10	1237	Completed	2024-03-23 16:35:40.527
11	11	36	Completed	2024-03-23 16:39:50.383
12	12	326	Completed	2024-03-23 16:54:00.970

Creating Roles and Grant permission:-
Now in this case I made a scheme with the name of Patient Medical history and transfer the stored procedure UpdateMedicalHistoryAndCompleteAppointment to this schema

and then I create a username and password with the name of doctor and than make a Role with the name of Doctor and then add the created user name in the role. However my approach for this was that every doctor has a login password and username and when he login into the portal he would be grant access to update medical history of the patients. But here I have done only for 1 doctor.

```
446  ----creating user login for each doctor to update medical history
447  CREATE SCHEMA
448  MedicalHistoryOfPatients;
449  GO
450  ALTER SCHEMA MedicalHistoryOfPatients TRANSFER dbo.UpdateMedicalHistoryAndCompleteAppointment;
451
452  CREATE LOGIN DrMichaelJohnson
453  WITH PASSWORD = 'djsegvwdue20!';
454
455
456
457  --- creating role doctor
458  create role Doctor;
459  GRANT SELECT, UPDATE, INSERT ON SCHEMA :: MedicalHistoryOfPatients
460  TO Doctor
461
462  --- now adding above user login into doctor role
463  ALTER ROLE Doctor ADD MEMBER DrMichaelJohnson;
464
```

4. Normalization

Normalization is the process of structuring the database in a way that it reduces the data redundancy and improves integrity of data. So now in this assignment I do normalization in this way

4.1 1NF:-

Each table above has unique identifier like PatientID, RecordID, DoctorID, AppointmentID, DepartmentID etc which ensure that each table has single concept.

4.2 2NF:-

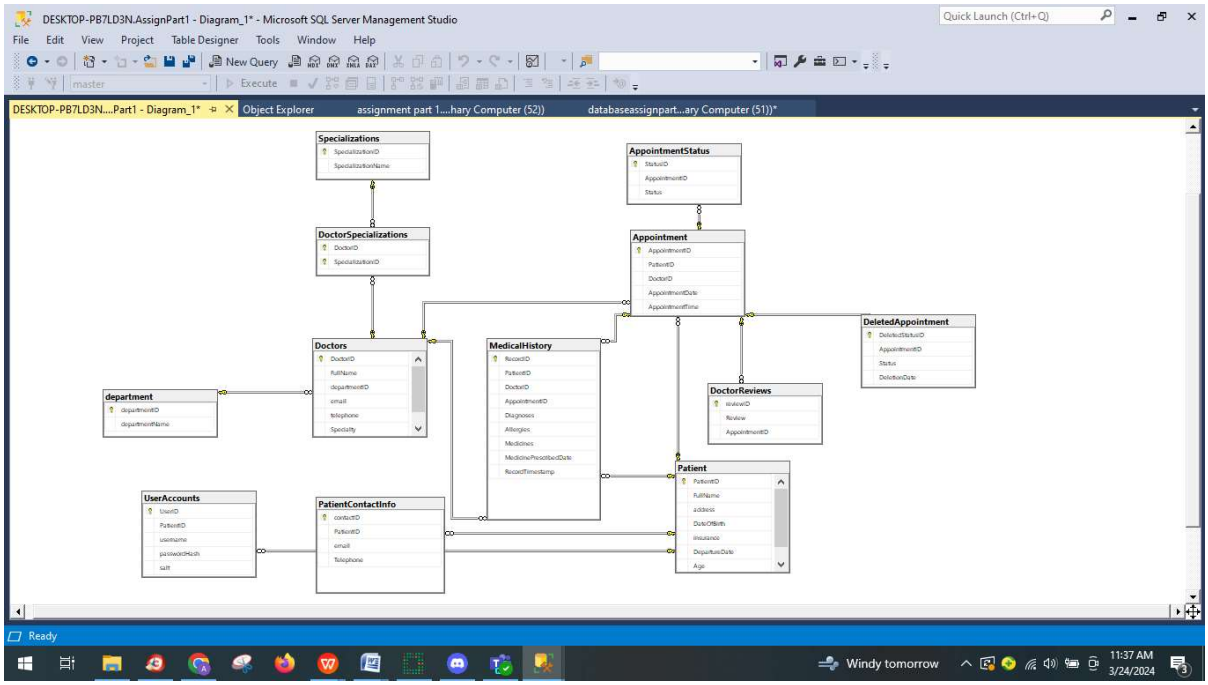
- Tables above are in 1NF.
- All non-key attributes are fully functional and dependent on primary key.

4.3 3NF:-

- Tables above are in 2NF.
- All attributes only depend on primary key and not on other non-primary attributes

- We have removed transitive dependency for example patient contact info is separated from main patent details. Doctor specialization tables addresses many to many relationship and do not require multiple update across the database.

4.4 ER Diagram:-



Task # 02

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

Now to address the answer of above I have added a check constraint using the GetDate command in the appointment table that will ensure that the Appointment date would not be from the past.

```
CREATE TABLE Appointment (
    AppointmentID INT  PRIMARY KEY,
    PatientID INT FOREIGN KEY (PatientID) REFERENCES Patient(PatientID) not null,
    DoctorID INT FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)  not null,
    AppointmentDate DATE not null,
    AppointmentTime TIME not null,
);
Alter table appointment
add  CONSTRAINT CHECK_AppointmentDate CHECK (AppointmentDate >= CAST(GETDATE() AS DATE));
```

1

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

Now for this first of all I added a age column in the Patient table where the age of each Patient stores

```
4
5  Create table Patient(
6  PatientID int primary key,
7  FullName nvarchar(50) not null,
8  address nvarchar(250) not null,
9  DateOfBirth date not null,
10 insurance nvarchar(250) not null,
11 DepartureDate DATE NULL,
12 );
13 ALTER TABLE Patient
14 ADD Age INT;
15
16
```

And then I created a trigger on Patient table which will takes patients date of birth using patientID as primary key from the patient table and then compare it with the present date to calculate the age and than store that age in the Age column of the Patient table

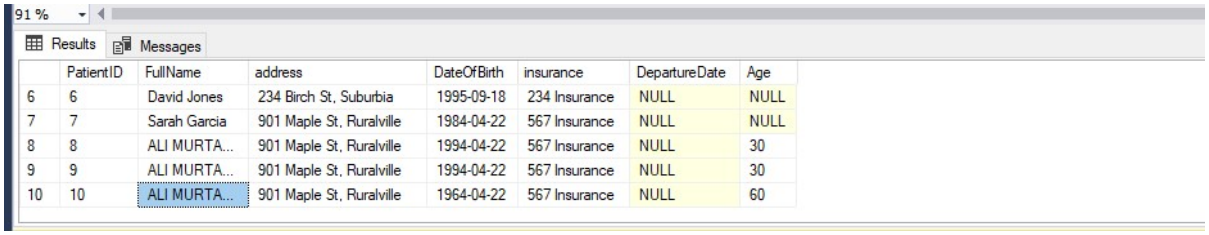
```
294  ---List all the patients with older than 40 and have Cancer in diagnosis.
295  CREATE TRIGGER CalculatePatientAge
296  ON Patient
297  AFTER INSERT
298  AS
299  BEGIN
300      SET NOCOUNT ON;
301
302      DECLARE @PatientID INT, @DateOfBirth DATE;
303
304      -- Get the newly inserted patient's ID and date of birth
305      SELECT @PatientID = PatientID, @DateOfBirth = DateOfBirth
306      FROM inserted;
307
308      -- Calculate the age
309      declare @age int;
310      SET @Age = DATEDIFF(YEAR, @DateOfBirth, GETDATE());
311
312      -- Update the Patient table with the calculated age
313      UPDATE Patient
314      SET Age = @Age
315      WHERE PatientID = @PatientID;
316  END;
```

Execution:-

Now my sample data for Medical history does not contan any patient with cancer so we add it manual through a stored procedure.

As the above trigger will work by after insert mean now when we insert a new patient than it will calculate the age now.

So I add a patient name with my name i.e Syed Ali Murtaza and than the triggers run and shows me the output with age.



	PatientID	FullName	address	DateOfBirth	insurance	DepartureDate	Age
6	6	David Jones	234 Birch St, Suburbia	1995-09-18	234 Insurance	NULL	NULL
7	7	Sarah Garcia	901 Maple St, Ruralville	1984-04-22	567 Insurance	NULL	NULL
8	8	ALI MURTA...	901 Maple St, Ruralville	1994-04-22	567 Insurance	NULL	30
9	9	ALI MURTA...	901 Maple St, Ruralville	1994-04-22	567 Insurance	NULL	30
10	10	ALI MURTA...	901 Maple St, Ruralville	1964-04-22	567 Insurance	NULL	60

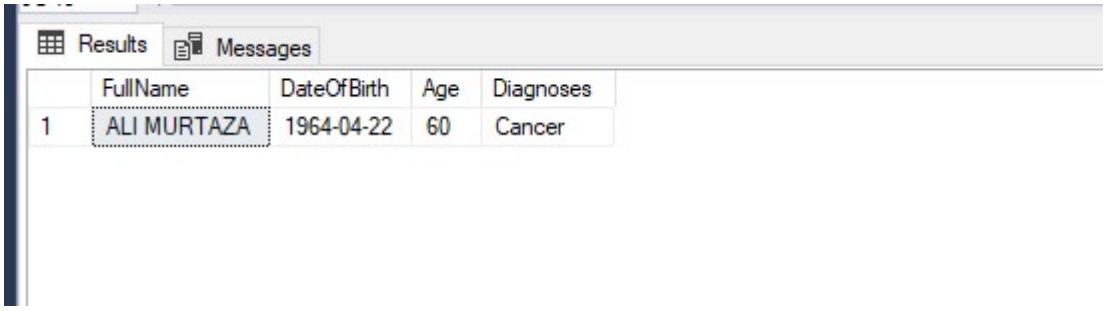
And than I book my appointment using my PatientID as follow

And than the patient go to doctor my diagnoses and the doctor diagnoses the Cancer

And now the hospital wants the data of patients having Cancer and their age must be greater than 40. For this purpose I write a queery that will inner join the medical history table with the Patient table so that it will retrieve all the information of the patient who is suffering from Cancer or any other specific desease we can write

```
324 SELECT p.FullName, p.DateOfBirth, p.Age, mh.Diagnoses
325 FROM Patient p
326 INNER JOIN MedicalHistory mh ON p.PatientID = mh.PatientID
327 WHERE p.Age > 40
328 AND mh.Diagnoses LIKE '%Cancer%';
329
```

ALAS.. the Patient with my name is only the Patient who diagnoses Cancer



Results | Messages

	FullName	DateOfBirth	Age	Diagnoses
1	ALI MURTAZA	1964-04-22	60	Cancer

5. **The hospital also requires stored procedures or user-defined functions to do the**

following things:

a) Search the database of the hospital for matching character strings by name of medicine. Results should be sorted with most recent medicine prescribed date first.

b) Return a full list of diagnosis and allergies for a specific patient who has an appointment today (i.e., the system date when the query is run)

c) Update the details for an existing doctor

d) Delete the appointment who status is already completed.

a)

I have created a store procedure in which I pass a parameter with the name of the medicine and it will search for similar character strings and give me the output with all the attributes present in medical history table with the MedicinePrescribeDate is Desc order.

1

```
331
332 CREATE PROCEDURE SearchMedicineByName
333     @MedicineName NVARCHAR(500)
334 AS
335 BEGIN
336     SELECT mh.PatientID, mh.DoctorID, mh.AppointmentID, mh.Medicines, mh.MedicinePrescribedDate
337     FROM MedicalHistory mh
338     WHERE mh.Medicines LIKE '%' + @MedicineName + '%'
339     ORDER BY mh.MedicinePrescribedDate DESC;
340 END;
341
```

Execution and result:-

According to the sample data we populate into the medical history table it will shows result

```
341
342 EXEC SearchMedicineByName @MedicineName = 'Metformin';
343
```

3 %

Results | Messages

	PatientID	DoctorID	AppointmentID	Medicines	MedicinePrescribedDate
1	1	1	433	Metformin	2023-01-15

If we change the medicine name of the medicine then

```
342 EXEC SearchMedicineByName @MedicineName = 'Sumatriptan';
343
```

3 %

Results | Messages

	PatientID	DoctorID	AppointmentID	Medicines	MedicinePrescribedDate
1	7	7	439	Sumatriptan	2023-07-08

b)

I have created a function which is used to return diagnoses and allergies for all the patients who have appointment today

In this function I make a inner join on appointmentID on both appointment table and Patient table and than create an inner join on patientID of medical history table and Patient table. I done this because as our model is 3NF so there must be similar keys between tables which I used to join these tables and get me desire results.

```sql
----) Return a full list of diagnosis and allergies for a specific patient who has an appointment today (i.e., the system date when the query is run)
CREATE FUNCTION GetTodayAppointmentDetails()
RETURNS TABLE
AS
RETURN
(
    SELECT
        p.PatientID,
        p.FullName,
        a.AppointmentDate,
        a.AppointmentTime,
        mh.Diagnoses,
        mh.Allergies
    FROM Patient p
    INNER JOIN Appointment a ON p.PatientID = a.PatientID
    INNER JOIN MedicalHistory mh ON p.PatientID = mh.PatientID
    WHERE CAST(a.AppointmentDate AS DATE) = CAST(GETDATE() AS DATE)
);
```

Execution:-

Its execution gives null values because in our sample data there is no appointment today but the logic for making the function correct.

c)

I have created a stored procedure for updating the details of the existing doctor which is already in the doctor table. First we have to pass some parameters and than the system will check that are the parameters are matching with the existing doctor in the doctor table using DoctorID as primary key. If it exists than it will update the details of the doctor. I have also applied a check which is used to determine that whether any row is updated or not. If any row is not updated than it will rise the error that Doctor ID not found

```
----c) Update the details for an existing doctor
CREATE PROCEDURE UpdateDoctorDetails
(
@DoctorID INT,
@FullName NVARCHAR(200),
@DepartmentID INT,
@Email NVARCHAR(100),
@Telephone BIGINT,
@Specialty NVARCHAR(100)
)
AS
BEGIN
-- Update the Doctors table
UPDATE Doctors
SET FullName = @FullName,
DepartmentID = @DepartmentID,
Email = @Email,
Telephone = @Telephone,
Specialty = @Specialty
WHERE DoctorID = @DoctorID;
```

```

-- Check if rows were affected (optional)
IF @@ROWCOUNT = 0
BEGIN
RAISERROR('Doctor with ID %d not found.', 16, 1, @DoctorID);
RETURN;
END
SELECT 'Doctor details updated successfully.' AS Message;
END;
```

Execution:-

```
EXEC UpdateDoctorDetails
@DoctorID = 2,  -- Replace with the actual doctor ID
@FullName = 'Dr. Jane Smith',
@DepartmentID = 1,  -- Replace with the department ID
@Email = 'jane.smith@hospital.com',
@Telephone = 1234567890,
@Specialty = 'Cardiology';
```

83 %

Results | Messages

	Message
1	Doctor details updated successfully.

d)

1

I have not created a UDF or store procedure to delete appointment whose status is completed. I have made a trigger for this purpose. The logic behind making the trigger is that it will executed automatically when the appointment status is equal to completed. However if I make the store procedure than I have to call it every time but for now whenever the appointment is completed than it will move to deleted appointment table and removes permanently from the appointment status table.

```
CREATE TRIGGER trg_AfterStatusUpdate
ON AppointmentStatus
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if the status has been updated to 'Completed'
    IF EXISTS (SELECT 1 FROM inserted i JOIN deleted d ON i.StatusID = d.StatusID WHERE i.Status = 'Completed')
    BEGIN
        -- Insert the completed appointment record into DeletedAppointment
        INSERT INTO DeletedAppointment (AppointmentID, Status)
        SELECT AppointmentID, Status
        FROM inserted
        WHERE Status = 'Completed';

        -- Delete the completed appointment record from AppointmentStatus
        DELETE FROM AppointmentStatus
        WHERE StatusID IN (SELECT StatusID FROM inserted WHERE Status = 'Completed');
    END
END;
GO
```

6. **The hospitals wants to view the appointment date and time, showing all previous**

and current appointments for all doctors, and including details of the department (the doctor is associated with), doctor's specialty and any associate review/feedback

given for a doctor. You should create a view containing all the required information.

Yes I have created a view for this purpose in which I make a inner join on the primary keys of appointment and doctor table. And than join the doctor and department table on department ID. Left join the appointment and doctorReviews table on AppointmentID. Joining the appointment and Patient tables on PatientID.Left join the appointment and deleted appointments tables on appointmentID.

```sql
CREATE VIEW DoctorsAppointmentsDetails AS
SELECT
    d.FullName AS DoctorName,
    d.Specialty,
    dep.departmentName AS Department,
    a.AppointmentDate,
    a.AppointmentTime,
    dr.Review AS DoctorReview,
    p.FullName AS PatientName,
    p.PatientID,
    da.Status AS DeletedStatus,
    da.DeletionDate AS DeletionDate
FROM Appointment a
INNER JOIN Doctors d ON a.DoctorID = d.DoctorID
INNER JOIN department dep ON d.departmentID = dep.departmentID
LEFT JOIN DoctorReviews dr ON a.AppointmentID = dr.AppointmentID
INNER JOIN Patient p ON a.PatientID = p.PatientID
LEFT JOIN DeletedAppointment da ON a.AppointmentID = da.AppointmentID;
```

Execution and results:-

```sql
SELECT * FROM DoctorsAppointmentsDetails
ORDER BY AppointmentDate DESC, AppointmentTime DESC;
```

Results | Messages

	DoctorName	Specialty	Department	AppointmentDate	AppointmentTime	DoctorReview	PatientName	PatientID	DeletedStatus	DeletionDate
7	Dr. Michael Johnson	Cardiologist	Cardiology	2026-08-25	09:00:00.0000000	NULL	John Doe	1	NULL	NULL
8	Dr. Michael Johnson	Cardiologist	Cardiology	2025-09-25	11:00:00.0000000	NULL	John Doe	1	Completed	2024-03-23...
9	Dr. Michael Johnson	Cardiologist	Cardiology	2025-07-25	11:00:00.0000000	NULL	John Doe	1	Completed	2024-03-23...

7. **Create a trigger so that the current state of an appointment can be changed to**

available when it is cancelled

```
138 CREATE TRIGGER TR_CancelledAppointment
139 ON AppointmentStatus
140 AFTER UPDATE
141 AS
142 BEGIN
143     IF UPDATE(Status)
144     BEGIN
145         DECLARE @CancelledStatus VARCHAR(50) = 'Cancelled';
146         UPDATE a
147         SET a.Status = 'Available'
148         FROM AppointmentStatus AS a
149         INNER JOIN inserted AS i ON a.AppointmentID = i.AppointmentID
150         WHERE i.Status = @CancelledStatus;
151     END
152 END;
```

8. Write a select query which allows the hospital to identify the number of completed appointments with the specialty of doctors as 'Gastroenterologists'.

```
433 CREATE PROCEDURE CountCompletedGastroAppointments
434 AS
435 BEGIN
436     SELECT COUNT(da.AppointmentID) AS CompletedGastroAppointments
437     FROM DeletedAppointment da
438     INNER JOIN Appointment a ON da.AppointmentID = a.AppointmentID
439     INNER JOIN Doctors d ON a.DoctorID = d.DoctorID
440     WHERE d.Specialty = 'Gastroenterologists';
441 END;
442 GO
443
```

Execution and results:-

```
444 EXEC CountCompletedGastroAppointments;
445
```

Results | Messages

	CompletedGastroAppointments
1	5

Within your report, you will also need to provide your client with advice and guidance on:

- **Data integrity and concurrency**
- **Database security**
- **Database backup and recovery**

1

Data integrity and concurrency

In this assignment I have added primary keys and foreign keys in tables to maintain data integrity. I have used triggers(like calculatePatientAge) that will ensure concurrency and data accuracy over time. I have also used stored procedure like medical history and updating Appointment status, ensure they are wrapped in transactions to maintain data consistency.

Database security

In this assignment I have used hashing passwords with a salt which is critical database security measure.

I have also added ROLE BASE ACCESS control like I have created a role (Doctor) and assign user to that role, and then grant the update permission to Doctor for updating medical record of patients.

While explicitly not shown, but I ensure that all user input is properly sanitized to avoid SQL injection vulnerablilities.

Database backup and recovery

I have performed database backup and also attached the backup files.