

**AIRIOHUODION CLEMENT**

**APRIL 24, 2024**

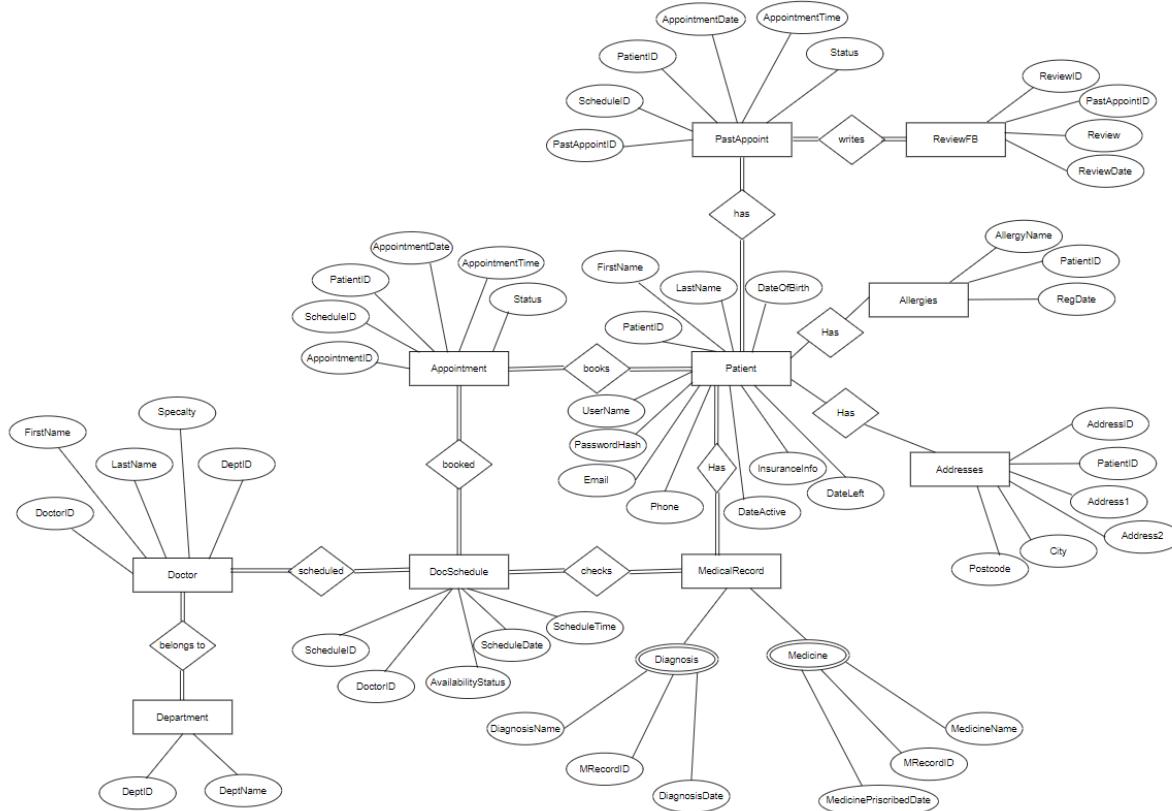
## **INTRODUCTION**

As a database developer consultant for a hospital, I'm to create a complete database system that satisfies the needs of the customer. To effectively store and handle data on patients, medical doctor, medical records, appointments and departments, the hospital is creating a new database system. Particular client requirements were obtained at the first meeting in order to make sure that the database system meets the demands of both patients and healthcare practitioners. By making patient registration, appointment scheduling, medical record administration, and feedback gathering easier, this database system will expedite healthcare operations and enhance patient care.

In this task the first step I've decided to take is to prepare an ER proposed database design into 3NF, with the Entities, attributes and the relationships between them, taking into consideration the principles of database design which were thought in lecture. Below is the sketch of my proposed database tables and attributes.

## PART ONE:

### Designing and normalizing my proposed database into 3NF



### ER DESIGN

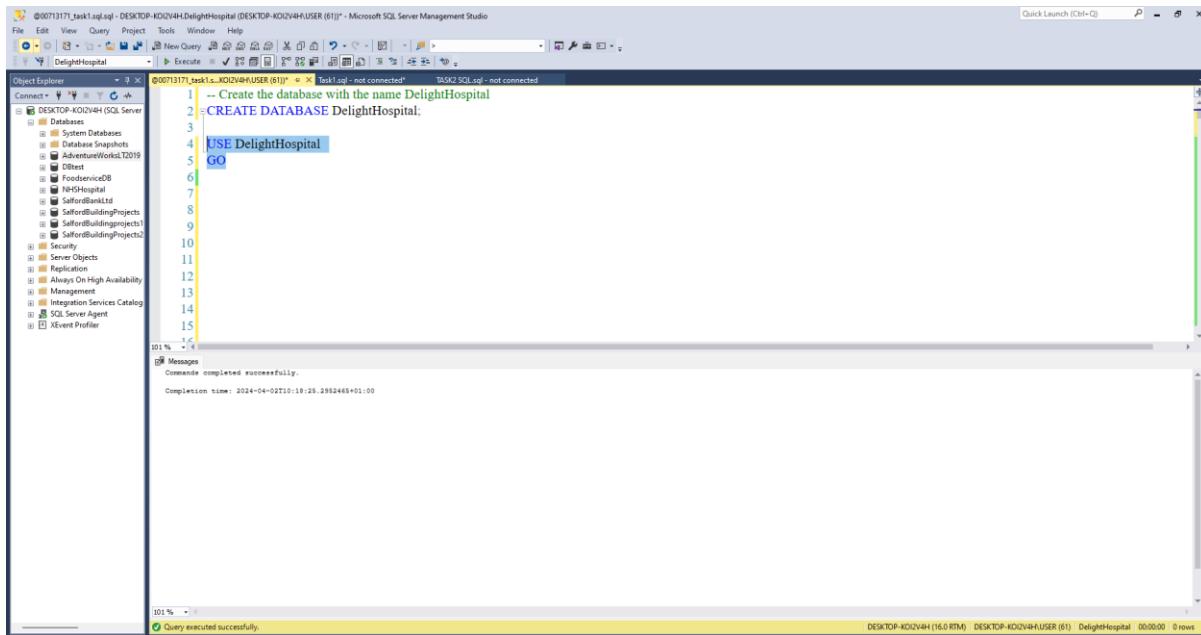
**Figure 1.1.1:**

Based on the request by the Hospital my design has been structured to support the hospital management system. To further explain the purpose of my normalization let me start by explaining the purpose of the entities suggested.

1. **Patient:** It keeps records of patient information, such as contact details, insurance information, and active/inactive status.
2. **Doctor:** Contains information about physicians, such as their names, specializations, and departmental affiliations.

3. **Department:** Provides details on the various hospital department and each doctor employed to that department.
4. **Appointment:** Keeps track of patient appointment with physicians.
5. **PastAppointment:** Holds historical information about previous appointments, such as the patients details and status.
6. **DocSchedule:** This program keeps track of doctor's availability for appointments as well as their schedules.
7. **Addresses:** Keeps track of patient addresses
8. **MedicalRecord:** Contains medical records associated with patients, including diagnosis and prescribed medicines.
9. **Diagnosis:** Documents medical record related to diagnoses of patients from tests.
10. **Medicine:** Keeps track of Prescribed medications in relation to medical records.
11. **ReviewFB:** Holds examine patient comments related to previous meetings with doctors they have met.
12. **Allergies:** It provides medical Doctors with crucial information on patients' allergy, ensuring their safety and the appropriate treatment course.

## CREATING DELIGHTHOSPITAL DATABASE



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a connection to 'DESKTOP-KD2V4H' is selected, showing databases like 'master', 'model', 'msdb', 'tempdb', 'AdventureworksLT', 'AdventureworksDW', 'AdventureworksLTReport', 'AdventureworksDWReport', 'masterdb', 'SafordBuildingProjects1', 'SafordBuildingProjects2', 'SafordBuildingProjects3', 'SafordBuildingProjects4', and 'SafordBuildingProjects5'. A new query window titled '(0)00713171\_task1.sql - DESKTOP-KD2V4H.DelightHospital (DESKTOP-KD2V4H\USER (6)) - Microsoft SQL Server Management Studio' is open. The T-SQL code entered is:

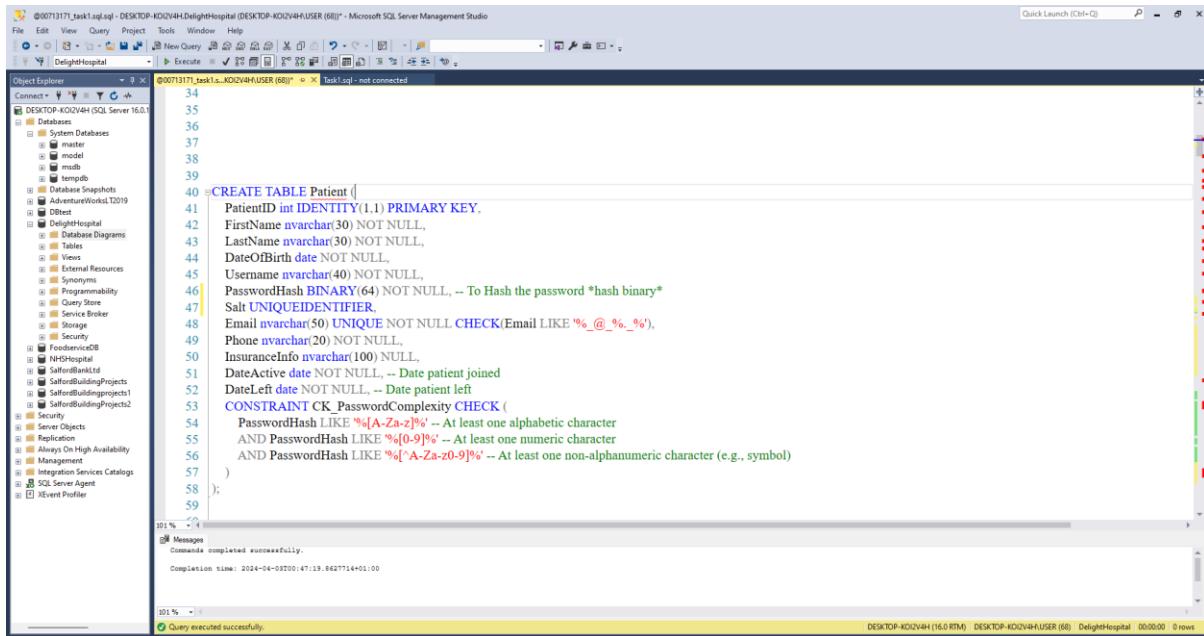
```
1 -- Create the database with the name DelightHospital
2 CREATE DATABASE DelightHospital;
3
4 USE DelightHospital;
5 GO
6
7
8
9
10
11
12
13
14
15
```

In the Messages pane at the bottom, it says 'Command completed successfully.' and 'Completion time: 2024-04-02T10:18:28.2932448+01:00'. The status bar at the bottom right shows 'DESKTOP-KD2V4H (16.0 RTM) | DESKTOP-KD2V4H\USER (6) | DelightHospital | 00:00:00 | 0 rows'.

**Figure 1.1.2:** In this first step I have written a TSQL code to create a database with name **DelightHospital** and with the ‘USE’ query to change from the current database to the just created database.

## Normalization Process for Patient table:

Given that atomic values are listed in each column, the patient table looks to be in 1NF. I have made sure there are no partial dependencies in order to guarantee 2NF. Every attribute in the patient table is dependent on the patient ID primary key. I had to get rid of transitive dependencies for 3NF. The patient table represents each patient and contains record of their contact and personal data.



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists databases, including 'DelightHospital'. The central pane displays the T-SQL code for creating the 'Patient' table:

```
CREATE TABLE Patient (
    PatientID int IDENTITY(1,1) PRIMARY KEY,
    FirstName nvarchar(30) NOT NULL,
    LastName nvarchar(30) NOT NULL,
    DateOfBirth date NOT NULL,
    Username nvarchar(40) NOT NULL,
    PasswordHash BINARY(64) NOT NULL, -- To Hash the password *hash binary*
    Salt UNIQUEIDENTIFIER,
    Email nvarchar(50) UNIQUE NOT NULL CHECK>Email LIKE '%_@_%.%',
    Phone nvarchar(20) NOT NULL,
    InsuranceInfo nvarchar(100) NULL,
    DateActive date NOT NULL, -- Date patient joined
    DateLeft date NOT NULL, -- Date patient left
    CONSTRAINT CK_PasswordComplexity CHECK (
        PasswordHash LIKE '%[A-Za-z-%]' -- At least one alphabetic character
        AND PasswordHash LIKE '%[0-9-%]' -- At least one numeric character
        AND PasswordHash LIKE '%[^A-Za-z0-9-%]' -- At least one non-alphanumeric character (e.g., symbol)
    );
);
```

The status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.1.3:** Justification

**PatientID** is the primary key for each patient record, auto-increasing. The **IDENTITY(1,1)** column uniquely identifies each patient. Other columns include **FirstName**, **LastName**, **Username**, **Email**, **Phone**, and **Insuranceinfo**, with string data stored in **nvarchar** and date values in **DateOfBirth**, **DateActive**, and **DateLeft** columns.

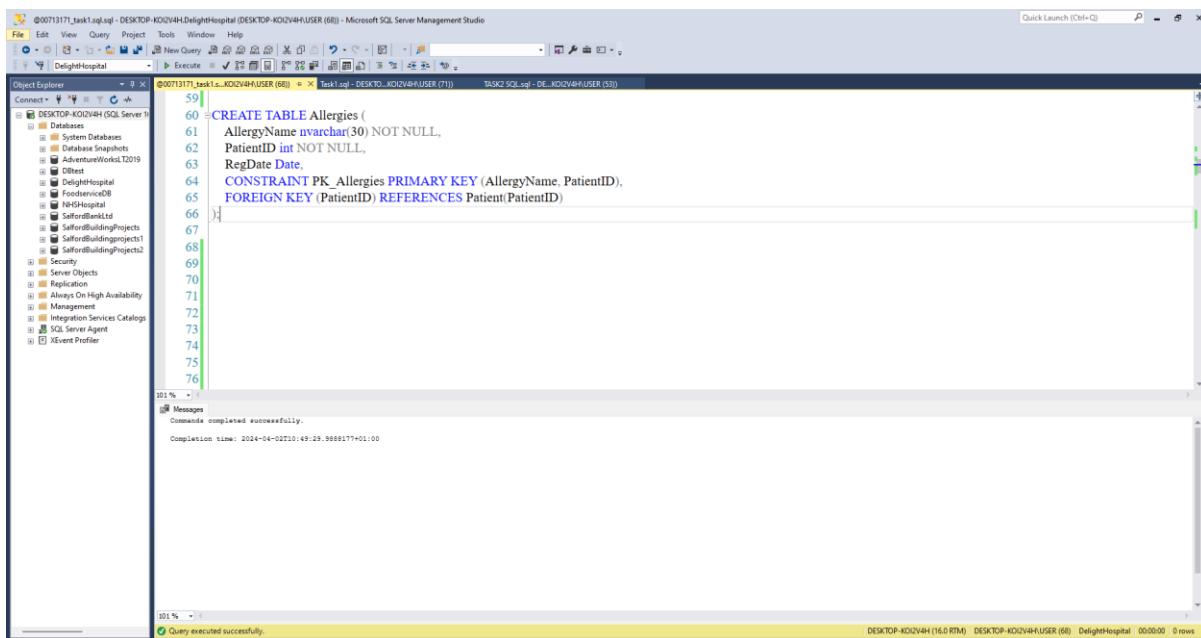
**PasswordHash** is stored as '**Binary(64)**' efficient storage of hashed password data. Unique salts are created using '**UNIQUEIDENTIFIER**'.

**Not null** constraints ensure required fields are filled in, while **CHECK** constraint ensures password complexity for security.

Unique constraints for emails ensure they contain the @ symbol and at least one character after the dot.

### **Normalization Process for Allergies table:**

The Allergies table is now in the 3NF, free of any partial or transitive dependencies, and designed to preserve data integrity thanks to this normalizing procedure.



A screenshot of Microsoft SQL Server Management Studio (SSMS) showing the creation of the Allergies table. The Object Explorer on the left shows the database structure, including the 'Allergies' table under the 'Tables' node. The central pane displays the T-SQL code for creating the table:

```
CREATE TABLE Allergies (
    AllergyName nvarchar(30) NOT NULL,
    PatientID int NOT NULL,
    RegDate Date,
    CONSTRAINT PK_Allergies PRIMARY KEY (AllergyName, PatientID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
)
```

The status bar at the bottom indicates "Query executed successfully." and provides execution details: "Completion time: 2024-04-04 22:10:49.29 9998177+01:00".

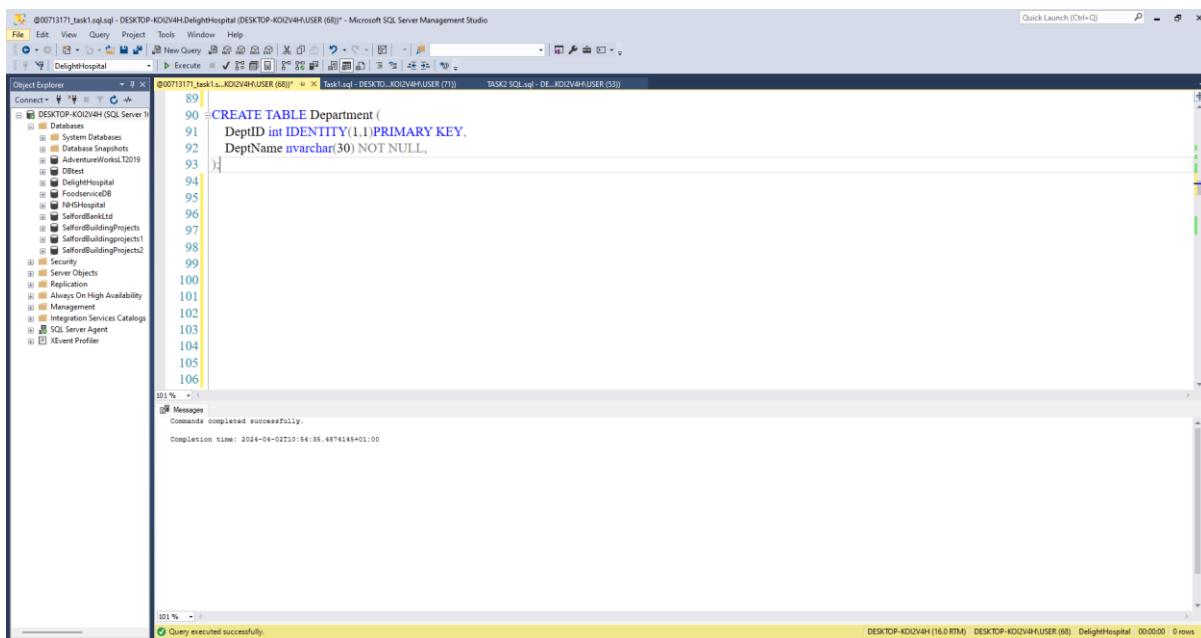
**Figure 1.1.4:** Justification

The table uses a **nvarchar** datatype for allergy names, allowing for a range of lengths. A **foreign key** is created for patients linked to allergies, and a column for Registration Date keeps dates for each allergy. The composite key (**AllergyName**, **PatientID**) ensures uniqueness in joint relations to MedicalRecord. The foreign key

**(PatientID)** ensures referential integrity between the Allergy and Patient tables, referencing the Patient column in the Patient table.

### **Normalization Process for Department table:**

There appears to be 1NF, 2NF, 3NF in the Department table. There are no transitive or partial dependencies, and it contains atomic values. The Department table facilitates departmental management and categorization by providing information about various hospital departments.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'DESKTOP-KOIZV4H' database is selected. A query window titled 'Task1.sql - DESKTOP-KOIZV4H\USER (66)' is open, displaying the following SQL code:

```
CREATE TABLE Department (
    DeptID int IDENTITY(1,1) PRIMARY KEY,
    DeptName nvarchar(30) NOT NULL,
)
```

The code is numbered from 89 to 106. Below the code, the 'Messages' pane shows the execution results:

```
Commands completed successfully.  
Completion time: 2024-04-02T10:14:35.4874145+01:00
```

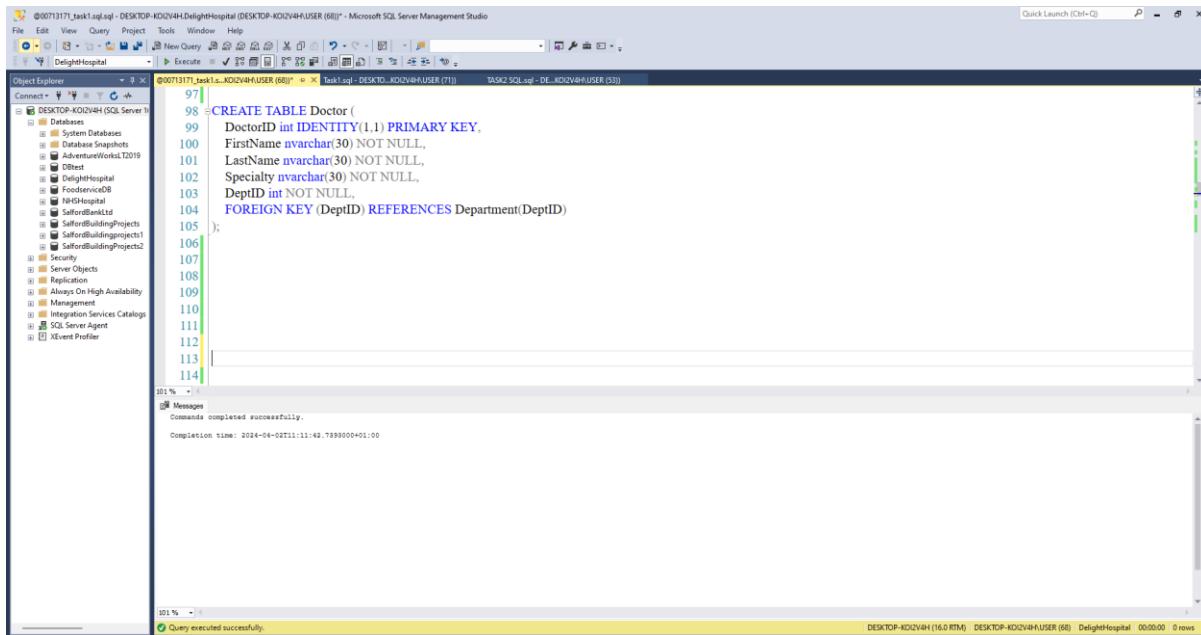
At the bottom of the screen, a status bar indicates: DESKTOP-KOIZV4H (16.0 RTM) | DESKTOP-KOIZV4H\USER (66) | DelightHospital | 00:00:00 | 0 rows.

**Figure 1.1.5:** Justification

DeptID is the **primary key** for each department record, auto incremented in **integers**, it uniquely identifies each department using IDENTITY(1,1). DeptName, a string data type, is typically stored in **nvarchar**, accepting variable lengths and international characters. The DeptName table does not have **null** constraints.

## Normalization Process for Doctor table:

The Doctor table appears to be in 1NF, 2NF, and 3NF, just like the Patient table. Physicians' data, such as names, specializations, and department affiliations, are kept in the Doctor table. The hospital can effectively manage doctor related data thanks to this table.



A screenshot of Microsoft SQL Server Management Studio (SSMS) showing the creation of a table named 'Doctor'. The code is as follows:

```
98: CREATE TABLE Doctor (
99:     DoctorID int IDENTITY(1,1) PRIMARY KEY,
100:    FirstName nvarchar(30) NOT NULL,
101:    LastName nvarchar(30) NOT NULL,
102:    Specialty nvarchar(30) NOT NULL,
103:    DeptID int NOT NULL,
104:    FOREIGN KEY(DeptID) REFERENCES Department(DeptID)
105: );
106:
107:
108:
109:
110:
111:
112:
113:
114:
```

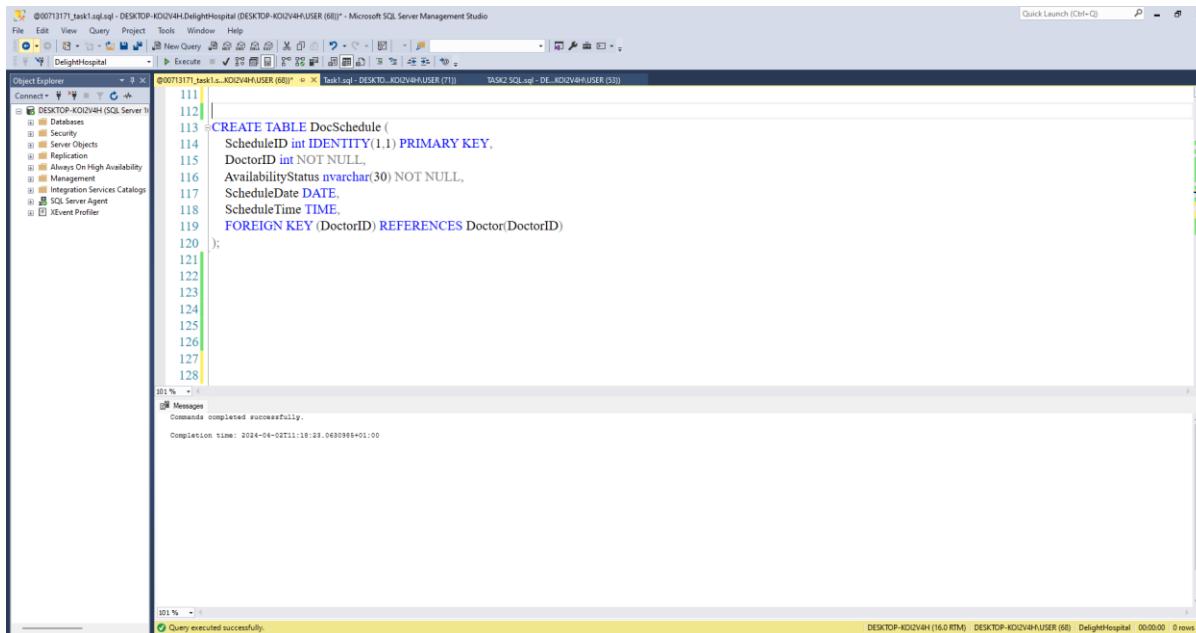
The 'Messages' pane shows the command completed successfully with a completion time of 2024-04-02T11:11:42.7390000+01:00. The status bar at the bottom indicates 'Query executed successfully'.

**Figure 1.1.6:** Justification

DoctorID is the primary key for each doctor record, auto incremented (int). it uniquely identifies each doctor using IDENTITY(1,1). FirstName, LastName, and Specialty are string data stored in nvarchar, with variable lengths and international characters. DeptID is the foreign key for the Department table, indicating the Department each Doctor is affiliated with. NOT NULL constraints ensure data integrity by providing necessary fields like FirstName, LastName, Specialty and Dept.

## Normalization Process for DocSchedule table:

There are 3NF, 2NF, and 1NF in the DocSchedule table, no transitive relationships and every attribute is dependent on the primary key, ScheduleID. The DocSchedule table helps the hospital efficiently arrange physician schedules by managing doctors' calendars, including their availability for appointment



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'DelightHospital' database is selected. In the center pane, a script window displays the SQL code for creating the 'DocSchedule' table:

```
111 |
112 |
113 CREATE TABLE DocSchedule (
114     ScheduleID int IDENTITY(1,1) PRIMARY KEY,
115     DoctorID int NOT NULL,
116     AvailabilityStatus nvarchar(30) NOT NULL,
117     ScheduleDate DATE,
118     ScheduleTime TIME,
119     FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
120 );
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
```

The 'Messages' pane at the bottom shows the command completed successfully with a timestamp of 2024-04-02T11:18:23.0480985+01:00.

**Figure 1.1.7: Justification**

Since **ScheduleID** distinguishes every Doctor Schedule record individually, it is selected as the primary key, it is auto incremented (int). Every DocSchedule is identified specifically and uniquely in this column using IDENTITY(1,1).

**AvailabilityStatus** have string data and is typically stored in datatypes like nvarchar, which accept variable lengths and support international characters. While the **DoctorID** carries (int) datatype to take integer only

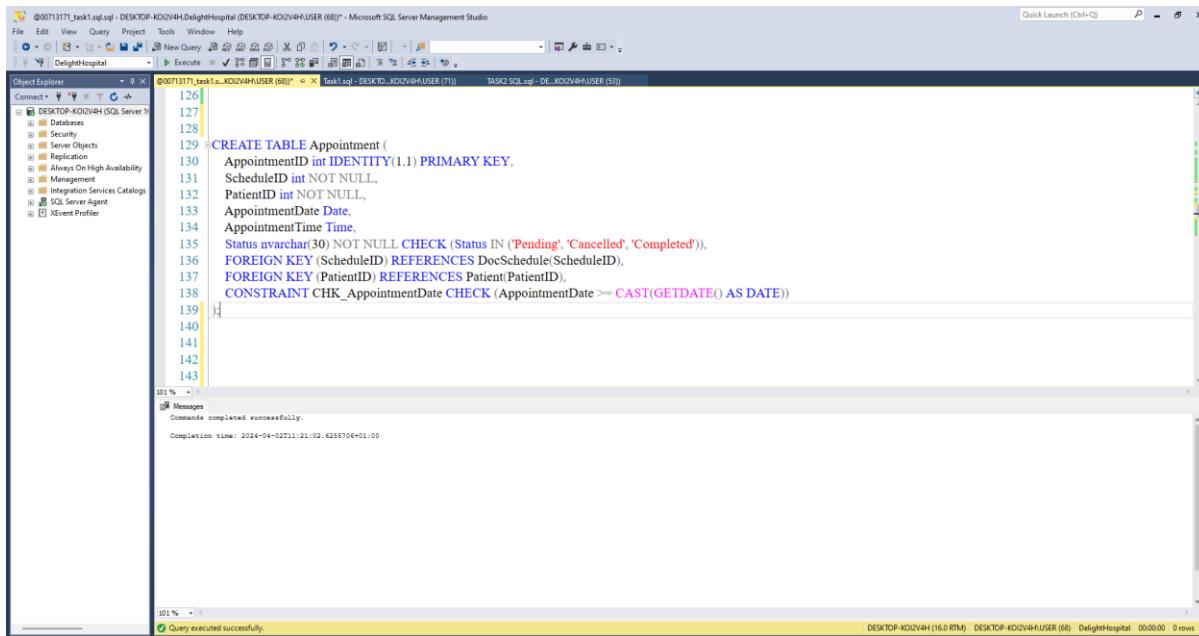
**ScheduleDate**, **ScheduleTime**, keeps the date and time record of the appointment.

Referential integrity between the DocSchedule and Doctor tables is ensured by the **FOREIGN KEY** constraint, which references the **DoctorID** column in the Doctor table.

**Not null** constraints in the DoctorID, AvailabilityStatus, tables guarantee that it is filled in at all times.

### Normalization Process for Appointment table:

There are no transitive relationships and every attribute is dependent on the primary key “AppointmentID” so the table is in 1NF, 2NF, and 3NF. Essential information including the appointment date, time, and status are stored in the Appointment table, which keeps track of patient-doctor appointments.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'DelightHospital' database is selected. In the center pane, a query window displays the SQL code for creating the 'Appointment' table. The code includes columns for AppointmentID (primary key, auto-increment), ScheduleID (not null), PatientID (not null), AppointmentDate (date type), AppointmentTime (time type), and Status (status type). It also includes foreign key constraints linking ScheduleID to DocSchedule(ScheduleID) and PatientID to Patient(PatientID). A check constraint ensures the appointment date is a valid date. The status column has a CHECK constraint that restricts values to 'Pending', 'Cancelled', or 'Completed'. The status column is defined as NOT NULL. The code spans from line 126 to 143. The status column is highlighted in red, indicating it is part of a constraint definition.

```
126 CREATE TABLE Appointment (
127     AppointmentID int IDENTITY(1,1) PRIMARY KEY,
128     ScheduleID int NOT NULL,
129     PatientID int NOT NULL,
130     AppointmentDate Date,
131     AppointmentTime Time,
132     Status nvarchar(30) NOT NULL CHECK (Status IN ('Pending', 'Cancelled', 'Completed')),
133     FOREIGN KEY (ScheduleID) REFERENCES DocSchedule(ScheduleID),
134     FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
135     CONSTRAINT CHK_AppointmentDate CHECK (AppointmentDate >= CAST(GETDATE() AS DATE))
136 )
137
138
139
140
141
142
143
```

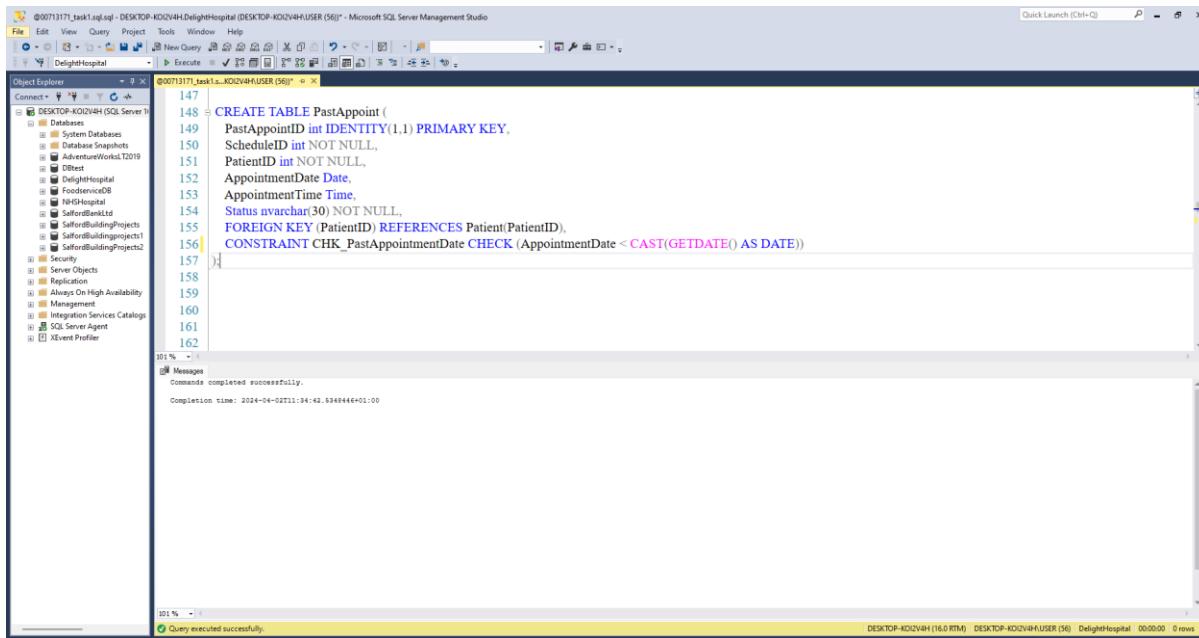
**Figure 1.1.8:** Justification

The **primary** key for Appointment table is AppointmentID, which is auto-incremented by IDENTITY(1,1), and uniquely identified for each record. The **ScheduleID** column is used to create a connection with the **DocSchedule** table

and represents the appointment's scheduled time slot. **PatientID** is used to identify the patient connected to their booked appointment. **AppointmentDate** records dates without time components, while **AppointmentTime** stores time separately from the date. The **status** column is set to nvarchar for descriptive representation. **Not null** constraints ensure that these columns are filled in at all times. **Foreign KEY** constraints ensure referral integrity between the Appointment table and related tables. The **CHECK** constraint guarantees the AppointmentDate is higher than or equal to the current date.

### Normalization Process for PastAppoint table:

There are no transitive relationships and every attribute is dependent on the primary key "PastAppointID" so the table is in 1NF, 2NF, and 3NF. Essential information including the **Past appointment** date, time, and **Completed** status are stored in the PastAppoint table, which keeps track of patient-doctor Past appointments.



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the 'DelightHospital' database and its objects like 'Patient', 'Appointment', and 'PastAppoint'. The central pane displays the T-SQL code for creating the 'PastAppoint' table. The code includes the table definition, primary key, foreign key constraint (linking to the 'Patient' table), and a CHECK constraint ensuring the appointment date is greater than or equal to the current date. The status bar at the bottom indicates the command was executed successfully.

```

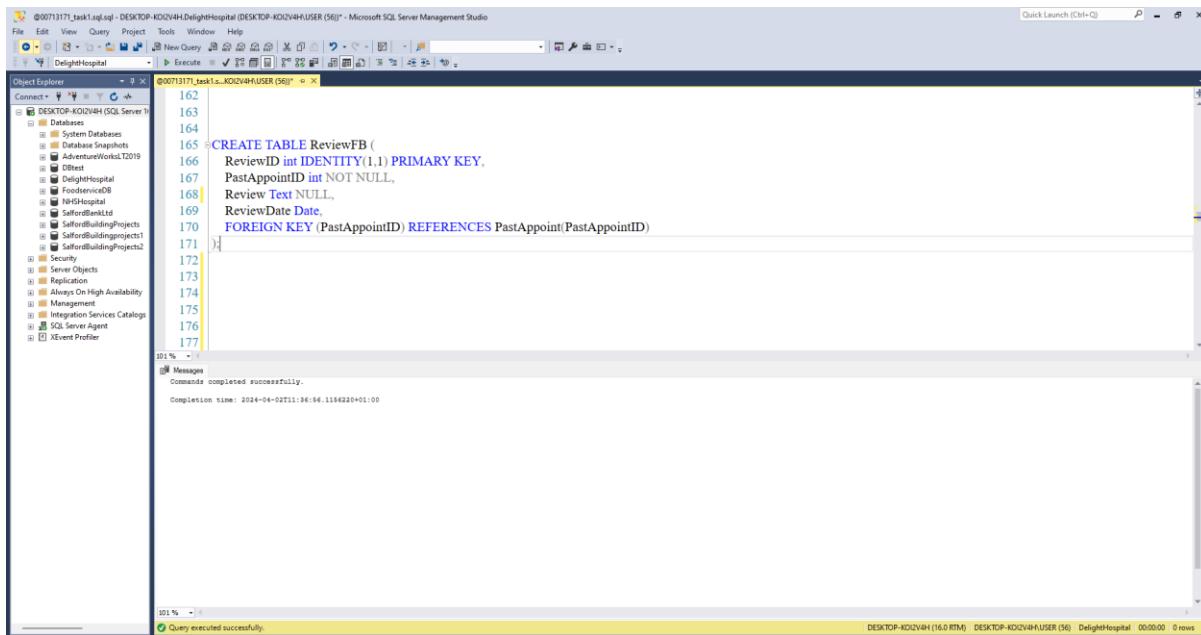
CREATE TABLE PastAppoint (
    PastAppointID int IDENTITY(1,1) PRIMARY KEY,
    ScheduleID int NOT NULL,
    PatientID int NOT NULL,
    AppointmentDate Date,
    AppointmentTime Time,
    Status nvarchar(30) NOT NULL,
    FOREIGN KEY(PatientID) REFERENCES Patient(PatientID),
    CONSTRAINT CHK_PastAppointmentDate CHECK (AppointmentDate < CAST(GETDATE() AS DATE))
)

```

**Figure 1.1.9:** Justification

**PastAppointID** is the **primary key** for each Past Appointment record, auto-increasing to identify each record uniquely. **ScheduleID** is used to store DocSchedule IDs in relation to other records moved from the Appointment table. **Patient ID** is used as a **foreign key** to establish a relationship with the Patient table. **AppointmentDate** records dates without time components, while **AppointmentTime** stores time separately from date. The **status** column is set to **nvarchar** for descriptive representation. **Not null** constraints ensure that these tables are filled in at all times. **Foreign KEY** constraint ensures referral integrity between the **PastAppoint** table and related tables, referencing the **PatientID** column in the Patient table. The **CHECK** constraint guarantees that the Past AppointmentDate is past, less than the current date.

## Normalization Process for ReviewID table:



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the 'DelightHospital' database and its objects like 'Patient', 'Appointment', 'PastAppoint', etc. The central pane displays a T-SQL script for creating the 'ReviewFB' table:

```

162
163
164
165 CREATE TABLE ReviewFB (
166     ReviewID int IDENTITY(1,1) PRIMARY KEY,
167     PastAppointID int NOT NULL,
168     Review Text NULL,
169     ReviewDate Date,
170     FOREIGN KEY (PastAppointID) REFERENCES PastAppoint(PastAppointID)
171 )
172
173
174
175
176
177

```

The status bar at the bottom indicates "Query executed successfully." and "Completion time: 2024-04-02T11:34:54.1184320+01:00".

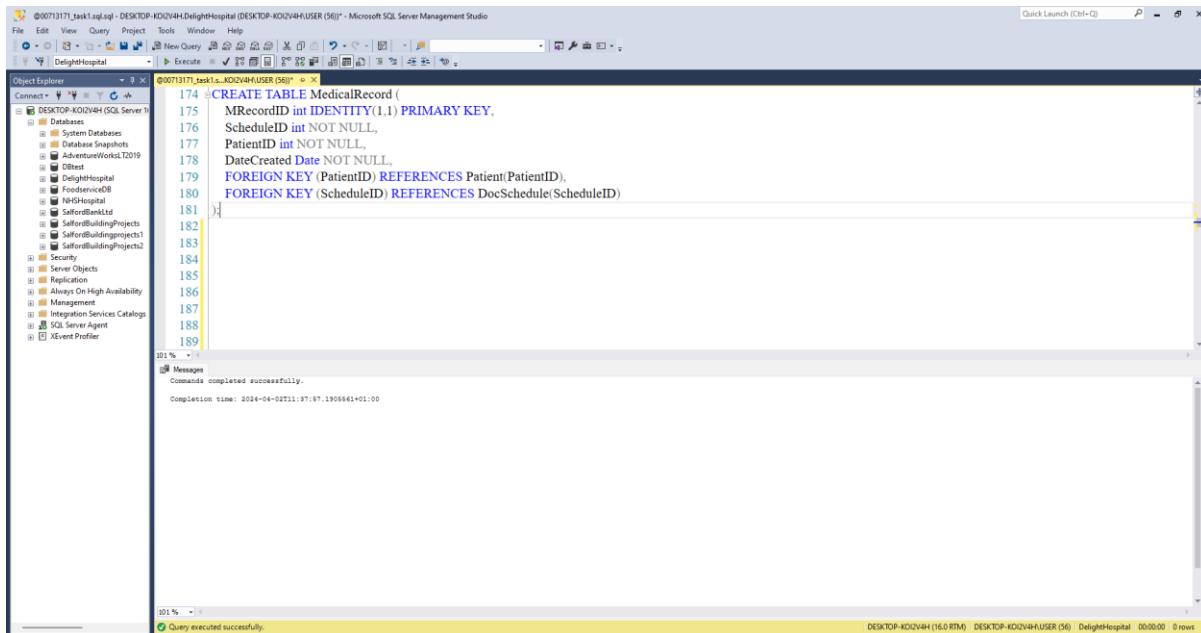
**Figure 1.2.0:** Justification

The table contains columns for review feedback input, previous appointment IDs, review text, review data, and references for the Foreign key(PastAppointID). The ReviewID column uniquely identifies each input and auto-increments using IDENTITY(1,1). The ReviewText column stores the textual substance of the comments, with “**TEXT**” being the data type. NULL values are allowed for non-review feedback situations. The ReviewDate column records the date of the review feedback. including ReviewID, PastAppointID, ReviewText, ReviewDate, and ReviewText, ReviewDate, and References for the Foreign key. Each input is uniquely identified using IDENTITY(1,1). The data type is TEXT, and NULL values are allowed for non-review feedback

This TSQL create a foreign key constraint, pastAppointID(PastAppoint), to ensure the PastAppointID in the ReviewFB table refers to a legitimate PastAppointID in the PastAppoint table.

## Normalization Process for MedicalRecord table:

There appears to be 1NF, 2NF, 3NF in the MedicalRecord table. There are no transitive or partial dependencies, and it contains atomic values. The MedicalRecord table houses medical information related to individual patient that have been uploaded after assessment.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'DelightHospital' is selected. In the center pane, a query window displays the T-SQL code for creating the 'MedicalRecord' table:

```
CREATE TABLE MedicalRecord (
    MRecordID int IDENTITY(1,1) PRIMARY KEY,
    ScheduleID int NOT NULL,
    PatientID int NOT NULL,
    DateCreated Date NOT NULL,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (ScheduleID) REFERENCES DocSchedule(ScheduleID)
)
```

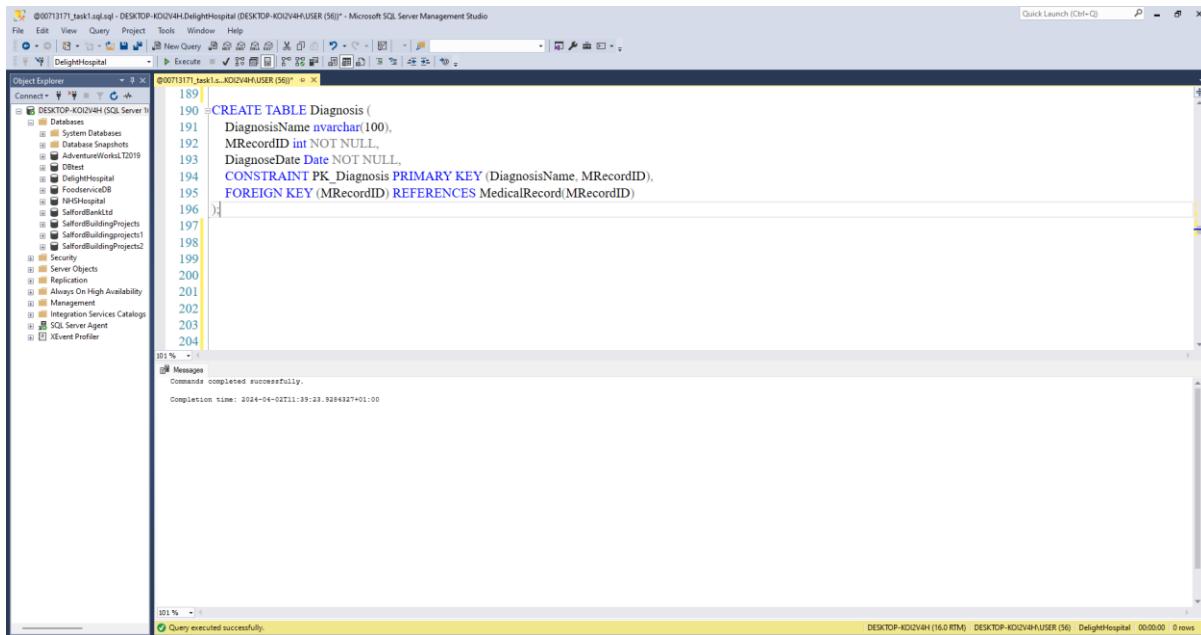
The status bar at the bottom indicates "Query executed successfully." and provides execution details: "Completion time: 2024-04-02T11:37:07.1908641+01:00".

**Figure 1.2.1:** Justification

The MedicalRecord is uniquely identified by its MRecordID column, which is an auto-incremented integer. The schedule linked to the medical record, which is linked to the DocSchedule table. The PatientID field indicates the patient linked to the medical record, and the DateCreated field indicates the health record's creation date.

## Normalization Process for Diagnosis table:

There appears to be 1NF, 2NF, 3NF in the Diagnosis table. There are no transitive or partial dependencies, and it contains atomic values. The Diagnosis table houses Diagnosed illnesses information related to individual patient that have been uploaded after assessment and attached to the medical record.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'DelightHospital' is selected. A query window titled '@00713171\_task1.sql' is open, displaying the following SQL code:

```
189 | CREATE TABLE Diagnosis (
190 |     DiagnosisName nvarchar(100),
191 |     MRecordID int NOT NULL,
192 |     DiagnoseDate Date NOT NULL,
193 |     CONSTRAINT PK_Diagnosis PRIMARY KEY (DiagnosisName, MRecordID),
194 |     FOREIGN KEY (MRecordID) REFERENCES MedicalRecord(MRecordID)
195 |
196 | )
197 |
198 |
199 |
200 |
201 |
202 |
203 |
204 |
```

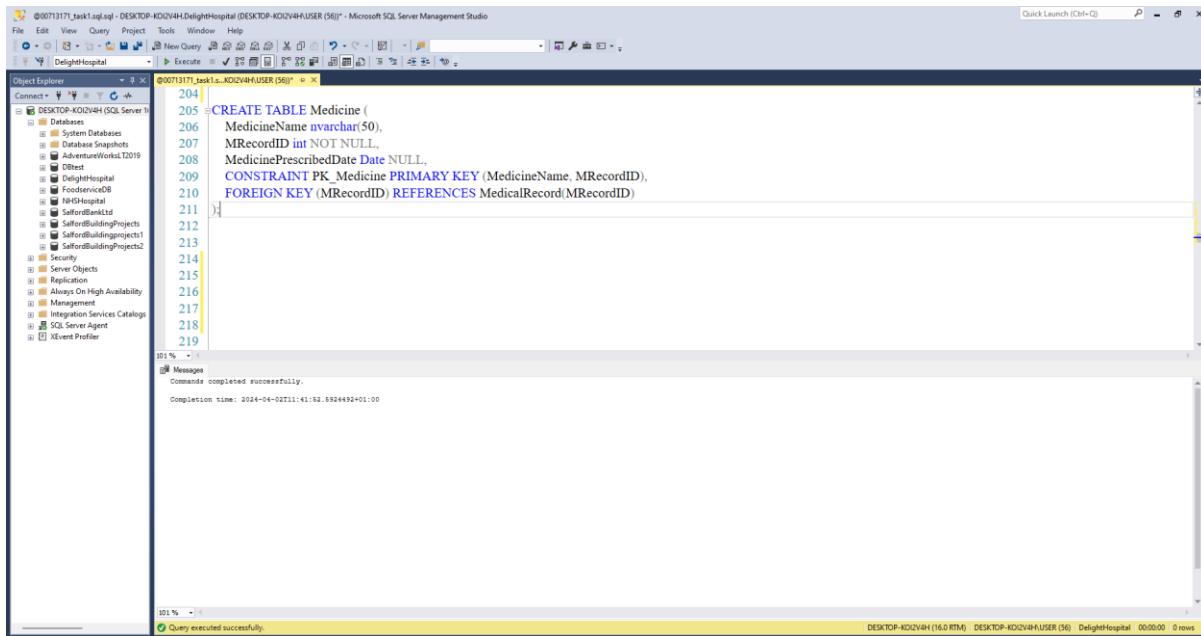
The status bar at the bottom indicates 'Query executed successfully.' and provides execution details: 'Completion time: 2024-04-04 22:11:39:23.9284327+01:00'. The title bar shows the full path: '@00713171\_task1.sql - DESKTOP-KOZV4H\DelightHospital (DESKTOP-KOZV4H\USER (56)) - Microsoft SQL Server Management Studio'.

**Figure 1.2.2:** Justification

The DiagnosisName column stores the diagnosis name in nvarchar(100) format. **DiagnoseDate:** The data type is Date, with the primary key being **DiagnosisName** and **MRecordID**, ensuring uniqueness for every combination of diagnosis name and medical recordID. The **FOREIGN KEY(MRecordID)** ensures referential integrity between the Diagnosis table and the MedicalRecord table, referencing the MRecordID column.

## Normalization Process for Medicine table:

There appears to be 1NF, 2NF, 3NF in the Medicine table. There are no transitive or partial dependencies, and it contains atomic values. The Medicine table houses Medicine medications administered to patient but attached to the medical records.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a connection to 'DESKTOP-KDZV4H' is selected, and the 'DelightHospital' database is expanded. A new query window titled '@00713171\_task1\_KDZV4H\USER (56)' is open, displaying the T-SQL code for creating the 'Medicine' table. The code includes constraints for the primary key (PK\_Medicine) and a foreign key (FK\_Medicine) referencing the 'MedicalRecord' table. The execution status at the bottom indicates 'Commands completed successfully.' and a completion time of '2024-04-02T11:41:52.5924492+01:00'.

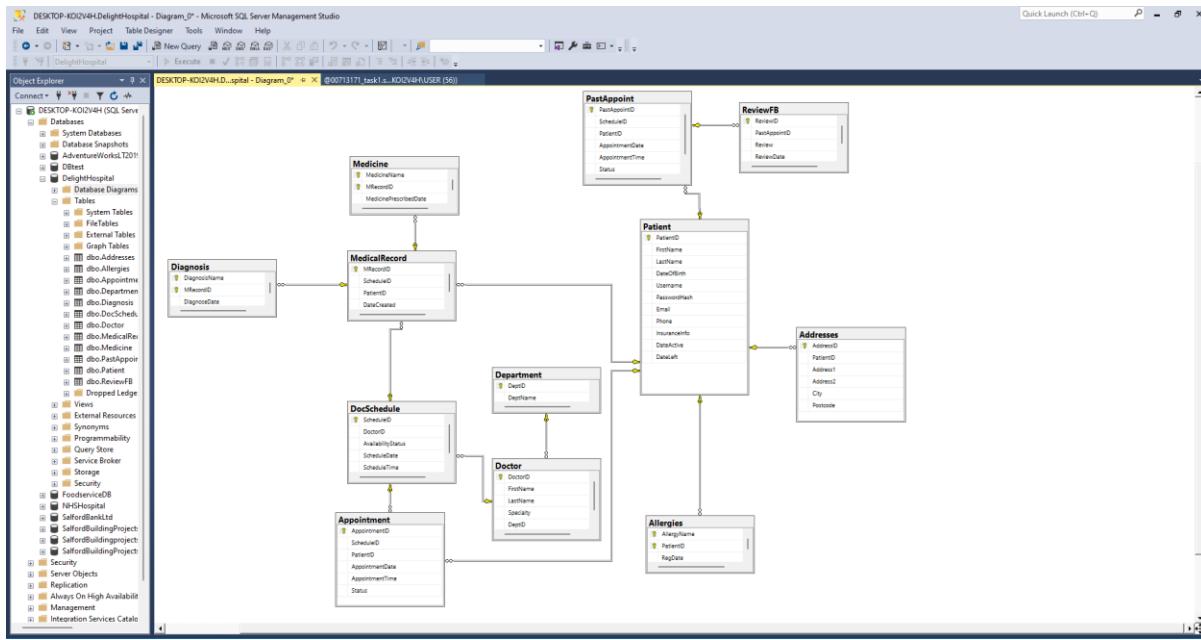
```
204 | CREATE TABLE Medicine (
205 |     MedicineName nvarchar(50),
206 |     MRecordID int NOT NULL,
207 |     MedicinePrescribedDate Date NULL,
208 |
209 |     CONSTRAINT PK_Medicine PRIMARY KEY (MedicineName, MRecordID),
210 |     FOREIGN KEY (MRecordID) REFERENCES MedicalRecord(MRecordID)
211 |
212 |
213 |
214 |
215 |
216 |
217 |
218 |
219
```

**Figure 1.2.3:** Justification

The table contains column for **Medication names**, **MRecordID**, and **Prescribed date**. The MedicineName column provides a variable length string using '**nvarchar**' for medication names, while the MRecordID column connects to the MedicalRecord table. The '**MedicinePrescribedDate**' column keeps dates that correspond to the medication's prescription date. The '**Composite**' primary key (MedicineName, MRecordID) ensures uniqueness, while the FOREIGN KEY (MRecordID) maintains referential integrity between the Medicine and MedicalRecord tables, finally **NOT NULL** is enforced on MRecordID and prescribed because they are very important data.

## Database Diagram

All tables contained in this database have been adequately joined with the required constraints such as foreign keys which has been used to reference or link together related tables so as to query accurate result. Below is the database diagram.



**Figure 1.2.4:** Showing the visual representation of each table and their relation.

## BELOW ARE THE STEPS TAKEN TO INSERT RECORDS INTO EACH TABLE CREATED ABOVE

### Patient inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'DelightHospital' is selected. In the center pane, a script window displays the following T-SQL code:

```

138 -- Procedure to populate patient table with encrypted secured password
139 CREATE PROCEDURE uspAddPatient
140     @FirstName nvarchar(30),
141     @LastName nvarchar(30),
142     @DateOfBirth date,
143     @Username nvarchar(40),
144     @PasswordHash nvarchar(50),
145     @Email nvarchar(50),
146     @Phone nvarchar(20),
147     @InsuranceInfo nvarchar(100),
148     @DateActive date,
149     @DateLeft date
150 AS
151 DECLARE @Salt UNIQUEIDENTIFIER=NEWID()
152
153 INSERT INTO Patient(FirstName, LastName, DateOfBirth, Username,
154 PasswordHash, Salt, Email, Phone, InsuranceInfo, DateActive, DateLeft)
155 VALUES( @FirstName, @LastName, @DateOfBirth, @Username, HASHBYTES('SHA2_512',
156 @PasswordHash+CAST(@Salt AS nvarchar(36))), @Salt,
157 @Email, @Phone, @InsuranceInfo, @DateActive, @DateLeft);
158 --The HASHBYTES function in SQL Server returns the hash of the input. The first
159 --argument specifies the algorithm to be used:
160

```

In the bottom pane, the results of the execution show:

```

Messages
Commands completed successfully.

Completion time: 2024-04-03T00:18:03.4034519+01:00

```

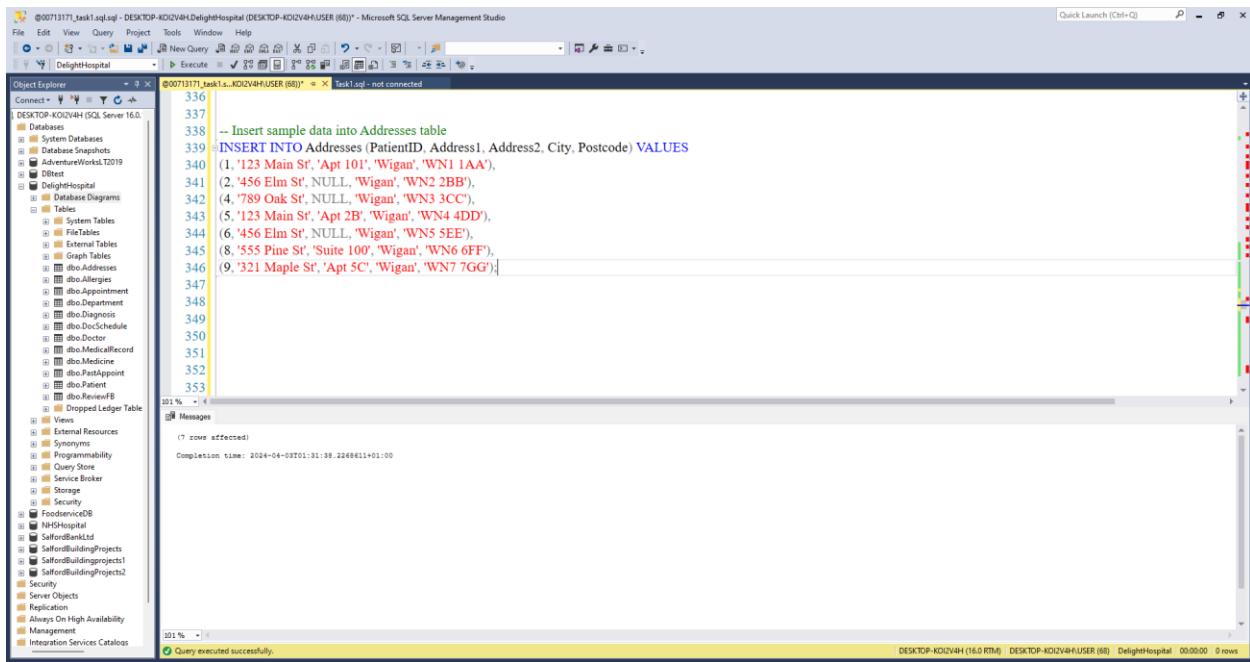
**Figure 1.2.5:** On my patient table there is need to secure patients' passwords and for this reason I did not use the regular INSERT statement to populate the table. I have used the **binary datatype** to store **password hash**, to make it harder for password to be cracked I have also introduced the **salted hash**. **Figure 1.2.7** shows the same list of columns in my patient table. The hash of the input is returned by SQL Server's HASHBYTES function. The algorithm to be applied is specified in the first argument.

**Figure 1.2.6:** The EXECUTE statements executes the stored procedure in **Figure 1.2.5**, despite two of the five successfully inserted records not running initially

**Figure 1.2.7:** The screen short above shows the inserted records into the patient

table with the Password hashed successfully and the unique Salt column simultaneously generated accordingly.

## Address inserted record:



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-KD1ZV4H'. The 'Tables' node under 'SafordBuildingProjects' is expanded, showing 'Addresses' and other tables like 'Patient', 'Appointment', and 'ReviewFB'. The 'Addresses' table is selected. The 'Script' tab in the top ribbon is active, displaying a T-SQL script. The script starts with a comment '-- Insert sample data into Addresses table' and then uses an 'INSERT INTO' statement to insert seven records into the 'Addresses' table. The 'Messages' pane at the bottom shows '0 rows affected' and a completion time of '2024-04-03T01:31:38.2248611+01:00'. The status bar at the bottom right indicates 'Query executed successfully'.

```
-- Insert sample data into Addresses table
INSERT INTO Addresses (PatientID, Address1, Address2, City, Postcode) VALUES
(1, '123 Main St', 'Apt 101', 'Wigan', 'WN1 1AA'),
(2, '456 Elm St', NULL, 'Wigan', 'WN2 2BB'),
(4, '789 Oak St', NULL, 'Wigan', 'WN3 3CC'),
(5, '123 Main St', 'Apt 2B', 'Wigan', 'WN4 4DD'),
(6, '456 Elm St', NULL, 'Wigan', 'WN5 5EE'),
(8, '555 Pine St', 'Suite 100', 'Wigan', 'WN6 6FF'),
(9, '321 Maple St', 'Apt 5C', 'Wigan', 'WN7 7GG');
```

**Figure 1.2.8:** This screen short shows my written Insert TSQL query for Address table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. A query window titled '000713171\_task1.sql.sql - DESKTOP-KOIV4H.DeslightHospital (DESKTOP-KOIV4H\USER (68)) - Microsoft SQL Server Management Studio' contains the following T-SQL code:

```

313
314
315
316
317
318 Select * From Addresses;
319
320
321
322
323
324
325
326
327
328
329
330

```

The results pane shows the following data from the Addresses table:

	AddressID	PatientID	Address1	Address2	City	Postcode
1	1	1	123 Main St	Apt 101	Wigan	WN1 1AA
2	2	2	456 Elm St	NULL	Wigan	WN2 2B8
3	3	4	789 Oak St	NULL	Wigan	WN2 3CC
4	4	5	123 Main St	Apt 2B	Wigan	WN4 4DD
5	5	6	456 Elm St	NULL	Wigan	WN8 5EE
6	6	8	555 Pine St	Suite 100	Wigan	WN8 6FF
7	7	9	321 Maple St	Apt 9C	Wigan	WN7 7GG

At the bottom, a message indicates 'Query executed successfully.'

**Figure 1.2.9:** Here is the result for address

### Allergies inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. A query window titled '000713171\_task1.sql.sql - DESKTOP-KOIV4H.DeslightHospital (DESKTOP-KOIV4H\USER (68)) - Microsoft SQL Server Management Studio' contains the following T-SQL code:

```

424
425 -- Insert statements for Allergies table
426 INSERT INTO Allergies (AllergyName, PatientID, RegDate) VALUES
427 ('Peanuts', 1, '2024-06-02'),
428 ('Penicillin', 2, '2024-06-03'),
429 ('Dust', 4, '2024-06-04'),
430 ('Cat Hair', 1, '2024-06-02'),
431 ('Shellfish', 2, '2024-06-03'),
432 ('Mold', 4, '2024-06-04'),
433 ('Pollen', 1, '2024-06-02'),
434 ('Eggs', 5, '2024-06-05'),
435 ('Milk', 6, '2024-06-06'),
436 ('Antibiotic', 8, '2024-06-07'),
437 ('Nickel', 9, '2024-06-08')
438
439
440
441

```

The results pane shows the following message:

(11 rows affected)  
Completion time: 2024-04-03T01:17:30.8346840+01:00

At the bottom, a message indicates 'Query executed successfully.'

**Figure 1.3.0:** This screen short shows my written Insert TSQL query for Allergies table and was returned successful with eleven (11) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the DelightHospital database which contains tables like Allergies, Doctors, and Patients. The main window displays a query results grid titled 'Results'.

```

313
314
315
316
317
318 Select * From Allergies;
319
320
321
322
323
324
325
326
327
328
329
330

```

AllergyName	PatentID	RegDate
Ambiotic	8	2024-06-07
Cat Hair	1	2024-06-02
Dust	4	2024-06-04
Eggs	5	2024-06-05
Latex	6	2024-06-06
Mold	4	2024-06-04
Nickel	9	2024-06-08
Peanuts	1	2024-06-02
Pencillin	2	2024-06-03
Pollen	1	2024-06-02
Shellfish	2	2024-06-03

Query executed successfully.

**Figure 1.3.1:** Here is the result for Allergies

### Department inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the DelightHospital database which contains tables like Departments, Doctors, and Patients. The main window displays a query results grid titled 'Messages'.

```

450
451
452 - Inserting data into the Departments table
453 =INSERT INTO Department (DeptName)
454 VALUES
455 ('Cardiology'),
456 ('Pediatrics'),
457 ('Orthopedics'),
458 ('Oncology'),
459 ('Neurology'),
460 ('Gastroenterology'),
461 ('Psychiatrist')
462
463
464
465
466
467

```

(7 rows affected)

Completion time: 2024-04-03T02:08:49.0897859+01:00

Query executed successfully.

**Figure 1.3.2:** This screen short shows my written Insert TSQL query for Department table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including 'DelightHospital'. The 'Tables' node under 'DelightHospital' is expanded, showing various tables like 'Department', 'Employee', 'Patient', etc. A query window titled 'Task1.sql - not connected' contains the following SQL code:

```

313
314
315
316
317
318 Select * From Department;
319
320
321
322
323
324
325
326
327
328
329
330

```

The results pane displays the data from the 'Department' table:

DeptID	DeptName
1	Administracy
2	Pediatrics
3	Orthopedics
4	Oncology
5	Neurology
6	Gastroenterology
7	Psychiatry

At the bottom, a message indicates: 'Query executed successfully.'

**Figure 1.3.3: Here is the result for Department**

### Doctor inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists databases, including 'DelightHospital'. A query window titled 'Task1.sql - not connected' contains the following SQL code:

```

215
216
217 -- Insert sample data into Doctor table
218 INSERT INTO Doctor (FirstName, LastName, Specialty, DeptID) VALUES
219 ('John', 'Doe', 'Cardiologist', 1),
220 ('Jane', 'Smith', 'Pediatrician', 2),
221 ('Michael', 'Johnson', 'Orthopedic Surgeon', 3),
222 ('Emily', 'Brown', 'Oncologist', 4),
223 ('David', 'Lee', 'Neurologist', 5),
224 ('James', 'Smith', 'Gastroenterologist', 6),
225 ('Clinton', 'Obama', 'Psychiatrist', 7)
226
227
228
229
230
231
232
233

```

The results pane shows the message: '(7 rows affected)' and 'Completion time: 2024-04-03T02:20:49.3430765+01:00'.

At the bottom, a message indicates: 'Query executed successfully.'

**Figure 1.3.4:** This screen short shows my written Insert TSQL query for Doctor table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The query window contains the following T-SQL code:

```

309
310
311
312 Select * From Doctor;
313
314
315
316
317
318
319
320
321
322
323
324
325
326

```

The results pane displays a table with 7 rows of data:

	DoctorID	FirstName	LastName	Specialty	DeptID
1	1	John	Doe	Cardiologist	1
2	2	Jane	Smith	Pediatrician	2
3	3	Michael	Johnson	Orthopedic Surgeon	3
4	4	Emily	Brown	Oncologist	4
5	5	David	Lee	Neurologist	5
6	6	James	Smith	Gastroenterologist	6
7	7	Clinton	Obama	Psychiatrist	7

At the bottom of the results pane, it says "Query executed successfully."

**Figure 1.3.5:** Here is the result for Doctors

### DocSchedule inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The query window contains the following T-SQL code:

```

225 -- Insert sample data into DocSchedule table
226 INSERT INTO DocSchedule (DoctorID, AvailabilityStatus, ScheduleDate, ScheduleTime) VALUES
227 (1, 'Available', '2024-06-02', '08:30'),
228 (2, 'Available', '2024-06-03', '09:50'),
229 (3, 'Available', '2024-06-04', '10:30'),
230 (4, 'Available', '2024-06-05', '10:00'),
231 (5, 'Available', '2024-06-06', '06:30'),
232 (6, 'Available', '2024-06-07', '11:30'),
233 (7, 'Available', '2024-06-08', '12:30');
234
235
236
237
238
239
240
241
242

```

The results pane shows the message: "(7 rows affected)" and "Completion time: 2024-04-03T02:49:13.2046623+01:00".

At the bottom of the results pane, it says "Query executed successfully."

**Figure 1.3.6:** This screen short shows my written Insert TSQL query for DocSchedule table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. The main window contains a query editor with the following T-SQL code:

```
Select * From DocSchedule;
```

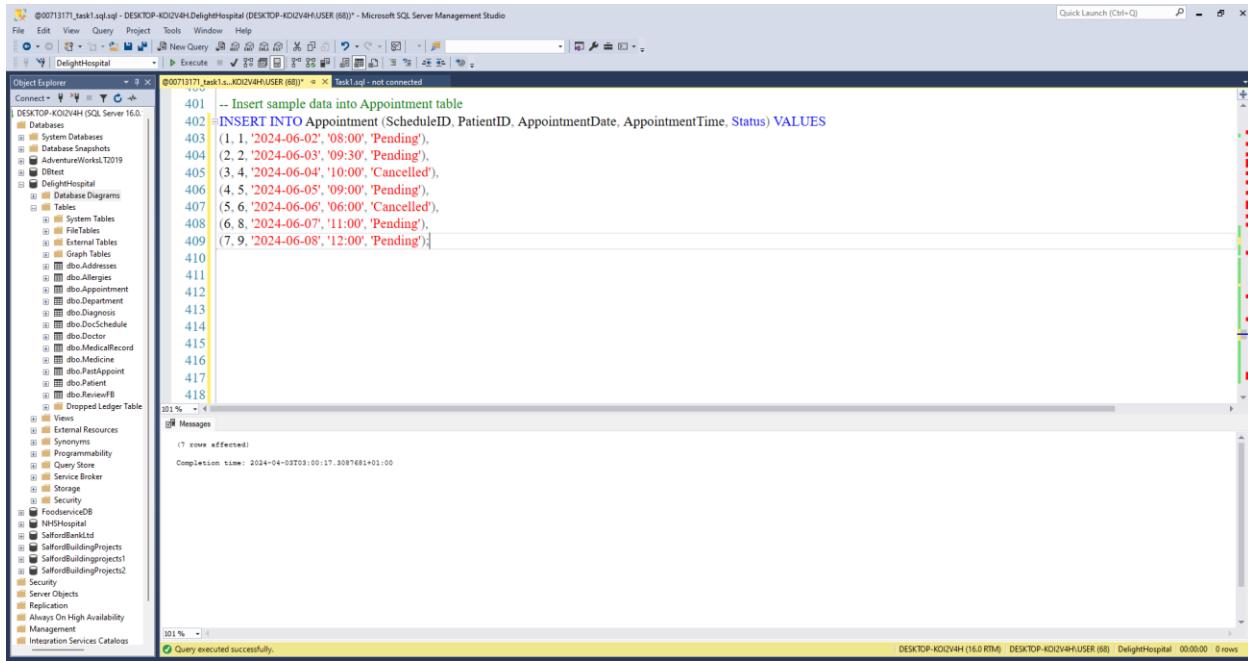
Below the code, the results pane displays a table with 7 rows of data:

ScheduleID	DoctorID	AvailabilityStatus	ScheduleDate	ScheduleTime
1	1	Available	2024-06-03	08:30:00.0000000
2	2	Available	2024-06-03	09:30:00.0000000
3	3	Available	2024-06-04	10:30:00.0000000
4	4	Available	2024-06-05	10:00:00.0000000
5	5	Available	2024-06-06	06:30:00.0000000
6	6	Available	2024-06-07	11:30:00.0000000
7	7	Available	2024-06-08	12:30:00.0000000

At the bottom of the results pane, it says "Query executed successfully."

**Figure 1.3.7:** Here is the result for DocSchedule

## Appointment inserted record:



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the DelightHospital database. The central pane displays a T-SQL script named 'Task1.sql' with the following content:

```

401 -- Insert sample data into Appointment table
402 INSERT INTO Appointment (ScheduleID, PatientID, AppointmentDate, AppointmentTime, Status) VALUES
403 (1, 1, '2024-06-02', '08:00', 'Pending'),
404 (2, 2, '2024-06-03', '09:30', 'Pending'),
405 (3, 4, '2024-06-04', '10:00', 'Cancelled'),
406 (4, 5, '2024-06-05', '09:00', 'Pending'),
407 (5, 6, '2024-06-06', '06:00', 'Cancelled'),
408 (6, 8, '2024-06-07', '11:00', 'Pending'),
409 (7, 9, '2024-06-08', '12:00', 'Pending')
410
411
412
413
414
415
416
417
418

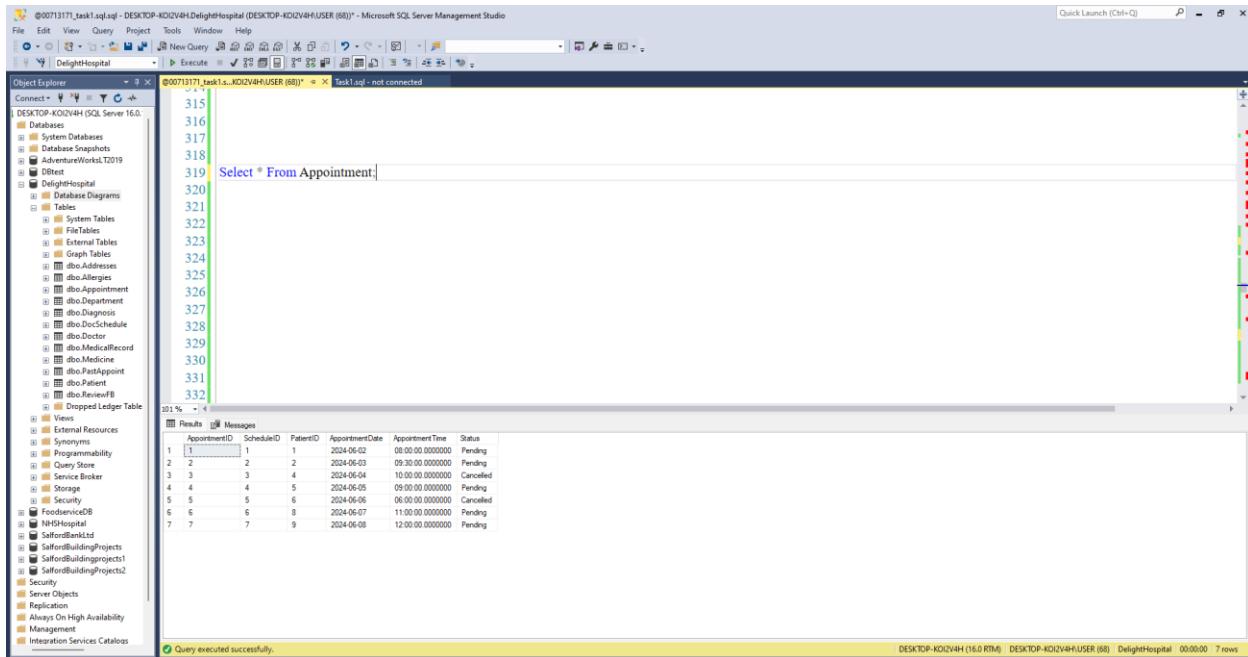
```

Below the script, the message pane shows:

- (7 rows affected)
- Completion time: 2024-04-03T03:00:17.3087681+01:00

The status bar at the bottom indicates: Query executed successfully.

**Figure 1.3.8:** This screen short shows my written Insert TSQL query for Appointment table and was returned successful with seven (7) record.



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the DelightHospital database. The central pane displays a T-SQL script named 'Task1.sql' with the following content:

```

315
316
317
318
319 Select * From Appointment;
320
321
322
323
324
325
326
327
328
329
330
331
332

```

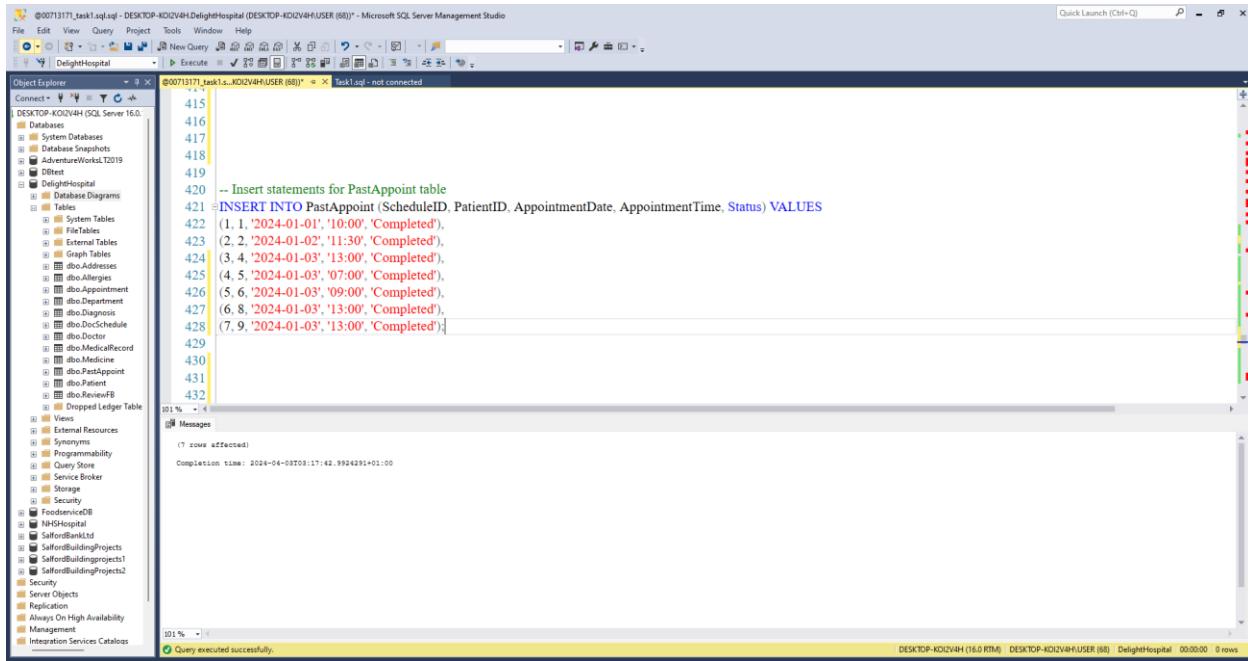
Below the script, the results pane shows the data from the Appointment table:

	ScheduleID	PatientID	AppointmentDate	AppointmentTime	Status
1	1	1	2024-06-02	08:00:00.000000	Pending
2	2	2	2024-06-03	09:30:00.000000	Pending
3	3	3	2024-06-04	10:00:00.000000	Cancelled
4	4	4	2024-06-05	09:00:00.000000	Pending
5	5	5	2024-06-06	06:00:00.000000	Cancelled
6	6	6	2024-06-07	11:00:00.000000	Pending
7	7	9	2024-06-08	12:00:00.000000	Pending

The status bar at the bottom indicates: Query executed successfully.

**Figure 1.3.9:** Here is the result for Appointment

## PastAppoint inserted record:



```

-- Insert statements for PastAppoint table
INSERT INTO PastAppoint (ScheduleID, PatientID, AppointmentDate, AppointmentTime, Status) VALUES
(1, 1, '2024-01-01', '10:00', 'Completed'),
(2, 2, '2024-01-02', '11:30', 'Completed'),
(3, 4, '2024-01-03', '13:00', 'Completed'),
(4, 5, '2024-01-03', '07:00', 'Completed'),
(5, 6, '2024-01-03', '09:00', 'Completed'),
(6, 8, '2024-01-03', '13:00', 'Completed'),
(7, 9, '2024-01-03', '13:00', 'Completed')

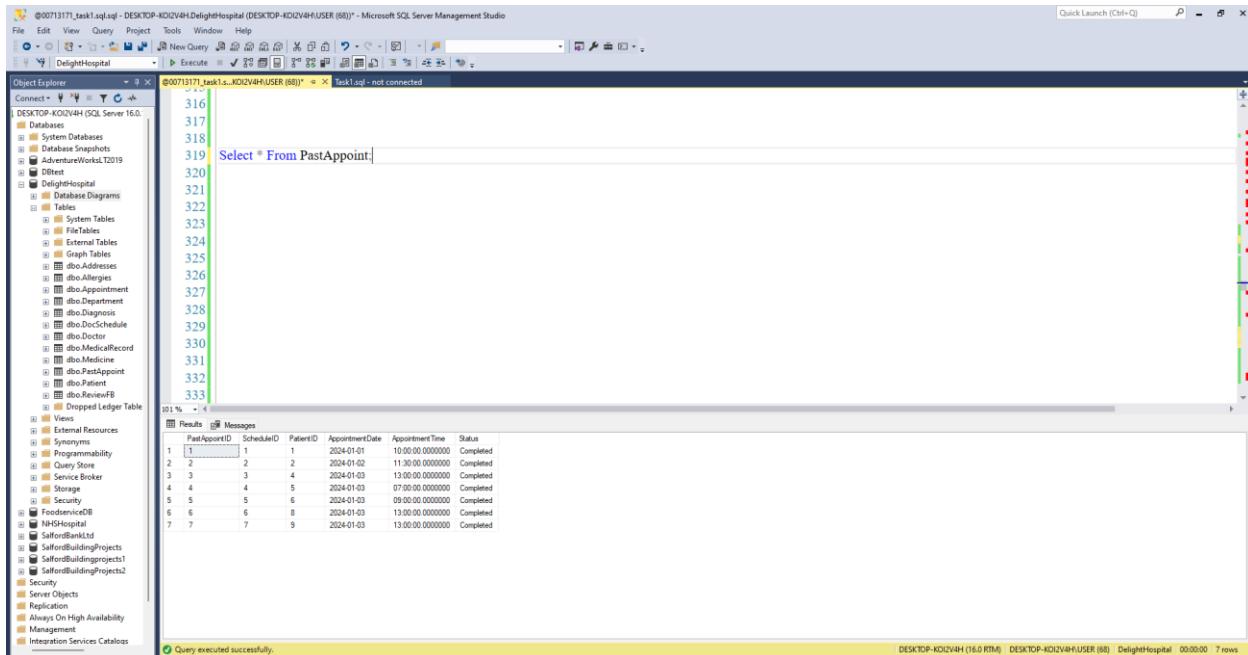
```

(7 rows affected)

Completion time: 2024-04-03T03:17:42.9924291+01:00

Query executed successfully.

**Figure 1.4.0:** This screen short shows my written Insert TSQL query for PastAppoint table and was returned successful with seven (7) record.



```

Select * From PastAppoint;

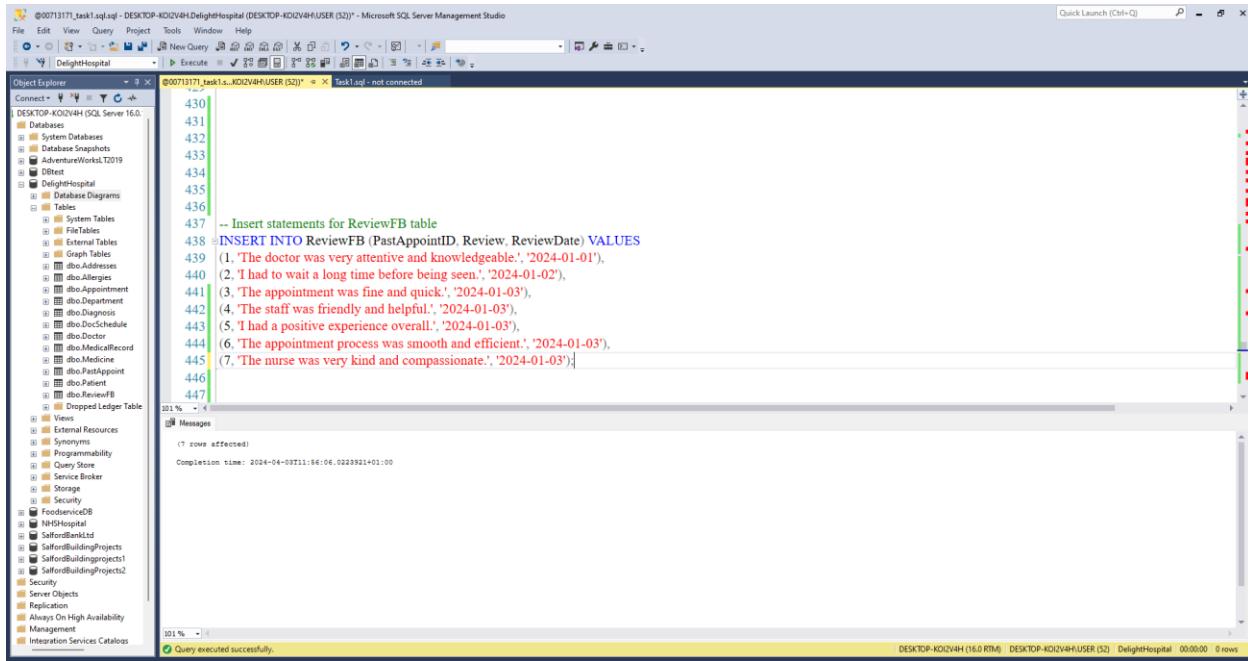
```

	PastAppointID	ScheduleID	PatientID	AppointmentDate	AppointmentTime	Status
1	1	1	1	2024-01-01	10:00:00.0000000	Completed
2	2	2	2	2024-01-02	11:30:00.0000000	Completed
3	3	3	4	2024-01-03	13:00:00.0000000	Completed
4	4	4	5	2024-01-03	07:00:00.0000000	Completed
5	5	5	6	2024-01-03	09:00:00.0000000	Completed
6	6	6	8	2024-01-03	13:00:00.0000000	Completed
7	7	7	9	2024-01-03	13:00:00.0000000	Completed

Query executed successfully.

**Figure 1.4.1:** Here is the result for address

## ReviewFB inserted record:



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including the DelightHospital database and its tables like ReviewFB, PastAppointment, and Patient. The central pane displays a T-SQL script named 'Task1.sql' with the following content:

```
-- Insert statements for ReviewFB table
INSERT INTO ReviewFB (PastAppointID, Review, ReviewDate) VALUES
(1, 'The doctor was very attentive and knowledgeable.', '2024-01-01'),
(2, 'I had to wait a long time before being seen.', '2024-01-02'),
(3, 'The appointment was fine and quick!', '2024-01-03'),
(4, 'The staff was friendly and helpful!', '2024-01-03'),
(5, 'I had a positive experience overall.', '2024-01-03'),
(6, 'The appointment process was smooth and efficient.', '2024-01-03'),
(7, 'The nurse was very kind and compassionate.', '2024-01-03');
```

The status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.4.2:** This screen short shows my written Insert TSQL query for **ReviewFB** table (review feedback record) and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. The main pane displays a T-SQL query:

```
316
317
318
319 Select * From ReviewFB;
```

The results grid shows the following data:

ReviewID	PatientID	Review	ReviewDate
1	1	The doctor was very attentive and knowledgeable.	2024-01-01
2	2	I had to wait a long time before being seen.	2024-01-02
3	3	The appointment was fine and quick.	2024-01-03
4	4	The staff was friendly and helpful.	2024-01-03
5	5	I had a positive experience overall.	2024-01-03
6	6	The appointment process was smooth and efficient.	2024-01-03
7	7	The nurse was very kind and compassionate.	2024-01-03

At the bottom, a message indicates "Query executed successfully."

**Figure 1.4.3:** Here is the result for address

### MedicalRecord inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. The main pane displays a T-SQL query:

```
349
350
351
352
353
354
355 -- Insert sample data into MedicalRecord table
356 INSERT INTO MedicalRecord (ScheduleID, PatientID, DateCreated) VALUES
357 (1, 1, '2024-01-01'),
358 (2, 2, '2024-02-01'),
359 (3, 4, '2024-03-05'),
360 (4, 5, '2024-03-28'),
361 (5, 6, '2024-03-29'),
362 (6, 8, '2024-02-29'),
363 (7, 9, '2024-03-01');
```

The results grid shows the message "(7 rows affected)" and "Completion time: 2024-04-03T12:10:08.3091258+01:00".

At the bottom, a message indicates "Query executed successfully."

**Figure 1.4.4:** This screen short shows my written Insert TSQL query for **MedicalRecord** table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including DESKTOP-KOIV4H, AdventureWorksLT2019, DBTest, and DelightHospital. The DelightHospital database is selected. In the center, a query window titled '000713171\_task1.s...KOIV4H\USER (S2)' displays the following T-SQL code:

```

316
317
318
319 Select * From MedicalRecord
320
321
322
323
324
325
326
327
328
329
330
331
332
333

```

Below the code, the results pane shows a table with the following data:

	MRecordID	SchedulerID	PatientID	DateCreated
1	1	1	1	2024-03-01
2	2	2	2	2024-03-01
3	3	3	4	2024-03-05
4	4	4	5	2024-03-28
5	5	5	6	2024-03-29
6	6	6	8	2024-02-29
7	7	7	9	2024-03-01

The status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.4.5:** Here is the result for MedicalRecord

## Diagnosis inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including DESKTOP-KOIV4H, AdventureWorksLT2019, DBTest, and DelightHospital. The DelightHospital database is selected. In the center, a query window titled '000713171\_task1.s...KOIV4H\USER (S2)' displays the following T-SQL code:

```

367 -- Insert sample data into Diagnosis table
368 INSERT INTO Diagnosis (DiagnosisName, MRecordID, DiagnoseDate) VALUES
369 ('Hypertension', 1, '2024-01-02'),
370 ('Asthma', 2, '2024-02-03'),
371 ('Fractures', 3, '2024-03-06'),
372 ('Cancer', 4, '2024-03-29'), -- cancer
373 ('Migraine', 5, '2024-03-30'),
374 ('Liver Disease', 6, '2024-02-29'),
375 ('Bipolar Disorder', 7, '2024-03-05');
376
377
378
379
380
381
382
383
384

```

Below the code, the results pane shows the message: '(7 rows affected)' and 'Completion time: 2024-04-03T12:23:39.9819342+01:00'.

The status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.4.6:** This screen short shows my written Insert TSQL query for Diagnosis table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. The main pane displays a T-SQL query and its results. The query is:

```
316
317
318
319 Select * From Diagnosis
320
321
322
323
324
325
326
327
328
329
330
331
332
333
```

The results pane shows the following data:

	DiagnoseName	MRecordID	DiagnoseDate
1	Asthma	2	2024-02-03
2	Cardio disorder	3	2024-03-05
3	Cancer	4	2024-03-29
4	Fractures	3	2024-03-06
5	Hypertension	1	2024-01-02
6	Liver Disease	6	2024-02-29
7	Migraine	5	2024-03-30

At the bottom, it says "Query executed successfully."

**Figure 1.4.7:** Here is the result for Diagnosis

### Medicine inserted record:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. The main pane displays a T-SQL query and its results. The query is:

```
378
379
380 -- Insert data into Medicine table
381 INSERT INTO Medicine (MedicineName, MRecordID, MedicinePrescribedDate) VALUES
382 ('Lisinopril', 1, '2024-01-02'),
383 ('Hydrochlorothiazide', 1, '2024-01-02'),
384 ('Fluticasone', 2, '2024-02-03'),
385 ('Ibuprofen', 3, '2024-03-06'),
386 ('Chemotherapy', 4, '2024-03-29'),
387 ('Painkillers', 4, '2024-03-29'),
388 ('Sumatriptan', 5, '2024-03-30'),
389 ('Ursodeoxycholic Acid (UDCA)', 6, '2024-02-29'),
390 ('Lamotrigine', 7, '2024-03-05'),
391 ('Carbamazepine', 7, '2024-03-05')
```

The results pane shows the message "(10 rows affected)" and "Completion time: 2024-04-03T12:42:49.9974546+01:00".

At the bottom, it says "Query executed successfully."

**Figure 1.4.8:** This screen short shows my written Insert TSQL query for Medicine table and was returned successful with seven (7) record.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists various databases and objects. A query window titled 'task1...KOI2V4H.USER (S2)' contains the following T-SQL code:

```
315
316
317
318
319 Select * From Medicine;
320
321
322
323
324
325
326
327
328
329
330
331
332
```

The results pane shows a table with 10 rows of data:

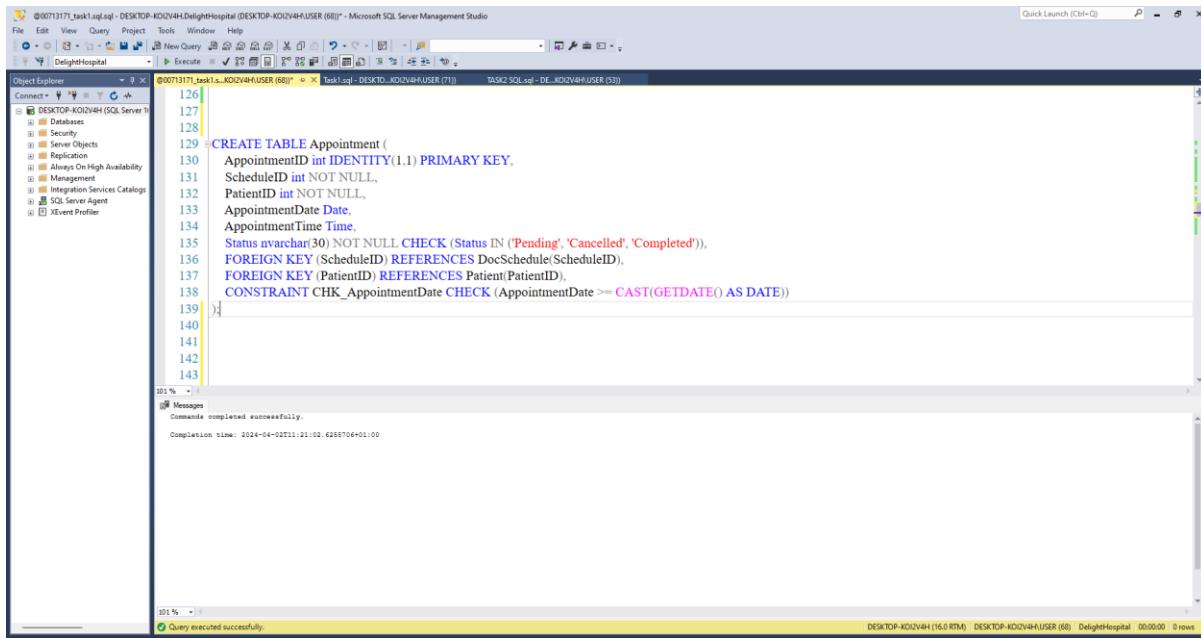
MedicineName	MRecordID	MedicinePrescribedDate
Carbamazepine	7	2024-03-05
Diazepam	4	2024-02-28
Uticosine	2	2024-02-03
Hydrochlorothiazide	1	2024-01-02
Ibuprofen	3	2024-03-06
Lamotrigine	7	2024-03-05
Liaopill	1	2024-01-02
Parkliles	4	2024-03-29
Sumatriptan	5	2024-03-30
Undesoxcholic Acid (UDCA)	6	2024-02-29

A status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.4.9:** Here is the result for Medicine

## PART 2:

Q2. Adding constraint to check the appointment date is not in the past:



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'DESKTOP-KOZV4H\KoziV4H' database is selected. A new query window titled 'Task1.sql - DESKTOP-KOZV4H\KoziV4H\USER (68)' is open, displaying the following T-SQL code:

```
CREATE TABLE Appointment (
    AppointmentID int IDENTITY(1,1) PRIMARY KEY,
    ScheduleID int NOT NULL,
    PatientID int NOT NULL,
    AppointmentDate Date,
    AppointmentTime Time,
    Status nvarchar(30) NOT NULL CHECK (Status IN ('Pending', 'Cancelled', 'Completed')),
    FOREIGN KEY (ScheduleID) REFERENCES DocSchedule(ScheduleID),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    CONSTRAINT CHK_AppointmentDate CHECK (AppointmentDate >= CAST(GETDATE() AS DATE))
)
```

The code is numbered from 126 to 143. The 'Messages' pane at the bottom shows the command completed successfully with a completion time of 2024-04-02T11:21:02.6288706+01:00.

**Figure 1.5.0: CHECK constraint:** The AppointmentDate is guaranteed to be present (i.e. larger than or equal to the current date) by the CHECK constraint **CHK\_AppointmentDate**.

### Q3.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including DelightHospital, which is currently selected. The script pane at the top contains a TSQL query:

```
--No 3 List all the patients who older than 40 and have Cancer in diagnosis.
SELECT P.PatientID, P.FirstName, P.LastName, DateOfBirth
FROM Patient P
JOIN MedicalRecord M ON P.PatientID = M.PatientID
JOIN Diagnosis D ON M.MRecordID = D.MRecordID
WHERE DATEDIFF(YEAR, P.DateOfBirth, GETDATE()) > 40
AND D.DiagnosisName LIKE '%Cancer%'
```

The results pane below shows the output of the query:

PatientID	FirstName	LastName
5	Sarah	Johnson

A status bar at the bottom right indicates "Query executed successfully."

**Figure 1.5.1:** This TSQL query above answers question two (3) by using select statement to list some patient details including names and date of birth from patient table aliased with “P” and using “JOIN” to join MedicalRecord based on PatientID in other to join Diagnosis table using MRecordID and finally use the WHERE clause to filter the rows needed. Result from the executed query is derived by getting difference between system date using “GETDATE()” and “DateOfBirth” but with additional condition using LIKE “%Cancer%” to get patient who also have Cancer as string.

## Q4a.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'DelightHospital' is selected. In the center pane, a script window titled '000713171\_Task1.sql - not connected' contains the following T-SQL code:

```
--4a 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.  
  
369 =CREATE PROCEDURE SearchMedicineName  
    (@MedicineName NVARCHAR(100))  
370     AS  
372 =BEGIN  
373     SELECT MedicineName, MedicinePrescribedDate  
     FROM Medicine  
     WHERE MedicineName LIKE '%' + @MedicineName + '%'  
     ORDER BY MedicinePrescribedDate DESC  
377 =END  
378 GO  
  
380 -- Call the stored procedure  
381 =EXEC SearchMedicineName @MedicineName = 'Chemotherapy'  
382  
383
```

Below the script, the 'Messages' section shows the command completed successfully with a completion time of 2024-04-03T18:37:47.0667000+01:00. At the bottom right, it says 'Query executed successfully.'

**Figure 1.5.2:** I named the stored procedure as “**SearchMedicineName**” intended to fetch records from the **Medicine** table. To retrieve the “**MedicineName**” and “**MedicinePrescribedDate**” columns from the **Medicine** table, a “**SELECT**” query is used at the beginning. Only the records having a “**MedicineName**” that contains the input string (given by the **@MedicineName** parameter) are included, with the WHERE clause’s filtering of the data. To ensure that the most current prescriptions display first the “**ORDERBY**” clause is introduced and “**DESC**” also used to put the result in descending order based on the “**MedicinePrescribedDate**” column. The Stored procedure runs successfully:

**Figure 1.5.3:** The `@MedicineName` argument is passed to the value ‘Chemotherapy’, and used the “**EXEC**” keyword to search using the stored procedure.

**Q4b.**

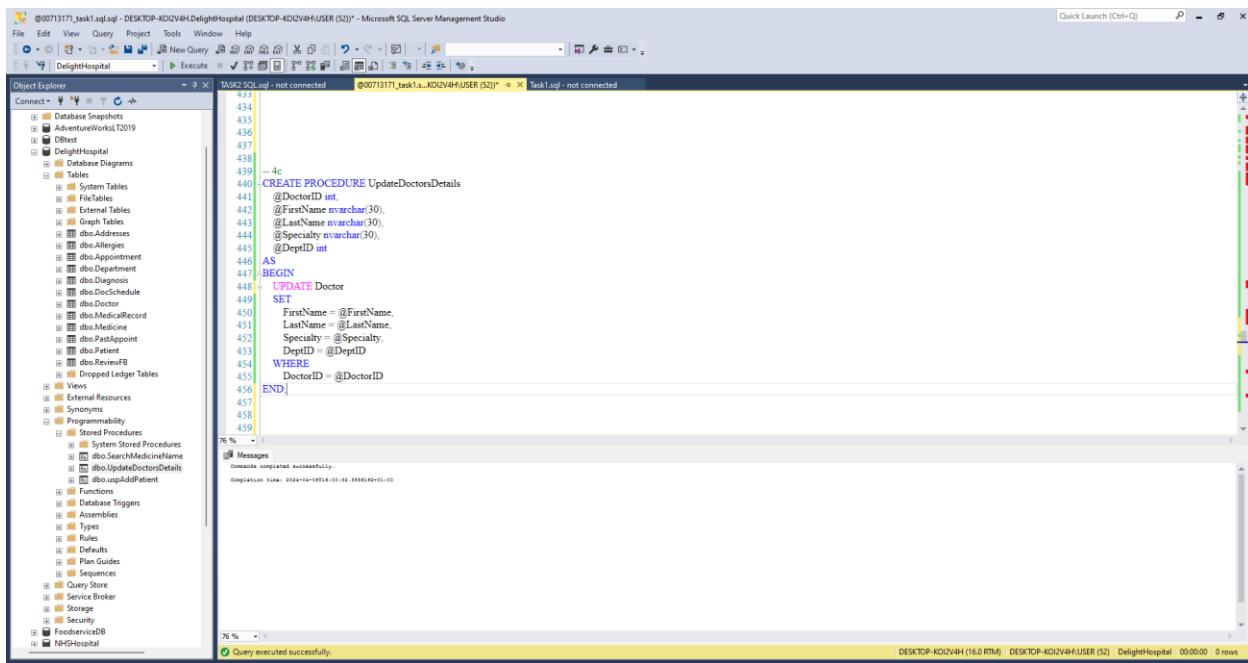
```

406 --- You can call this stored procedure and pass the patient's ID as a parameter to get the diagnoses
407 --- and allergies for that patient's appointment scheduled for today. For example:
408 EXEC GetPatientDiagnosisAndAllergies @PatientID = 1;
409
410 --- It returns a full list of diagnosis and allergies for a specific patient who has an appointment today
411 --- (i.e., the system date when the query is run).
412 --CREATE FUNCTION GetPatientDiagnosisAndAllergies (@PatientID INT)
413 RETURNS TABLE
414 AS
415   RETURN
416   (
417     SELECT D.DiagnosisName AS Diagnosis, A.AllergyName AS Allergy
418     FROM Appointment AP
419       JOIN Patient P ON AP.PatientID = P.PatientID
420       JOIN Diagnosis D ON P.PatientID = MR.PatientID
421       JOIN MRRecord MR ON MR.MRRecordID = D.MRRecordID
422       JOIN Allergies A ON P.PatientID = A.PatientID
423       WHERE P.PatientID = @PatientID
424       AND CONVERT(DATE, AP.AppointmentDate) = GETDATE()
425
426
427
428
429
430 -- Execute the UDF for a specific patient ID
431 --SELECT * FROM GetPatientDiagnosisAndAllergies(5); -- Replace with the actual patient ID
432
433
434
435
436

```

**Figure 1.5.4:** At the explorer window we can see the User defined function installed. The function accepts a specific patient ID as input and outputs a table containing the patient’s diagnosis and allergy details. To obtain the results, I use the select statement to run the function name with the intended patient ID it was successful but there was no appointment today.

## Q4c. Updating the details for an existing doctor



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure for 'DelightHospital'. The 'Scripting' tab in the ribbon is selected. A code editor window displays the following T-SQL script:

```
437
438 -- 4c
439 CREATE PROCEDURE UpdateDoctorsDetails
440     @DoctorID int,
441     @FirstName nvarchar(30),
442     @LastName nvarchar(30),
443     @Specialty nvarchar(30),
444     @DeptID int
445
446 BEGIN
447     UPDATE Doctor
448     SET
449         FirstName = @FirstName,
450         LastName = @LastName,
451         Specialty = @Specialty,
452         DeptID = @DeptID
453
454 WHERE
455     DoctorID = @DoctorID
456 END
```

The status bar at the bottom right indicates "Query executed successfully".

**Figure 1.5.5:** I created the stored procedure to update Doctor records by first giving it a name “UpdateDoctorDetails”. I took certain parameter for example doctors ID = (@DoctorID) etc.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'DelightHospital' database is selected. In the center pane, a query window titled 'Task1.sql - not connected' contains the following code:

```

457
458 select * from Doctor;
459
460
461
462
463
464
465
466
467 -- Call this stored procedure and pass the DoctorID along with the updated details as parameters to update the doctor's information.
468 EXEC UpdateDoctorDetails
469 @DoctorID = 7,
470 @FirstName = 'Anderson',
471 @LastName = 'Clinton',
472 @Specialty = 'Psychiatry',
473 @DeptID = 7;
474
475
476
477
478
479
480
481
482
483

```

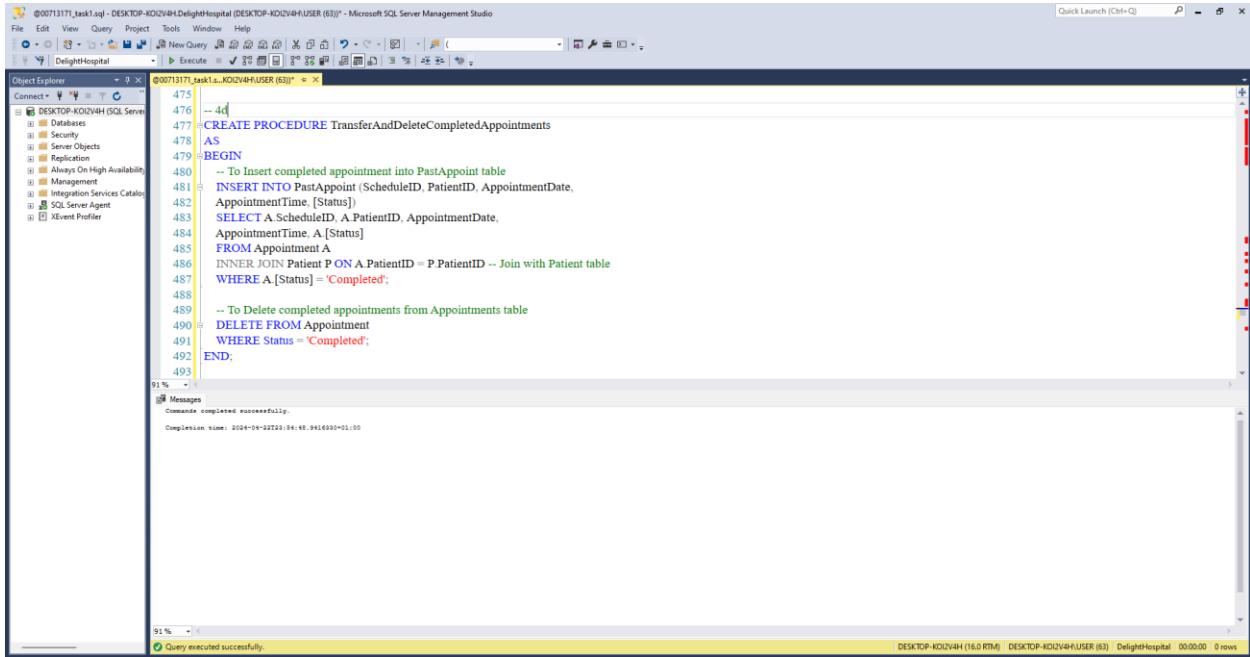
The results pane shows a table with the following data:

	DoctorID	FirstName	LastName	Specialty	DeptID
1	1	John	Doe	Cardiologist	1
2	2	Sarah	Smith	Pediatrician	2
3	3	Michael	Jordan	Orthopedic Surgeon	3
4	4	Emily	Brown	Oncologist	4
5	5	David	Lee	Neurologist	5
6	6	James	Smith	Gastroenterologist	6
7	7	Clynton	Anderson	Forensic Psychiatry	7

At the bottom of the results pane, it says 'Query executed successfully.'

**Figure 1.5.6:** Then the stored procedure updates the supplied data for the designated doctor ID by invoking the stored procedure using **EXEC** as shown in the slide above. I updated the record of doctor with Doctor ID “7” by correcting his name from **Clinton** to **Clynton**, and updating the specialty from **Psychiatry** to **Forensic Psychiatry**.

#### Q4d. Delete the appointment whos' status is already completed.



The screenshot shows the Microsoft SQL Server Management Studio interface. A new stored procedure has been created and is displayed in the center pane. The code is as follows:

```
CREATE PROCEDURE TransferAndDeleteCompletedAppointments
AS
BEGIN
    -- To Insert completed appointment into PastAppoint table
    INSERT INTO PastAppoint (ScheduleID, PatientID, AppointmentDate,
    AppointmentTime, [Status])
    SELECT A.ScheduleID, A.PatientID, AppointmentDate,
    AppointmentTime, A.[Status]
    FROM Appointment A
    INNER JOIN Patient P ON A.PatientID = P.PatientID -- Join with Patient table
    WHERE A.[Status] = 'Completed';

    -- To Delete completed appointments from Appointments table
    DELETE FROM Appointment
    WHERE Status = 'Completed';
END;
```

In the bottom right corner of the interface, there is a status bar with the text "DESKTOP-KOIZV4H (16.0 RTM) | DESKTOP-KOIZV4H\USER (6) | DelightHospital | 00:00:00 | 0 rows".

**Figure 1.5.7:** After giving the stored procedure a name, I've linked the Patient and the Appointment table together using INNER JOIN to move the historical record, the “INSERT INTO” statement ensures the authenticity of related patients by transferring completed appointments from the Appointment table to the PastAppoint table. Immediately after the record is moved it is then deleted from the Appointment table.

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "000713171\_task1.sql - DESKTOP-KOIZV4H\DelightHospital (DESKTOP-KOIZV4H\USER (63)) - Microsoft SQL Server Management Studio". The main window contains a query editor with the following T-SQL code:

```
493
494
495 -- To Exec the move and deletion procedure
496 EXEC TransferAndDeleteCompletedAppointments;
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
```

Below the code, the status bar displays "Query executed successfully".

**Figure 1.5.8:** Above is the TSQL **EXEC** command query that runs the '**TransferAndDeleteCompletedAppointment**' stored procedure although no completed appointment at this time due and date in future.

## Q5. Creating View

```

CREATE VIEW DoctorAppointmentsView AS
SELECT
    A.AppointmentID,
    A.AppointmentTime,
    A.AppointmentDate,
    D.FirstName + ' ' + D.LastName AS DoctorName,
    D.Specialty AS DoctorSpecialty,
    Dep.DepName AS Department,
    NULL AS DoctorReview
FROM
    Appointment A
JOIN
    Patient P ON A.PatientID = P.PatientID
JOIN
    DocSchedule DS ON A.ScheduledID = DS.ScheduleID
JOIN
    Doctor D ON DS.DoctorID = D.DoctorID
JOIN
    Department Dep ON D.DepID = Dep.DepID
UNION
SELECT
    PA.AppointmentID,
    PA.AppointmentDate,
    PA.AppointmentTime,
    D.FirstName + ' ' + D.LastName AS DoctorName,
    D.Specialty AS DoctorSpecialty,
    Dep.DepName AS Department,
    R.Review AS DoctorReview
FROM
    PastAppointment PA
JOIN
    Patient P ON PA.PatientID = P.PatientID
JOIN
    Appointment A ON P.PatientID = A.PatientID
JOIN
    DocSchedule DS ON A.ScheduledID = DS.ScheduleID
JOIN
    Doctor D ON DS.DoctorID = D.DoctorID
JOIN
    Department Dep ON D.DepID = Dep.DepID
JOIN
    ReviewFB R ON PA.PastAppointmentID = R.PastAppointmentID
    
```

Query executed successfully.

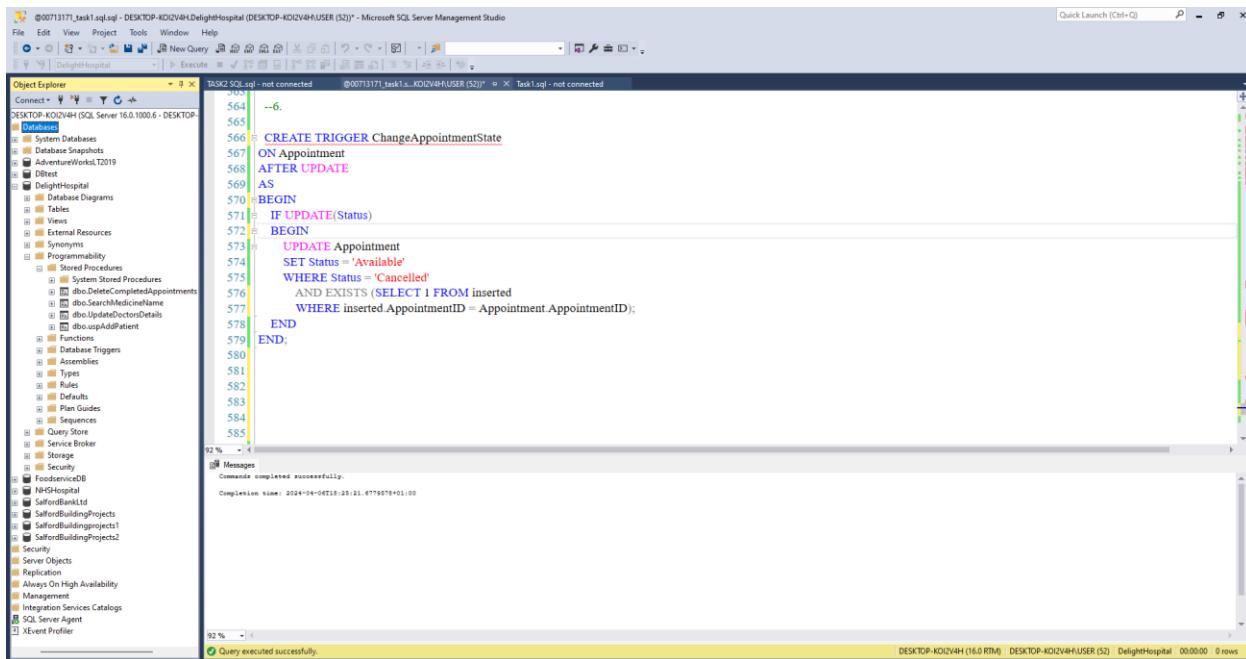
Figure 1.5.9: In this view query I've used the UNION operator to combine results of two (2) SELECT statements to become one result set. First query retrieve details of Appointment table while the second returns record of Pastappointment, they select similar query only for the second query that includes review/feedback for doctors from past appointments. I also ensured datatypes are the same to avoid error, finally I aliased NULL value for Doctor review on the first select statement to align the structure of the result sets for the UNION operation with the second select statement.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, under the 'DelightHospital' database, there is a node for 'Views' which contains 'dbo.DoctorAppointmentsView'. A tooltip above the view node says: '--To view the data from the DoctorAppointmentView'. Below this, a query window displays the T-SQL code: 'SELECT \* FROM DoctorAppointmentsView;'. The results grid shows 14 rows of appointment data. The columns are: AppointmentID, AppointmentDate, AppointmentTime, DoctorName, DoctorSpecialty, Department, and DoctorReview. The DoctorReview column contains several positive comments from patients.

AppointmentID	AppointmentDate	AppointmentTime	DoctorName	DoctorSpecialty	Department	DoctorReview
1	2024-01-01	10:00:00 0000000	John Doe	Cardiologist	Cardiology	The doctor was very attentive and knowledgeable.
2	2024-06-02	08:00:00 0000000	John Doe	Cardiologist	Cardiology	NULL
3	2024-06-03	11:00:00 0000000	Jane Smith	Pediatrician	Pediatrics	I had to wait a long time before being seen.
4	2024-06-03	09:30:00 0000000	Jane Smith	Pediatrician	Pediatrics	NULL
5	2024-01-03	13:00:00 0000000	Michael Johnson	Orthopedic Surgeon	Orthopedics	The appointment was fine and quick.
6	2024-06-03	10:00:00 0000000	Michael Johnson	Orthopedic Surgeon	Orthopedics	NULL
7	2024-01-03	07:00:00 0000000	Emily Brown	Oncologist	Oncology	The staff was friendly and helpful.
8	2024-06-05	09:00:00 0000000	Emily Brown	Oncologist	Oncology	NULL
9	2024-01-03	09:00:00 0000000	David Lee	Neurologist	Neurology	I had a positive experience overall.
10	2024-06-06	06:00:00 0000000	David Lee	Neurologist	Neurology	NULL
11	2024-06-07	13:00:00 0000000	James Smith	Gastroenterologist	Gastroenterology	The appointment process was smooth and efficient.
12	2024-06-07	11:00:00 0000000	James Smith	Gastroenterologist	Gastroenterology	NULL
13	2024-01-03	13:00:00 0000000	Christina Anderson	Forensic Psychiatry	Psychiatrist	The nurse was very kind and compassionate.
14	2024-06-08	12:00:00 0000000	Christina Anderson	Forensic Psychiatry	Psychiatrist	NULL

**Figure 1.6.0:** Above is the TSQL SELECT statement to view the stored View table with name “**DoctorAppointmentView**”.

**Q6.** Creating a trigger so that the current state of an appointment can be changed to available when it is cancelled:



The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer pane displays the database structure of 'DelightHospital'. In the center, the 'TSQL2.SQl - not connected' window contains the following T-SQL code:

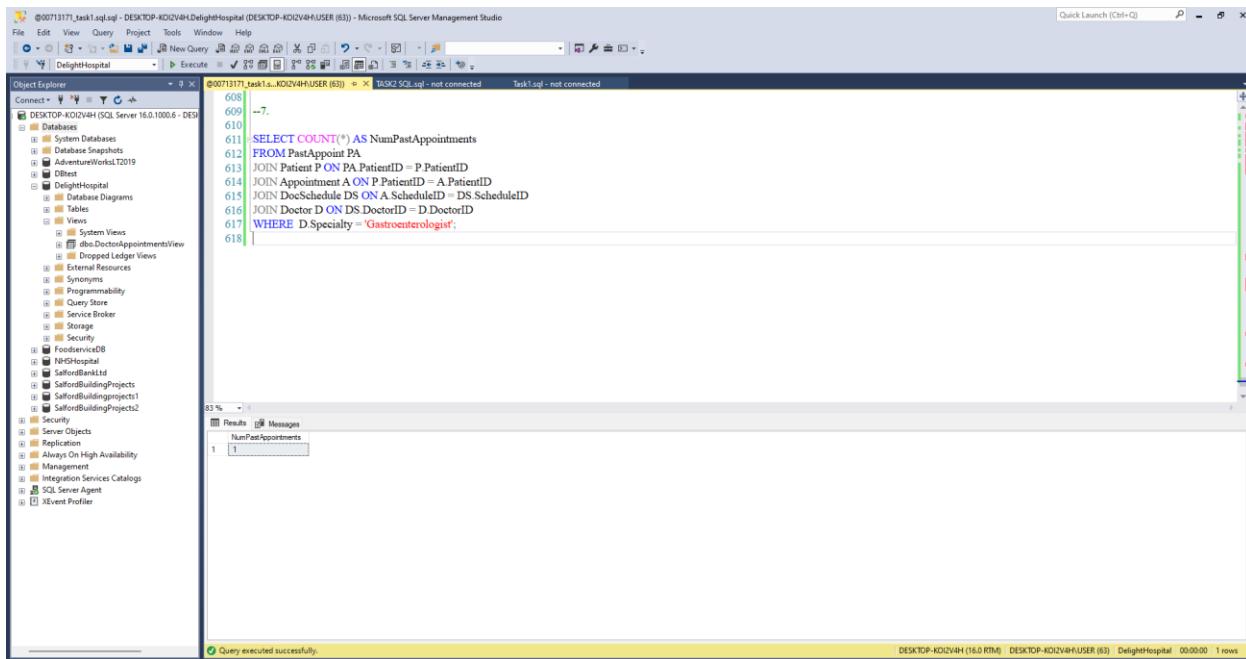
```
563 --6.
564
565
566 CREATE TRIGGER ChangeAppointmentState
567 ON Appointment
568 AFTER UPDATE
569 AS
570 BEGIN
571 IF UPDATE(Status)
572 BEGIN
573 UPDATE Appointment
574 SET Status = 'Available'
575 WHERE Status = 'Cancelled'
576 AND EXISTS (SELECT 1 FROM inserted
577 WHERE inserted.AppointmentID = Appointment.AppointmentID);
578 END
579 END;
```

The code defines a trigger named 'ChangeAppointmentState' on the 'Appointment' table. It triggers after an update. If the 'Status' column is updated, it checks if the new value is 'Cancelled'. If so, it updates all rows in the 'Appointment' table where the status is still 'Cancelled' to 'Available', but only if the appointment ID in the 'inserted' table matches the one being updated in the 'Appointment' table.

In the bottom right corner of the central window, there is a message: 'Query executed successfully.'

**Figure 1.6.1:** The TSQL Trigger above checks if the status column has been updated to “Cancelled” and then automatically updates the states of the cancelled appointment to “Available”. It guards against inadvertent adjustments to other appointments and guarantees that only the appointments that are being updated are impacted.

**Q7.** Write a query that allows the hospital to identify the number of completed appointments with the specialty of doctors as ‘**Gastroenterologist**’.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a tree view of databases is visible, including 'DelightHospital' which is expanded to show 'Tables', 'Views', and 'Stored Procedures'. A query window titled 'Task1.sql - not connected' contains the following SQL code:

```
608 |--7.
609 |
610 |
611 |SELECT COUNT(*) AS NumPastAppointments
612 |FROM PastAppoint PA
613 |JOIN Patient P ON PA.PatientID = P.PatientID
614 |JOIN Appointment A ON P.PatientID = A.PatientID
615 |JOIN DocSchedule DS ON A.ScheduleID = DS.ScheduleID
616 |JOIN Doctor D ON DS.DoctorID = D.DoctorID
617 |WHERE D.Specialty = 'Gastroenterologist';
618 |
```

The results pane below shows the output of the query:

NumPastAppointments
1

A status bar at the bottom indicates 'Query executed successfully.'

**Figure 1.6.2:** This SQL code connects tables like PastAppoint, Patient, Appointment, DocSchedule, and Doctor to track previous appointments for patients with gastrointestinal specialists.

Q8.

## **EXPLAINING WHAT I HAVE CONSIDERED IN YOUR DATABASE FOR:**

### **DATA INTEGRITY AND CONCURRENCY**

Primary and Foreign keys: In my tables I've established connections between data points, preserving referential integrity by ensuring distinct identity for each record and persistent linking of related material

I've established check constraints to ensure data consistency and cleanliness, such as correct email format and secure passwords, to maintain a clean and secure database. Be committed to continuous surveillance, examining and verifying the data to identify potential issues promptly and ensuring the reliability and trustworthiness of information's.

### **DATA SECURITY**

The importance of using object permissions and database schemas to enhance the security of **DelightHospital** database;

Using the Database schema to organize and manage object like tables, stored procedures, functions, and views, streamlining administration and ownership. They also allow for granular permission control, granting permissions to all objects within the schema, simplifying security configuration and reducing administration overhead.

“View-level security” regulates and limits access for their roles. Combining table-level and view-level security can improve security without compromising data usefulness, thereby preserving **Patient** privacy. Some and more Stored procedure can be utilized to restrict access to data while maintaining security by enclosing business logic within these procedures, thereby reducing the risk of unwanted access.

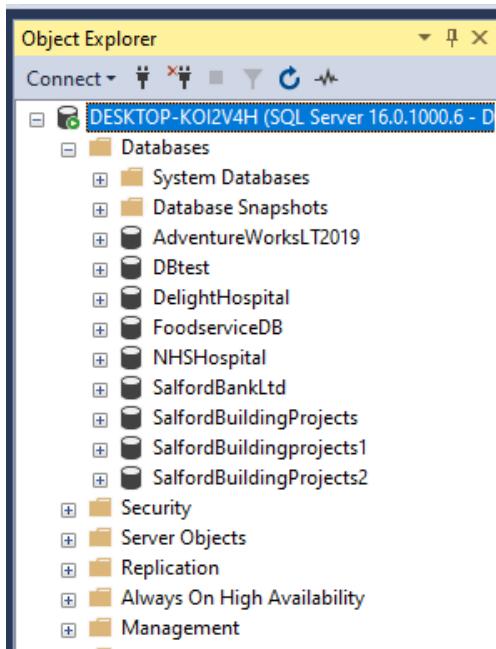
Finally to protect against SQL injection attacks, it's crucial to implement string escaping techniques and built-in methods like PHP's `mysql_real_escape_string()`. Avoid keeping Java/JSP scripts accessible over HTTP, store configuration files in secure locations, and limit database access to trusted clients at the port level or through database capabilities.

## DATABASE BACKUP AND RECOVERY

### Steps In Performing a Full Database Backup Using SSMS

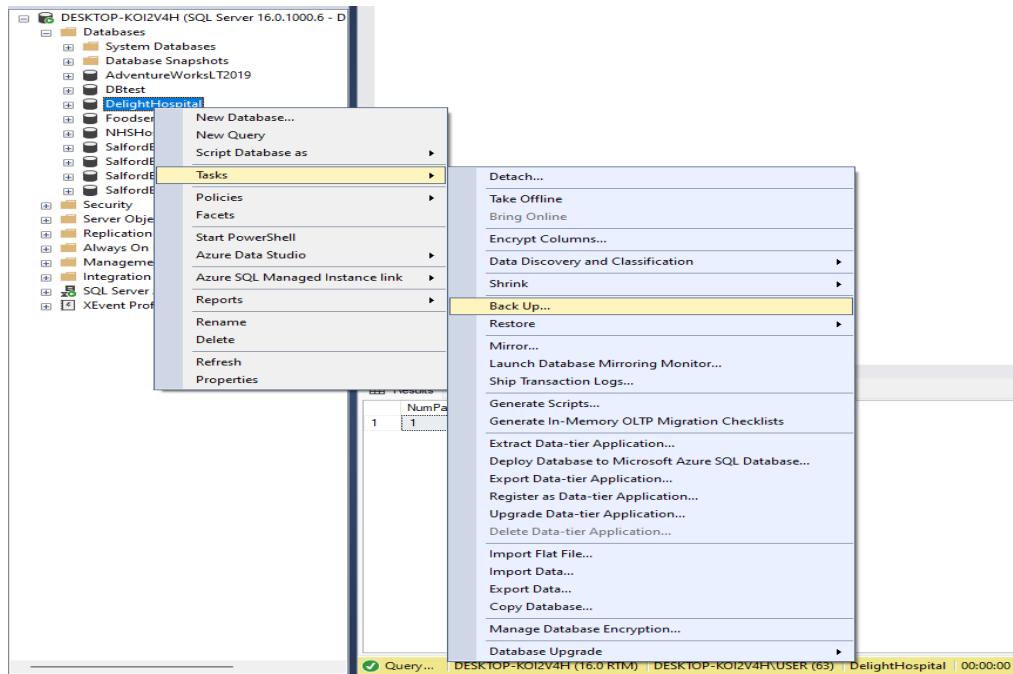
In providing my client with advice and guidance on backup and recovery I've shown him steps in carrying out the process using SSMS

1. I've created a folder called **DelightHospital\_Backup** in the 'C:' drive, then I opened my SSMS and connected to an instance of SQL Server then expanded the server tree and then expand the database folder as seen in **Figure 1.6.3** below



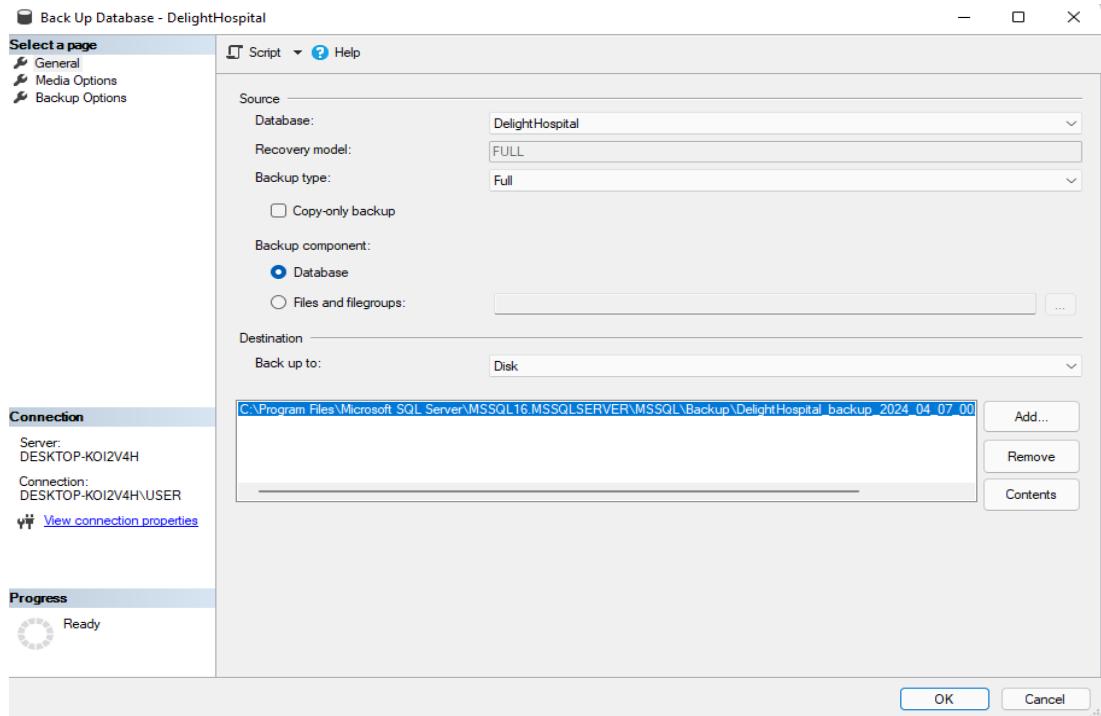
**Figure 1.6.3**

2. I right clicked the DelightHospital database and selected Tasks and then Backup as seen in **Figure 1.6.4** below



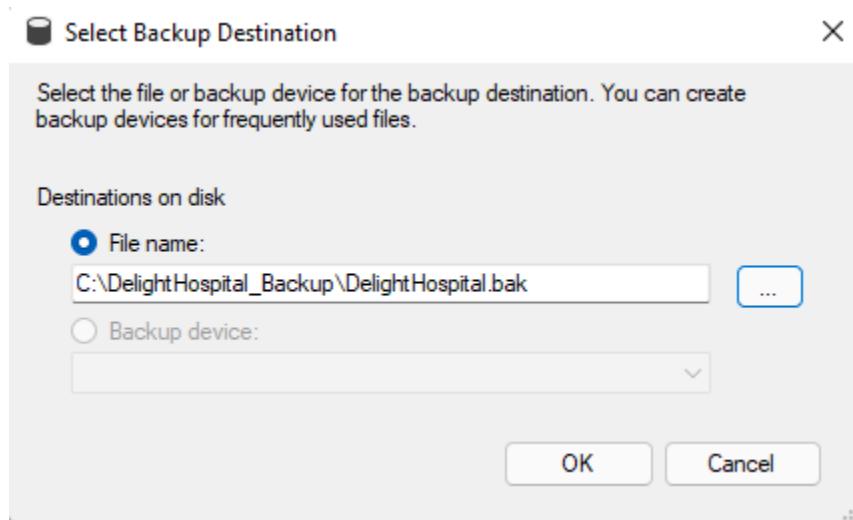
**Figure 1.6.4:**

3. I've left my backup type as full also confirmed that I'm on the right database and then I selected Disk instead of URL, I clicked remove to replace the automatically included location and then added my backup location as shown below in **Figure 1.6.5**:



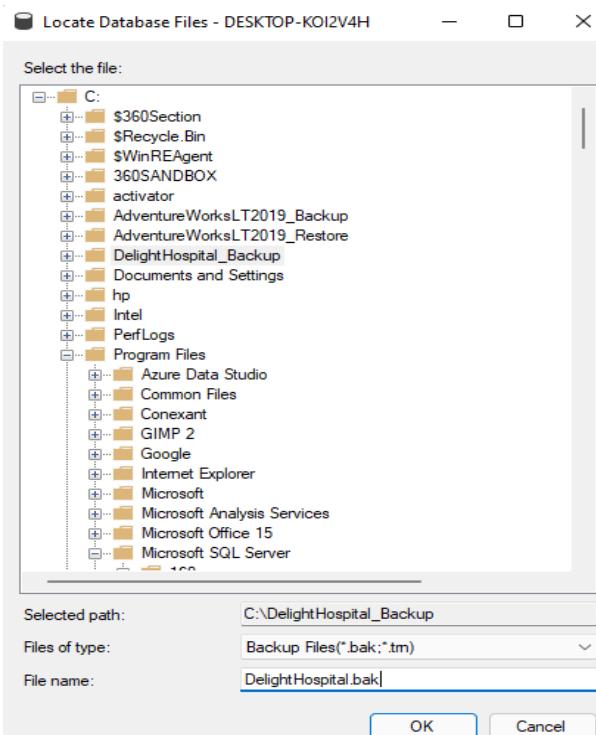
**Figure 1.6.5**

4. The dialog box below called “Select Backup Destination” shows when I click ok, now on this box I clicked the ellipsis (...) after selecting File name as shown below in **Figure 1.6.6**:



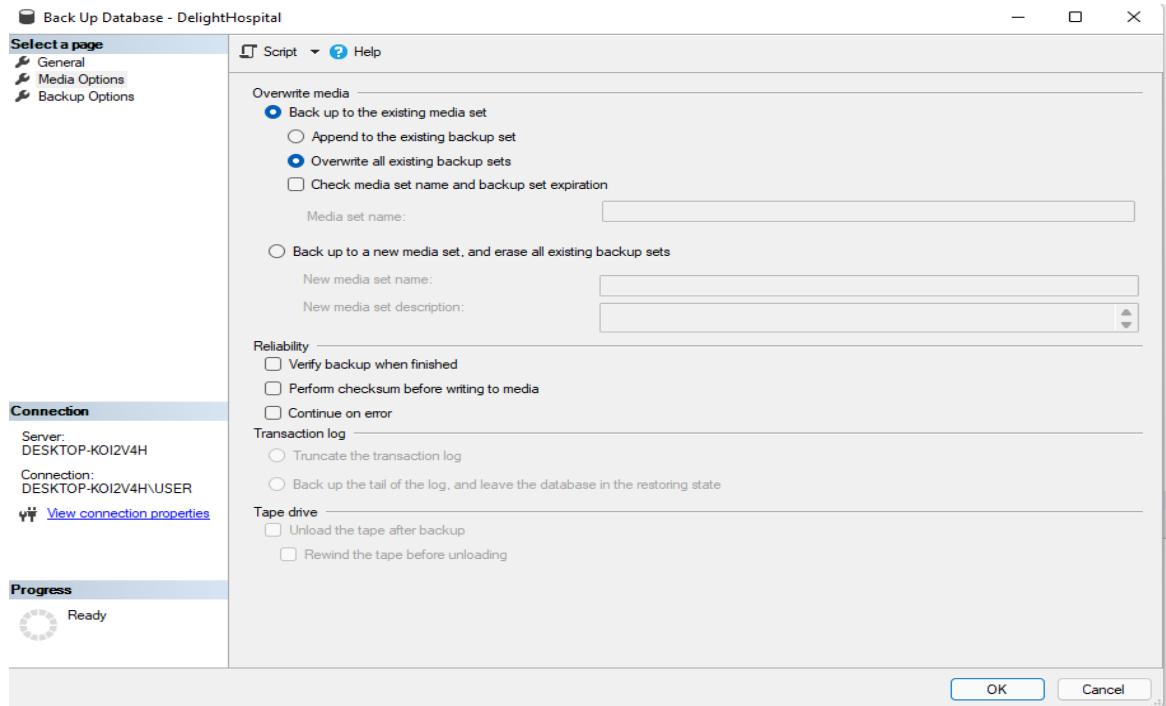
**Figure 1.6.6**

5. In the Locate Database Files dialog box, I located my backup file and then for the File name below I typed **DelightHospital.bak** and then clicked ok twice as seen in **Figure 1.6.7**.



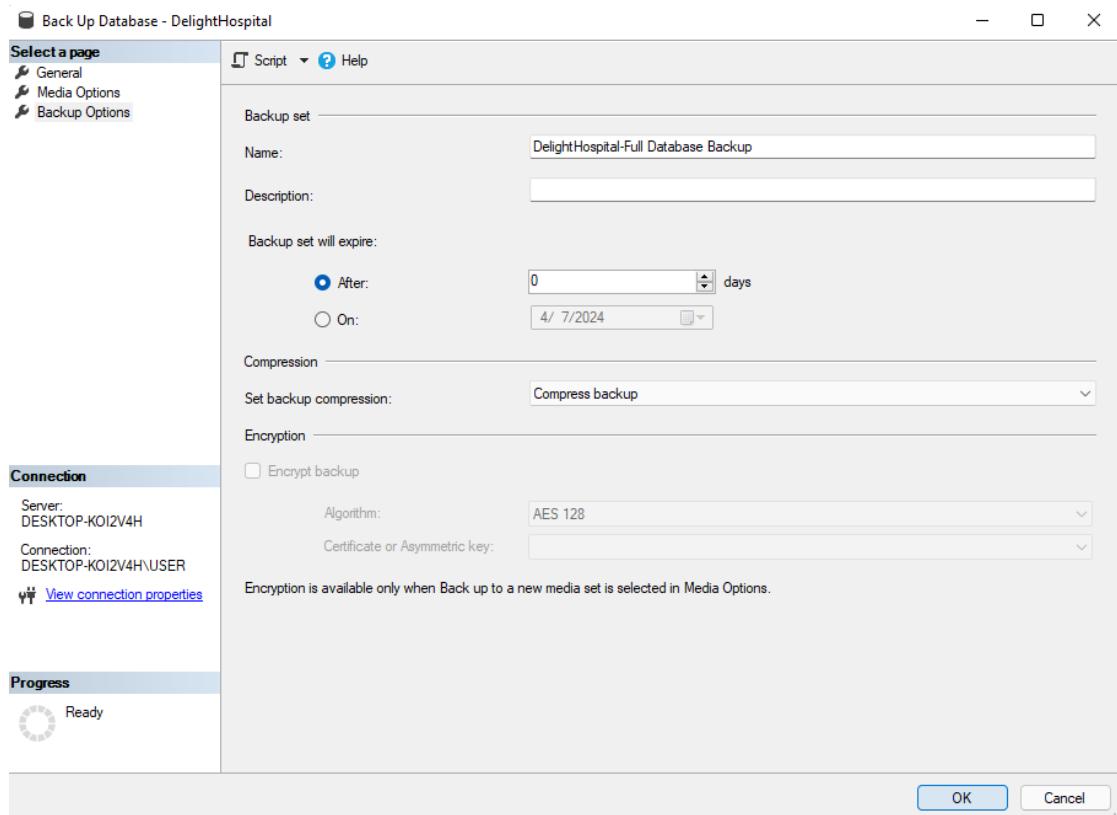
**Figure 1.6.7**

6. In the Select a Page pane of the Back Up Database dialogue box, I selected **Media Options**. In the first section I picked **Overwrite All Existing Backup Sets option**. I left the reliability section as seen in **Figure 1.6.8** Below.



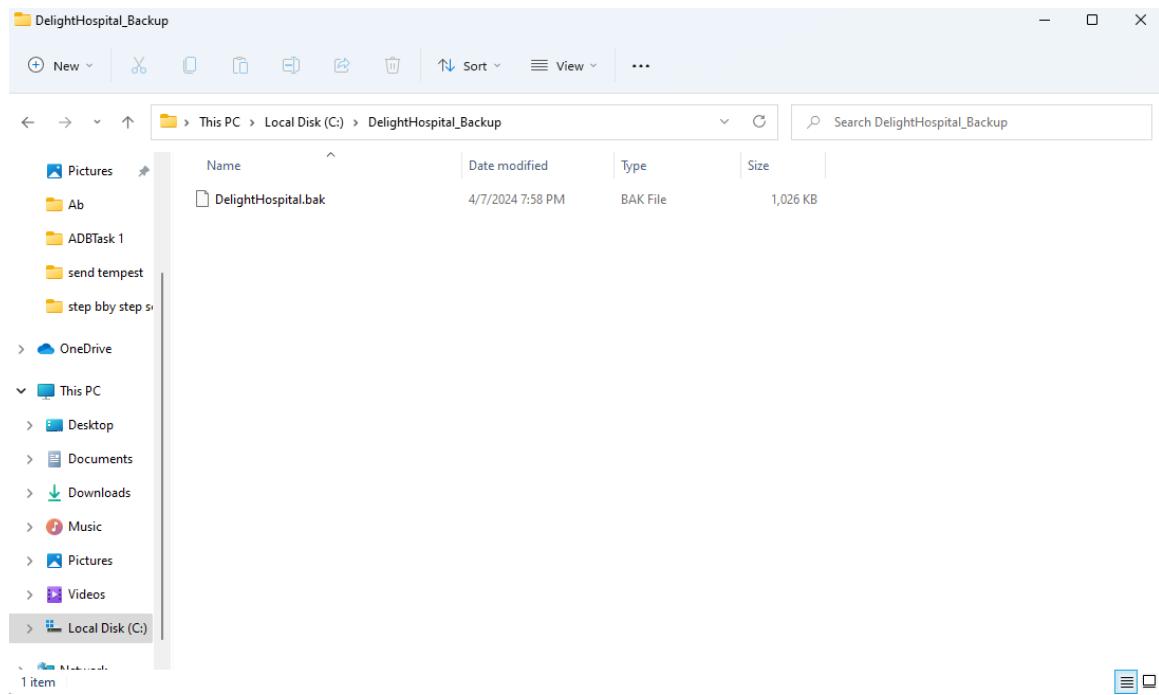
**Figure 1.6.8**

7. In this stage I selected ‘compress backup’ as seen in **Figure 1.6.9** Below.



**Figure 1.6.9**

8. MY back Up is complete after the step above and then I can find my back up in the folder location as seen in **Figure 1.7.0** Below.

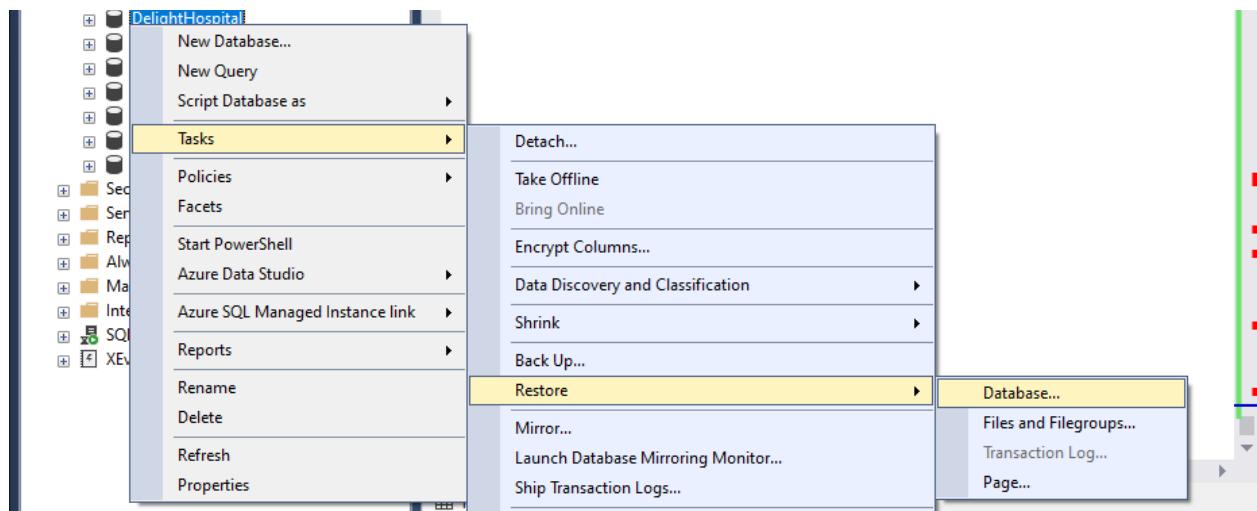


**Figure 1.7.0**

STEPS FOR RESTORING THE DELIGHTHOSPITAL DATABASE IS SHOWN FROM ONE SCREENSHOT TO THE OTHER;

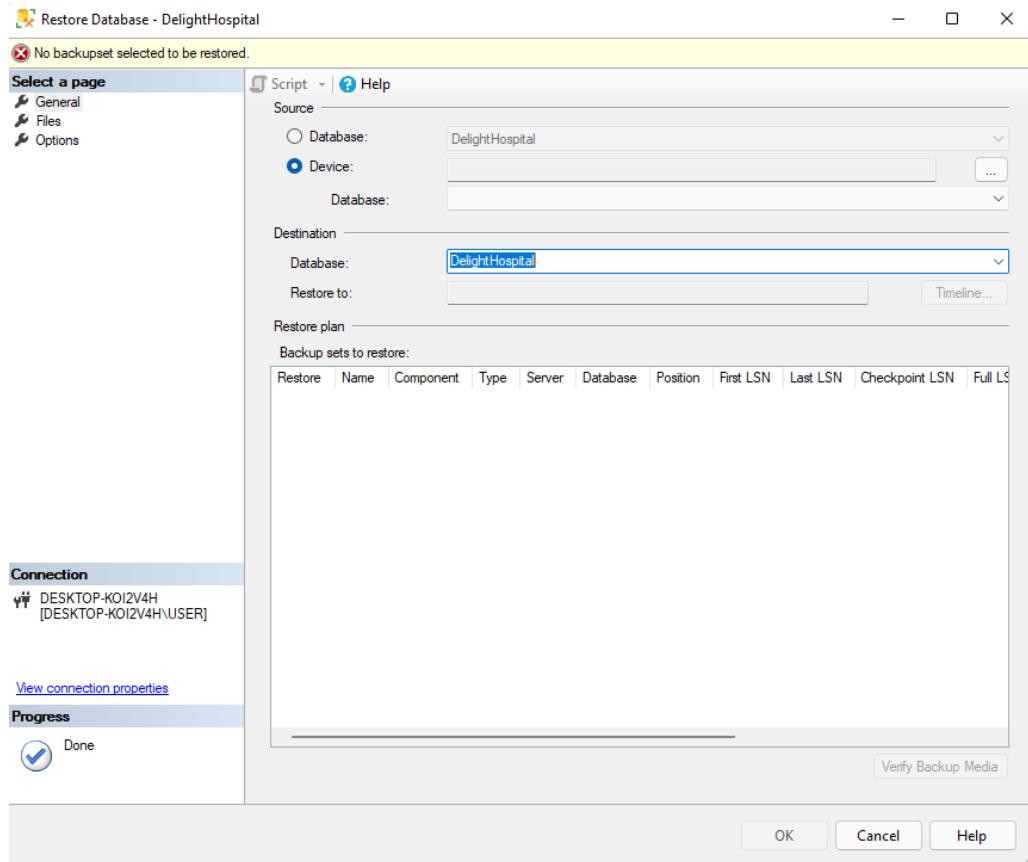
1. Right click on the Database, Task, Restore, Database as seen in the

**Figure 1.7.1:**



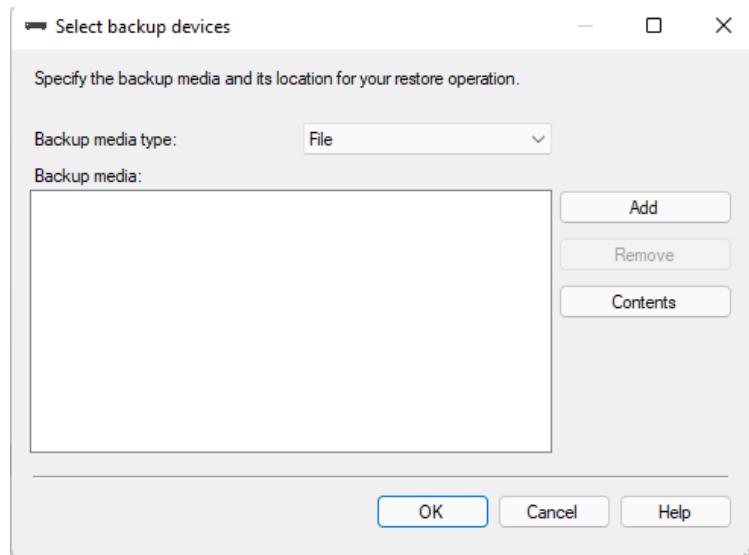
**Figure 1.7.1**

Select Device option, click on the “...” button, as seen in the **Figure 7.2** below:



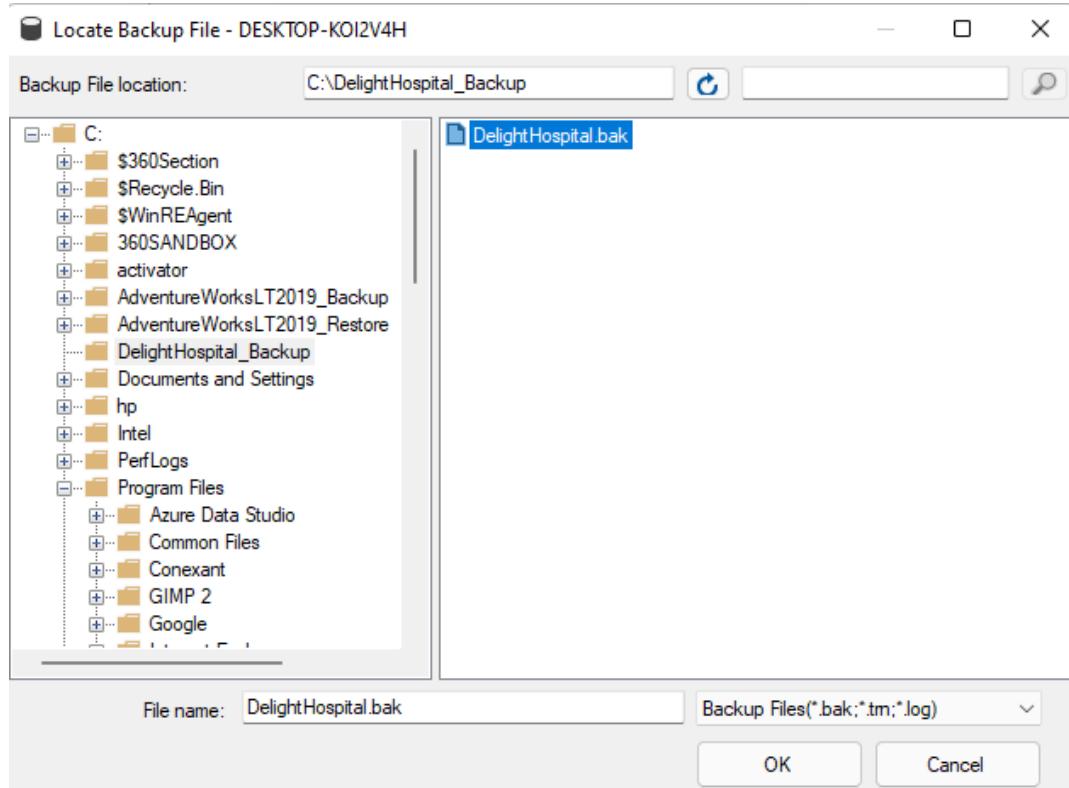
**Figure 1.7.2**

2. **Select backup devices** dialog box option then select File for Backup media type as seen in **Figure 7.3** Below. Click on the **ADD** button.



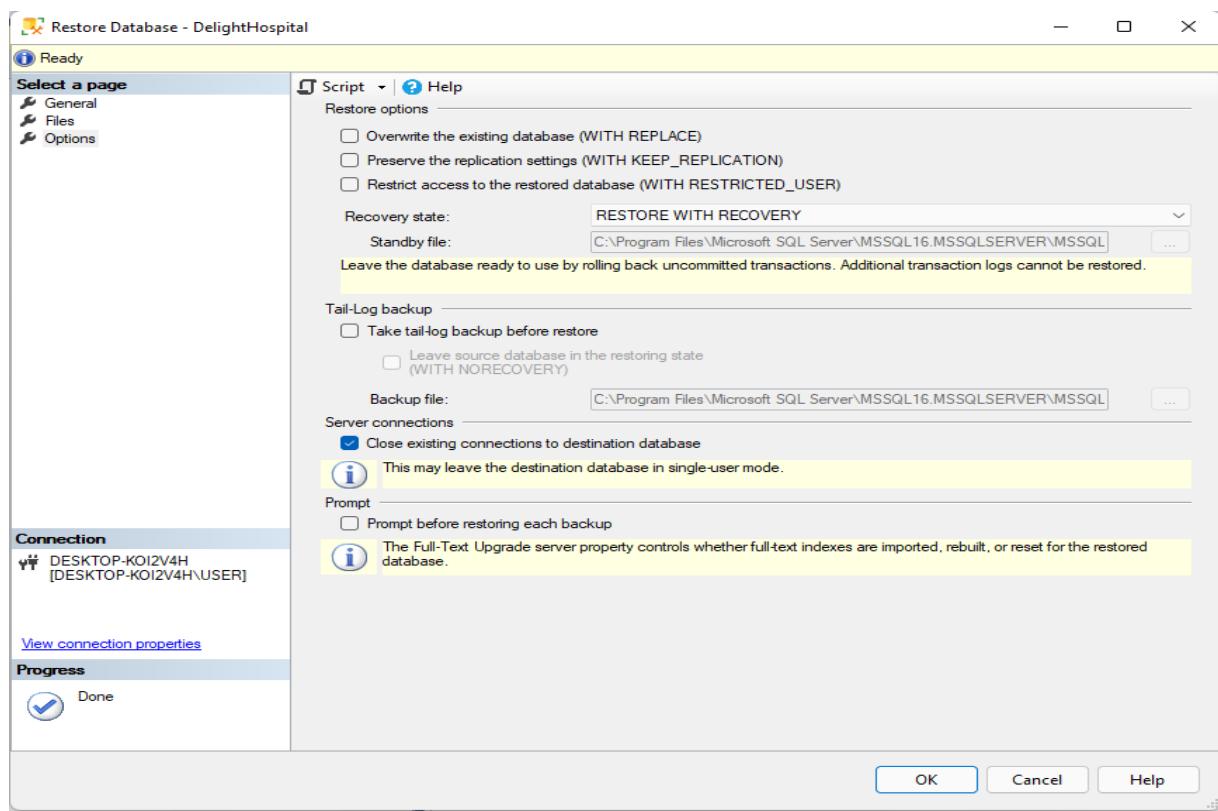
**Figure 1.7.3**

3. Select the **DelightHospital.bak** file from Local drive as shown in the diagram below



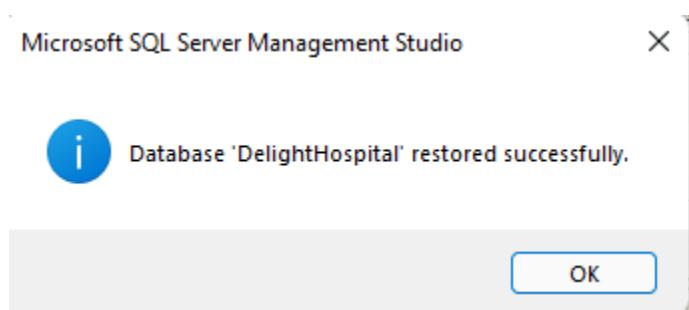
**Figure 1.7.4**

4. Select the **Close Existing Connections to Destination Database** option in the Server Connections section. Click **OK**



**Figure 1.7.5**

5. Click **OK**



**Figure 1.7.6**

## CONCLUSION

The **DelightHospital** database is a significant advancement in data integrity, security protocols, and operational optimization. It efficiently manages patient, appointment, medical record, and staff information while maintaining confidentiality. The database structure includes tables with defined properties, ensuring quality and dependability of information through primary keys, foreign keys, and constraints.

The database is secure through strict procedures, including the use of hashing techniques to store passwords and improve access control and confidentiality. Database schemas, object permissions, and views ensure users have access to only the necessary data for their roles. The database's resilience is ensured through strong backup and recovery protocols, with regular testing to ensure data regular in case of crises or system failures.

## **TASK TWO REPORT (2)**

## **INTRODUCTION**

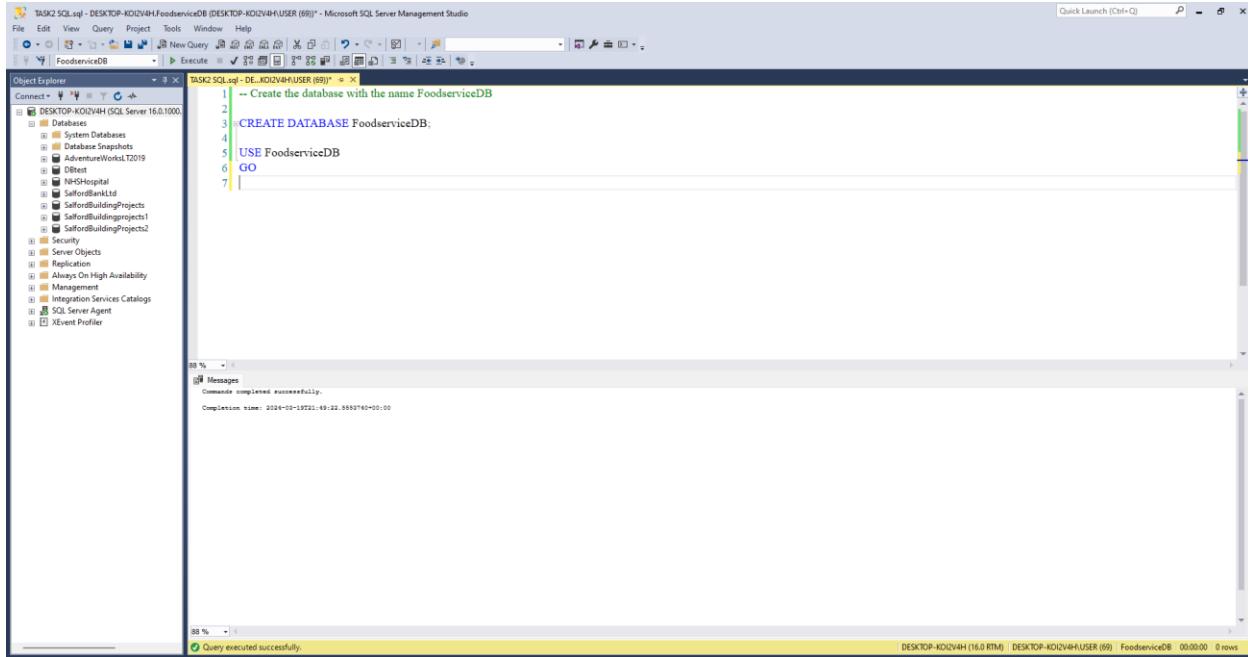
The job requires me to serve as a database consultant for a food service business. Construction of a database and data import from four related CSV files into separate tables are the goals. Restaurant cuisines, ratings, customers, and other information are all included in the provided CSV files.

I'm going to accomplish this goal by doing the following steps:

- A brand-new database called "FoodserviceDB" will be made using SQL.
- The contents of the tables are imported from the CSV files unchanged.
- All tables have the necessary primary keys added to them. However, referential integrity is protected by properly defining and maintaining foreign keys.
- A database diagram is made to illustrate the relationships between the tables in order to provide a clear image of the database structure.

The database will be properly created, the necessary data will be imported, and the tables will be correctly linked and ready for further analysis according to the previously discussed methods.

## PART 1:

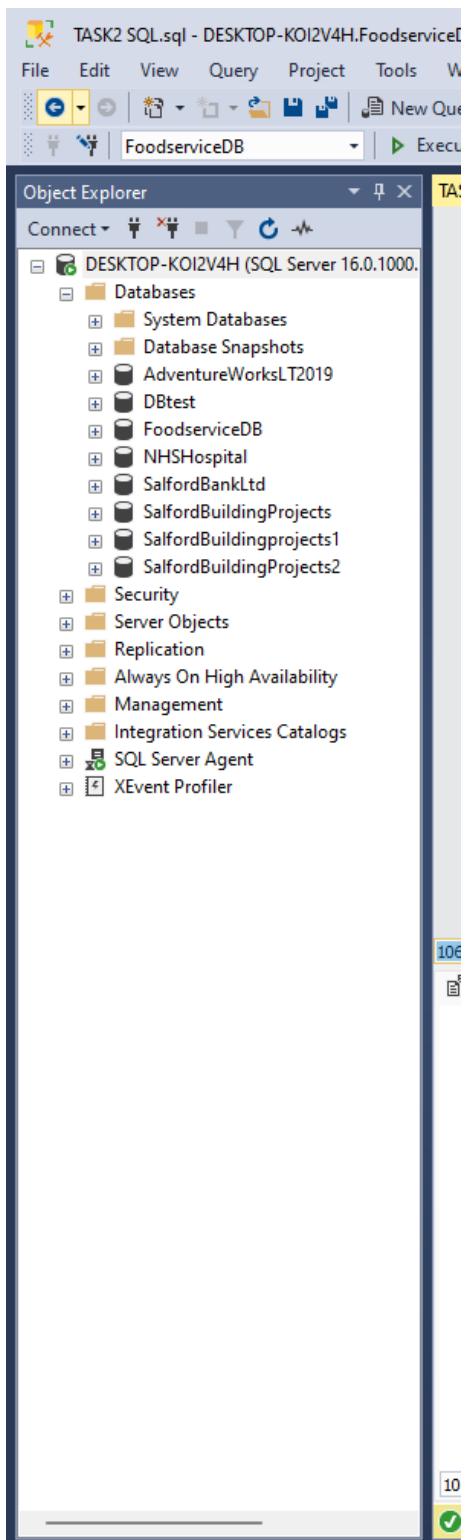


The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a connection to 'DESKTOP-KOIVYAH (SQL Server 16.0.1000)' is selected, with 'FoodserviceDB' expanded to show its objects. The 'T-SQL' tab in the top ribbon is active. In the main query window, the following T-SQL code is written:

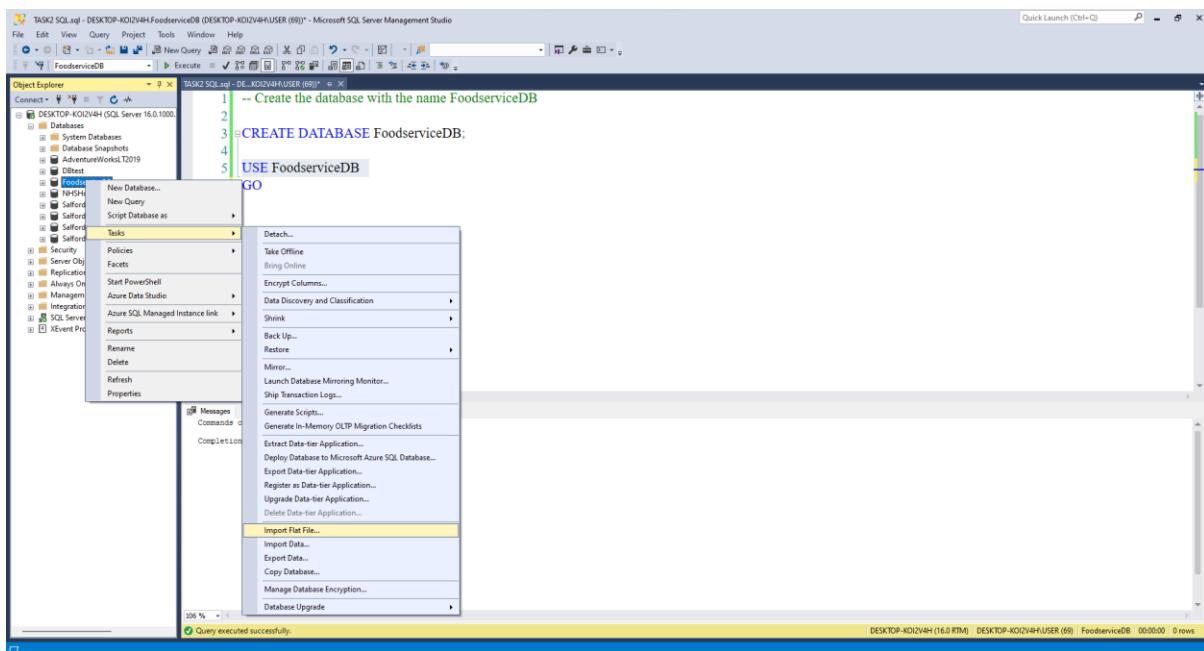
```
-- Create the database with the name FoodserviceDB
CREATE DATABASE FoodserviceDB;
USE FoodserviceDB;
GO
```

The status bar at the bottom right indicates 'DESKTOP-KOIVYAH (16.0 RTM) | DESKTOP-KOIVYAH\USER (69) | FoodserviceDB | 00:00:00 | 0 rows'.

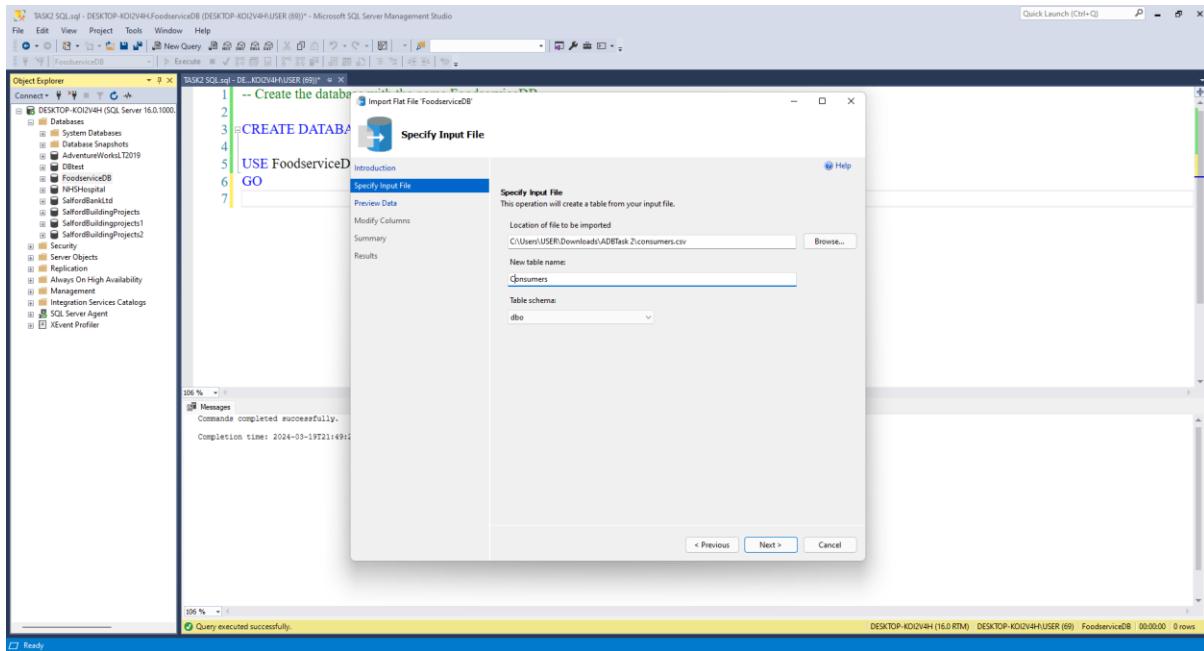
**Figure 2.1.0:** In this first step I have written a TSQL code to create a database with name **FoodServiceDB** and with the 'USE' query to change from the current database to the just created database.



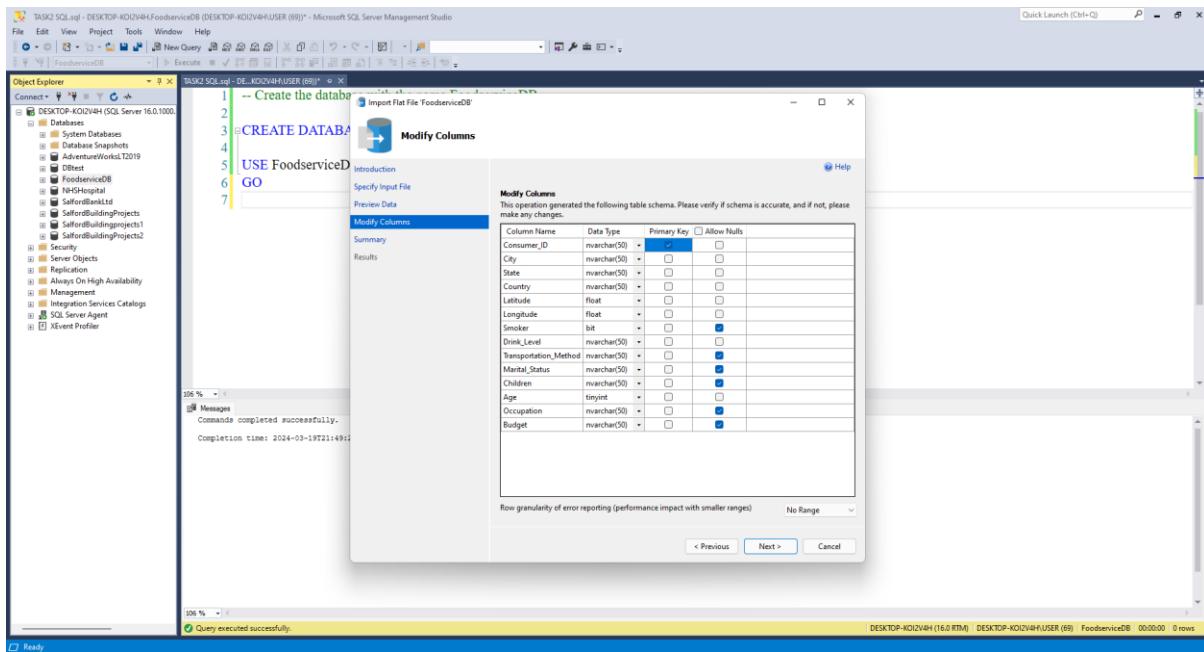
**Figure 2.1.1:** This figure shows us the successful creation of the FoodServiceDB database.



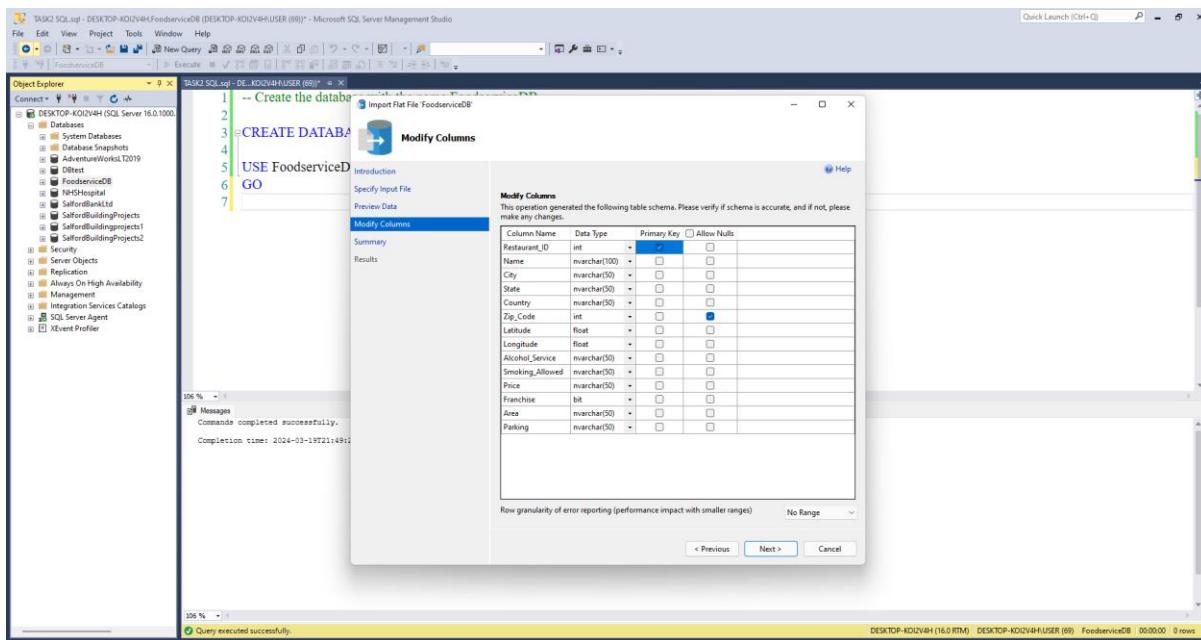
**Figure 2.1.2:** Step by step procedure to move/import files into the FoodServiceDB database



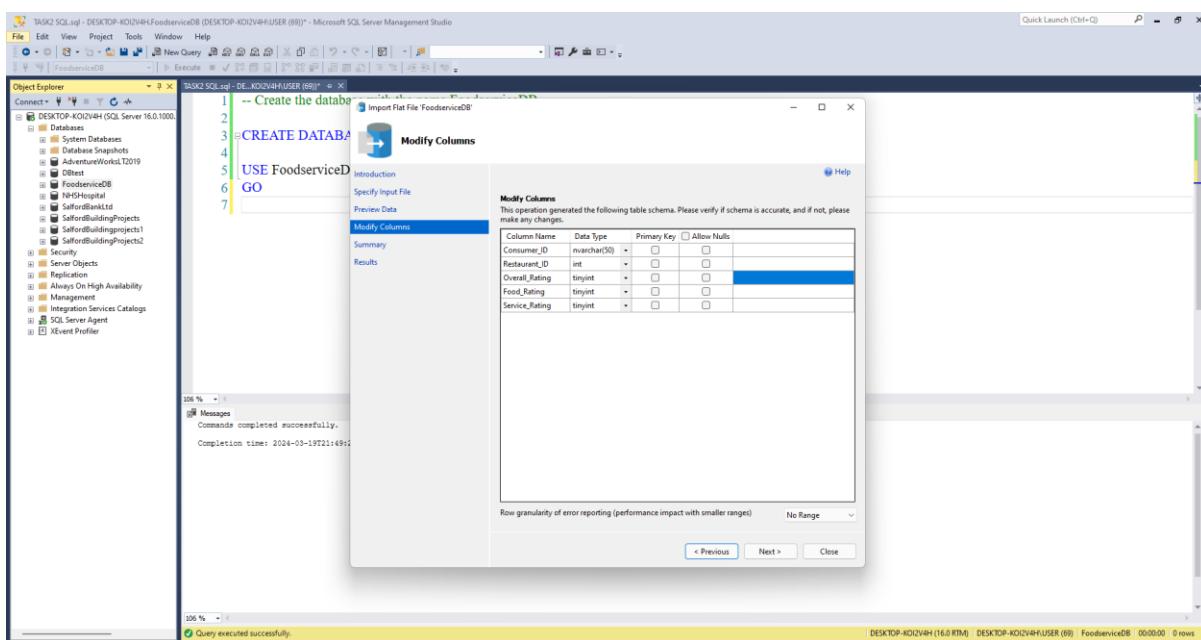
**Figure 2.1.3:** In this step while importing the consumers.csv file I rewrote the table name as requested in the assessment.



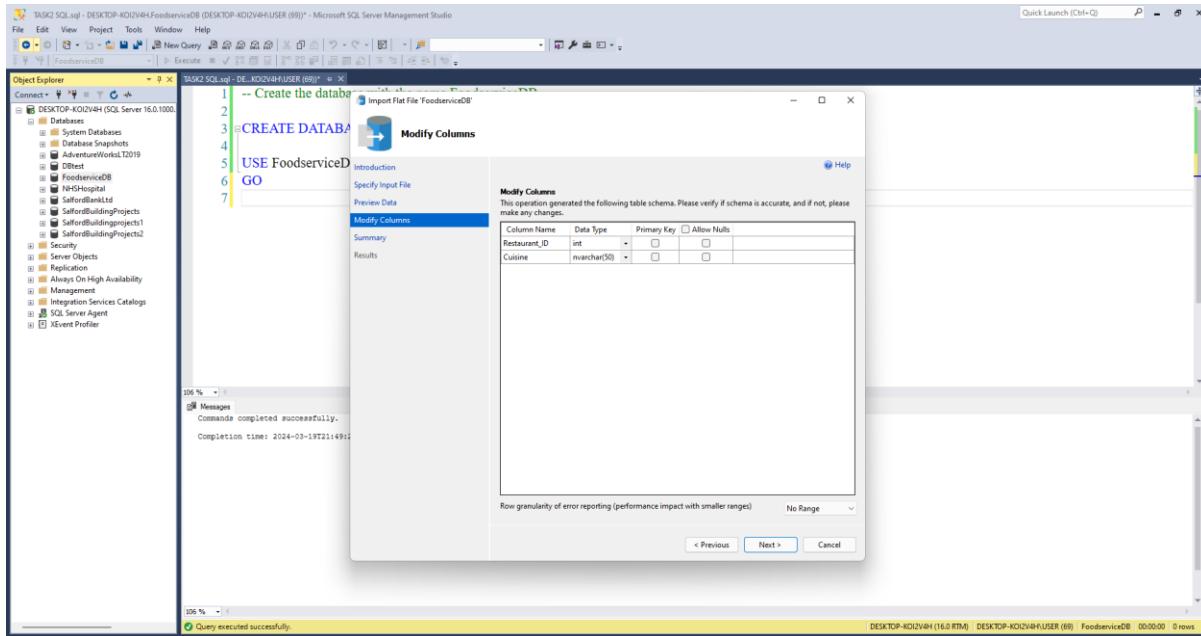
**Figure 2.1.4:** I imported the Consumers file into a table and selected the Consumer\_ID as primary key



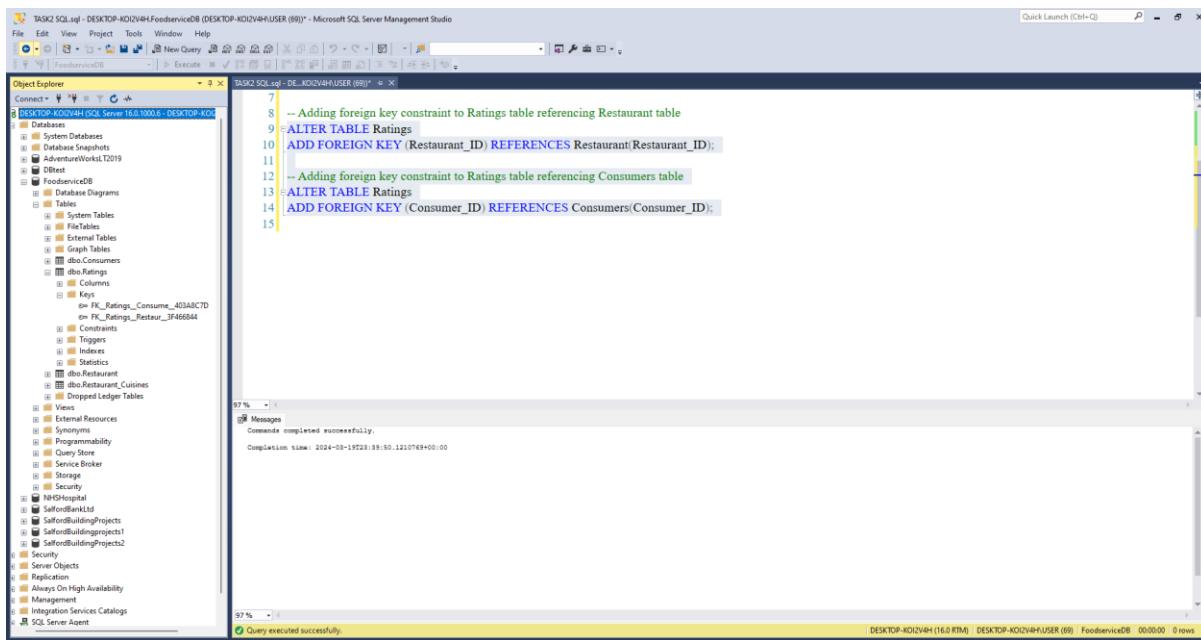
**Figure 2.1.5** I imported the restaurant file into a table and as done in **Figure 2.1.3**, I rewrote the table name and then selected the Restaurant\_ID as primary key.



**Figure 2.1.6** I imported the ratings file into a table and as done in **Figure 2.1.3**, I rewrote the table name to Ratings.



**Figure 2.1.7** I imported the restaurant\_cuisines file into a table and as done in **Figure 2.1.3**, I rewrote the table name to Restaurant\_Cuisines.



**Figure 2.1.8** Adding the foreign key constraint for the Ratings table

### Explain in details element

```

File Edit View Query Project Tools Window Help
File Edit View Query Project Tools Window Help
New Query Execute
15 --Adding a primary key to Ratings table using two foreign keys
16 ALTER TABLE Ratings
17 ADD CONSTRAINT PK_Consumer_Restaurant PRIMARY KEY (Consumer_ID, Restaurant_ID)
18

```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including the FoodserviceDB. The central pane displays the T-SQL code being executed:

```

--Adding a primary key to Ratings table using two foreign keys
ALTER TABLE Ratings
ADD CONSTRAINT PK_Consumer_Restaurant PRIMARY KEY (Consumer_ID, Restaurant_ID)

```

The status bar at the bottom indicates "Query executed successfully." and shows the completion time as 2024-03-20T00:18:33.298867+00:00.

**Figure 2.1.9:** The purpose of the **ALTER** query here is to use the `Consumer_ID` and `Restaurant_ID` columns as composite keys to add a primary key constraint to the `Ratings` table. According to this method, every combination of `Restaurant_ID` and `Consumer_ID` ought to identify a record in the `Ratings` table in a distinct way.

```

File Edit View Query Project Tools Window Help
File Edit View Query Project Tools Window Help
New Query Execute
19
20 -- Add Restaurant_Cuisines_ID to Restaurant_Cuisines table and assign primary key
21 ALTER TABLE Restaurant_Cuisines
22 ADD Restaurant_Cuisines_ID int IDENTITY(1000,1) PRIMARY KEY;

```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including the FoodserviceDB. The central pane displays the T-SQL code being executed:

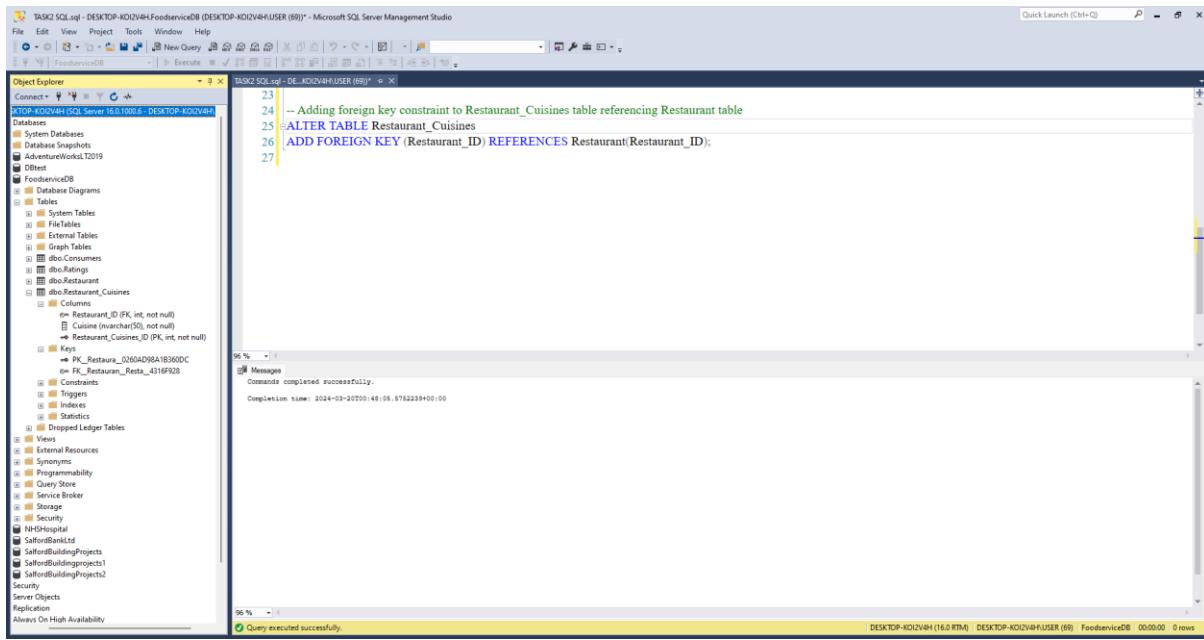
```

-- Add Restaurant_Cuisines_ID to Restaurant_Cuisines table and assign primary key
ALTER TABLE Restaurant_Cuisines
ADD Restaurant_Cuisines_ID int IDENTITY(1000,1) PRIMARY KEY;

```

The status bar at the bottom indicates "Query executed successfully." and shows the completion time as 2024-03-20T00:37:54.8143994+00:00.

**Figure 2.2.0:** Adding a new column (`Restaurant_Cuisines_ID`) with a primary key into the `Restaurant_Cuisines` table



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like `Restaurant`, `Consumers`, `Employees`, and `Restaurant_Cuisines`. The `Restaurant_Cuisines` table is selected. The main pane displays T-SQL code for creating a primary key constraint:

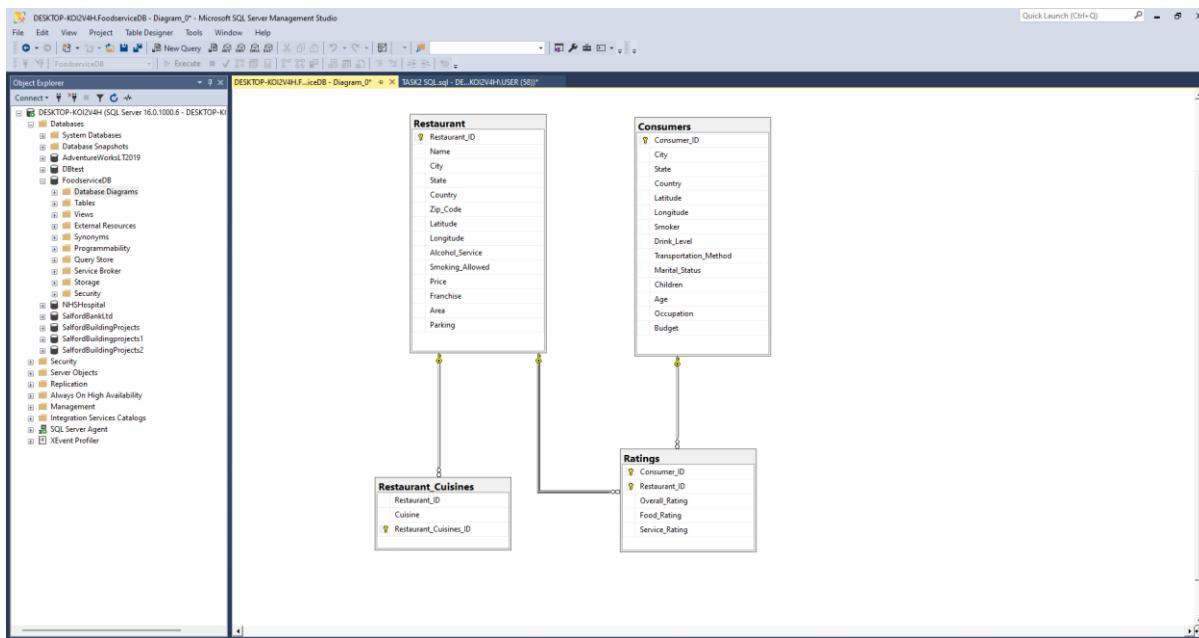
```
23 -- Adding foreign key constraint to Restaurant_Cuisines table
24 ALTER TABLE Restaurant_Cuisines
25 ADD FOREIGN KEY (Restaurant_ID) REFERENCES Restaurant(Restaurant_ID);
26
27
```

The status bar at the bottom indicates "Query executed successfully."

**Figure 2.2.1:** Adding foreign key constraint for the `Restaurant_Cuisines` table to reference the `Restaurant` table.

## Database Diagram

All tables contained in this database have been adequately joined with the required constraints such as foreign keys which has been used to reference or link together related tables so as to query accurate result. Below is the database diagram **Figure 2.2.2**, showing the visual representation of each table and their relation.



**Figure 2.2.2:** The Microsoft SQL Server Management Studio database diagram above shows the tables, their names, primary keys and the relationships between them.

Each table has a unique identifier (**Restaurant\_id** as primary key, **Consumer\_id** as primary key for **Restaurant** and **Consumer** tables respectively, '**Restaurant\_id** and **Consumer\_id** as **Foreign keys** and used to create **Composite** primary key in **Ratings** table, **Restaurant\_Cuisines\_id** as primary key to distinguish individual records within the **Restaurant\_cuisines** table.

### Restaurant Table:

One-to-many relationship with the "Restaurant\_Cuisines" table based on **Restaurant\_id**.

One-to-many relationship with the "Ratings" table based on Restaurant\_id.

**Consumers Table:**

One-to-many relationship with the "Ratings" table based on Consumer\_id.

**Ratings Table:**

Many-to-one relationship with the "Restaurant" table based on Restaurant\_id.

Many-to-one relationship with the "Consumers" table based on Consumer\_id.

**Restaurant\_Cuisines Table:**

One-to-many relationship with the "Restaurant" table based on Restaurant\_id.

## Part 2:

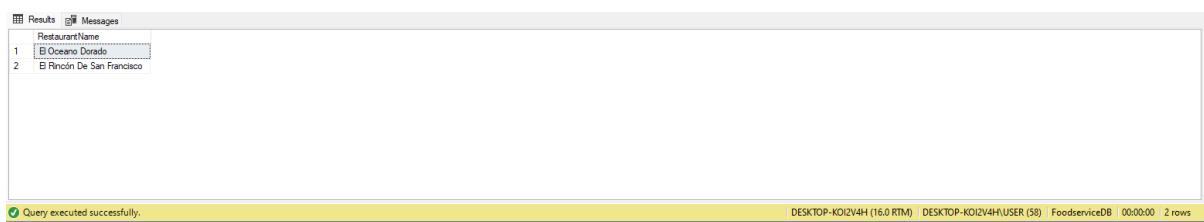
### Answer to Question 1.

1. In order to retrieve or list all restaurants with a medium range price, serving Mexican food in an open area, firstly I have identified the area, and price to be columns under the restaurants table while the Mexican food under the cuisine column is found in the restaurant\_cuisines table. In this case the restaurant table is joined with the restaurant\_cuisines table.



```
TASK2 SQL.sql - DE...KOI2V4H\USER (58)* 28 --No 1. list all restaurant with medium range price with open area, serving Mexican food
29
30 SELECT r.Name AS RestaurantName
31 FROM Restaurant r
32 JOIN Restaurant_Cuisines c
33 ON r.Restaurant_ID = c.Restaurant_ID
34 WHERE r.Price = 'Medium'
35 AND r.Area = 'Open'
36 AND c.Cuisine = 'Mexican'
37
```

**Figure 2.2.3:** The TSQL to list all restaurants with a medium range price, serving Mexican food in an open area.



RestaurantName
El Oceano Dorado
El Rincon De San Francisco

Query executed successfully.

**Figure 2.2.4:** Result gotten from the query in **Figure 2.2.3**

### Answer to Question 2.

2. In Comparing the results of the total number of restaurants who have the overall rating as 1 serving Italian food with Mexican food. Using count Distinct keyword to count the number of restaurants since a restaurant can have ratings multiple times from consumers. This time the restaurants, ratings and restaurant\_cuisines tables are joined together and the where clause will be used to filter the joined result set.

```

37
38 --> No 2 Compare two results
39 -- 2.1 Query that returns the total number of restaurants who have the overall rating as 1 and are serving Mexican food
40
41 =SELECT COUNT(DISTINCT r.Restaurant_ID) Total_Mexican_Restaurants_Rating1
42 FROM Restaurant r
43 JOIN Ratings ra ON r.Restaurant_ID = ra.Restaurant_ID
44 JOIN Restaurant_Cuisines c ON r.Restaurant_ID = c.Restaurant_ID
45 WHERE ra.Overall_rating = 1
46 AND c.Cuisine = 'Mexican';
47

```

**Figure 2.2.5:** This TSQL query shows the total number of restaurants serving Mexican food with overall rating of 1

Total_Mexican_Restaurants_Rating1
27

Query executed successfully.

**Figure 2.2.6:** Result gotten from the query in **Figure 2.2.5**

```

47
48 --> No 2 Compare two results
49 -- 2.2 Query that returns the total number of restaurants who have the overall rating as 1 and are serving Italian food
50
51 =SELECT COUNT(DISTINCT r.Restaurant_ID) Total_Italian_Restaurants_Rating1
52 FROM Restaurant r
53 JOIN Ratings ra ON r.Restaurant_id = ra.Restaurant_id
54 JOIN Restaurant_Cuisines c ON r.Restaurant_id = c.Restaurant_id
55 WHERE ra.Overall_rating = 1
56 AND c.Cuisine = 'Italian';
57

```

**Figure 2.2.7:** This TSQL query shows the total number of restaurants serving Mexican food with overall rating of 1

Total_Italian_Restaurants_Rating1
4

Query executed successfully.

**Figure 2.2.8:** Result gotten from the query in **Figure 2.2.7**

Total number of restaurants	Overall rating of 1
Mexican food	27
Italian food	4

In restaurants with an overall rating of one, Mexican food is more common than Italian food, based on the comparison. This could indicate that there are just more Mexican eateries in the dataset or that patrons of lower-rated businesses tend to prefer Mexican food. This might be the result of varying consumer expectations, societal factors, or individual preferences. When thinking about menu options, customer satisfaction tactics, and general business decisions, restaurant owners and managers can benefit greatly from an understanding of these dynamics.

In general, the study draws attention to possible distinctions between Italian and Mexican eating experiences with regard to client satisfaction within the dataset. In order to gain a deeper comprehension of the observed discrepancies in ratings, this could spur further investigation into elements such as food quality, service standards, or the preference of a customer to a particular cuisine.

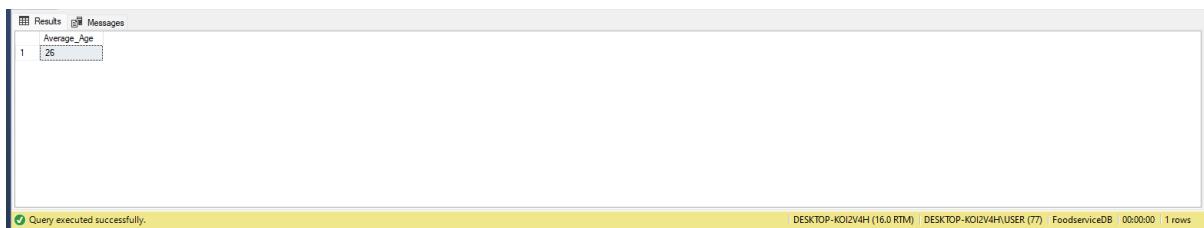
### Answer to Question 3.

3. The average of consumers who have given a service rating of 0 is calculated by utilizing the AVG function and performing a join between the consumers and Ratings tables



```
TASK2 SQL.sql - DE_KOIZV4H\USER (77)* 58 --No 3 Calculates the average age of consumers who have given a 0 rating to the 'Service_Rating' column.  
59  
60 :SELECT ROUND(AVG(c.Age), 0) AS Average_Age  
61 :FROM Consumers c  
62 :JOIN Ratings ra ON c.Consumer_id = ra.Consumer_id  
63 :WHERE ra.Service_Rating = 0;  
64 |
```

**Figure 2.2.9:** This TSQL query shows the average age of consumers giving restaurants a service rating of 0.



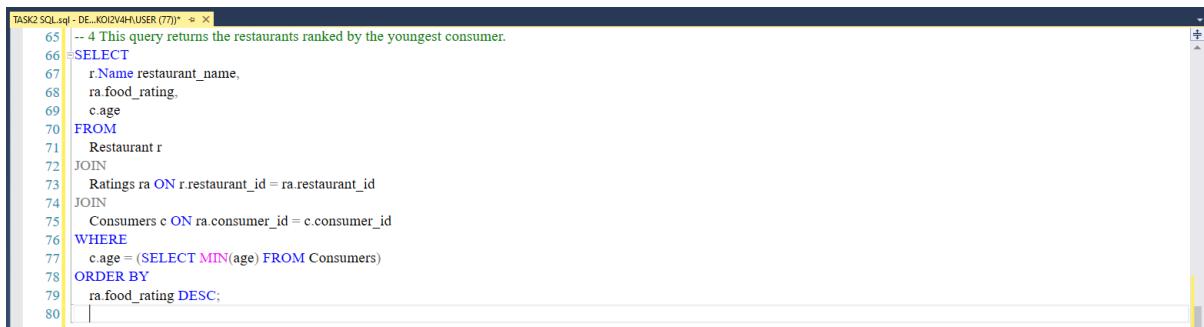
Average_Age
26

Query executed successfully.

**Figure 2.3.0:** Result from the executed query in **Figure 2.2.9**

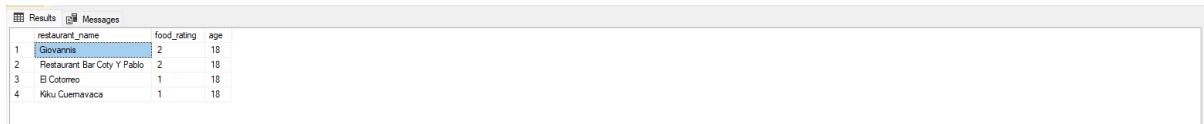
#### Answer to Question 4.

4. Having a composite primary key constraint to the Ratings table with the columns Consumer\_ID and Restaurant\_ID which are foreign keys corresponding the primary keys in the Consumer and Restaurant tables, the join conditions is used to first join the Restaurant table with the Ratings table and then the Ratings table with the Consumer table, after which we filter the consumers who are the youngest by using a subquery in the WHERE clause to find the minimum age, then we sort the results based on the food rating in the descending order.



```
TASK2 SQL.sql - DE_KO2V4H\USER (77)*  ✘ ×
65  -- 4 This query returns the restaurants ranked by the youngest consumer.
66  SELECT
67    r.Name restaurant_name,
68    ra.food_rating,
69    c.age
70  FROM
71    Restaurant r
72  JOIN
73    Ratings ra ON r.restaurant_id = ra.restaurant_id
74  JOIN
75    Consumers c ON ra.consumer_id = c.consumer_id
76  WHERE
77    c.age = (SELECT MIN(age) FROM Consumers)
78  ORDER BY
79    ra.food_rating DESC;
80
```

**Figure 2.3.1:** This query returns the restaurants ranked by the youngest consumer

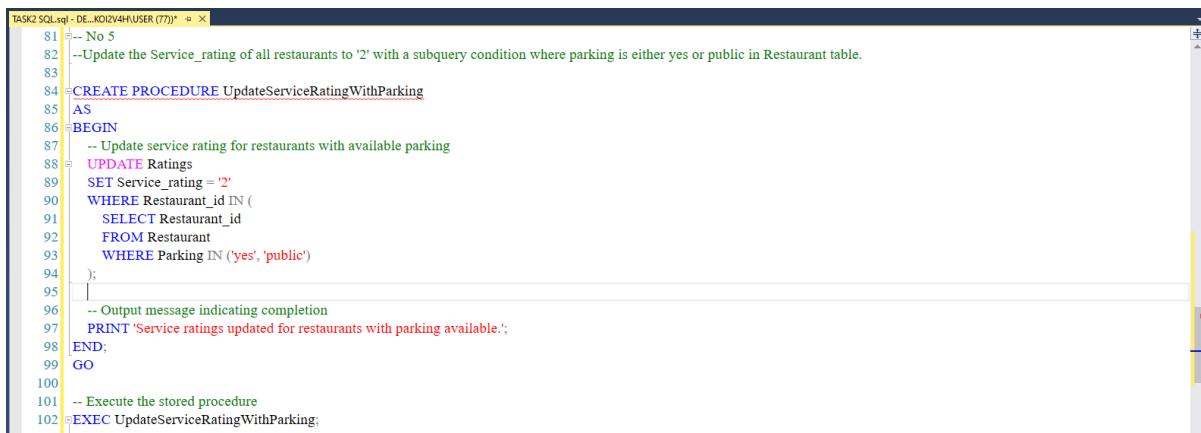


	restaurant_name	food_rating	age
1	Giovannis	2	18
2	Restaurant Bar Coto Y Pablo	2	18
3	El Cotero	1	18
4	Kiku Cuemavaca	1	18

**Figure 2.3.2:** Result from the query in **Figure 2.3.1**

## Answer to Question 5.

5. The stored procedure ‘**UpdateServiceRatingWithParking**’ modifies the ‘**Service\_rating**’ entries in the ‘Ratings’ table. It specifically targets restaurants with available parking, identified by values of ‘yes’ or ‘public’ in the ‘Parking’ column of the ‘Restaurant’ table. The procedure sets the service rating for these restaurants to ‘2’ upon completion. It displays confirming the update of service ratings for restaurants with parking availability.



```
TASK2 SQL.sql - DE_KOIZV4H(USER (77)) * X
81 -- No 5
82 --Update the Service_rating of all restaurants to '2' with a subquery condition where parking is either yes or public in Restaurant table.
83
84 CREATE PROCEDURE UpdateServiceRatingWithParking
85 AS
86 BEGIN
87    -- Update service rating for restaurants with available parking
88    UPDATE Ratings
89    SET Service_rating = '2'
90    WHERE Restaurant_id IN (
91        SELECT Restaurant_id
92        FROM Restaurant
93        WHERE Parking IN ('yes', 'public')
94    );
95
96    -- Output message indicating completion
97    PRINT 'Service ratings updated for restaurants with parking available.';
98 END;
99 GO
100
101 -- Execute the stored procedure
102 EXEC UpdateServiceRatingWithParking;
```

**Figure 2.3.3:** The TSQL query to write a stored procedure

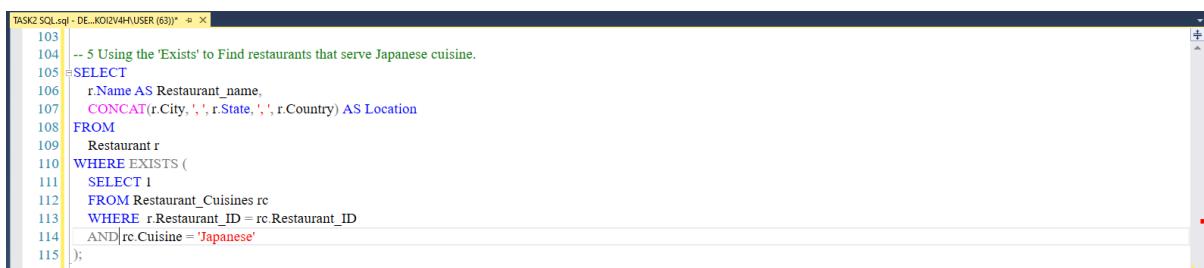


```
Messages
(571 rows affected)
Service ratings updated for restaurants with parking available.
Completion time: 2024-03-21T18:57:26.6753559+00:00
```

**Figure 2.3.4:** The result shows 571 rows affected and the print statement

## Answer to Question 6.

- (i) Using the ‘**EXISTS**’ clause to check for the existence of at least one row in the Restaurant\_Cuisines table where the restaurant serves Japanese cuisine, this was done by matching the Restaurant\_ID column from the outer query with the Restaurant\_ID in the subquery, the outer query selects restaurant names and their location from the Restaurant table



```
103
104 -- 5 Using the 'Exists' to Find restaurants that serve Japanese cuisine.
105 +SELECT
106     r.Name AS Restaurant_name,
107     CONCAT(r.City, ',', r.State, ',', r.Country) AS Location
108 FROM
109     Restaurant r
110 WHERE EXISTS (
111     SELECT 1
112     FROM Restaurant_Cuisines rc
113     WHERE r.Restaurant_ID = rc.Restaurant_ID
114     AND rc.Cuisine = 'Japanese'
115 );
```

**Figure 2.3.5:** This TSQL query finds restaurants that serve Japanese cuisine and returns their name and location.



	RestaurantName	Location
1	Mikasa	Cuernavaca, Morelos, Mexico
2	Shi Ro Ie	San Luis Potosi, San Luis Potosi, Mexico
3	Kiku Cuernavaca	Cuernavaca, Morelos, Mexico
4	Motoko Restaurant Japones	San Luis Potosi, San Luis Potosi, Mexico
5	Sushi Ito	San Luis Potosi, San Luis Potosi, Mexico

**Figure 2.3.6:** Here is the result of the query in **Figure 2.3.5**

- (ii) Using ‘**IN**’ in this case, the innermost subquery selects consumers who are younger than 21 years old from the Consumers table, while the middle subquery matches Ratings table with the innermost query using Consumer\_id to ensure we get ratings made by consumers who are younger than 21. Finally the outer query then selects the names, city and country from the Restaurant table where the Restaurant\_id’s matches in the middle query, to ensure we get restaurants that have been rated by consumers younger than 21

```

117
118 SELECT TOP 10 Name AS RestaurantName, city, country
119 FROM Restaurant
120 WHERE Restaurant_id IN (
121     SELECT Restaurant_id
122     FROM Ratings
123     WHERE Consumer_id IN (
124         SELECT Consumer_id
125         FROM Consumers
126         WHERE Age < 21
127     )
128 );

```

**Figure 2.3.7:** The TSQL query selects the names of restaurants from the restaurant table that have been rated by consumers younger than 21 years old and then the result limited to TOP ten.

	RestaurantName	city	country
1	Rincon Huasteco	San Luis Potosi	Mexico
2	Cafeteria Y Restaurant El Pacifico	San Luis Potosi	Mexico
3	Luna Cafe	San Luis Potosi	Mexico
4	Restaurant Oriental Express	San Luis Potosi	Mexico
5	Pizza Casera	San Luis Potosi	Mexico
6	Restaurante La Estrella De Dina	San Luis Potosi	Mexico
7	El Hacienda Restaurante And Bar	San Luis Potosi	Mexico
8	Restaurante Guerra	San Luis Potosi	Mexico
9	Restaurante Pueblo Bonito	San Luis Potosi	Mexico
10	Tacos Abi	Ciudad Victoria	Mexico

**Figure 2.3.8:** Here is the result of the query in **Figure 2.3.7**

(iii) Using ‘**System Function**’, to find the average food rating of restaurants that offer alcohol service to those that do not

```

134
135
136 --(iii)
137
138 --What is the average food rating of restaurants that offer alcohol service compared to those that do not?
139 = SELECT
140   ISNULL(R.Alcohol_Service, 'No Alcohol Service') AS Alcohol_Service_Status,
141   AVG(RT.Food_Rating) AS Avg_Food_Rating
142   FROM
143   Restaurant R
144   LEFT JOIN
145   Ratings RT ON R.Restaurant_ID = RT.Restaurant_ID
146   GROUP BY
147   R.Alcohol_Service,
148
149
150
151
152
153
154
155
156
157
158
159
160

```

**Figure 2.3.9** This query uses **ISNULL** to determine a restaurant’s average (**AVG**) meal rating by considering factors like alcohol service. It helps restaurant management make informed decisions about menu items by highlighting the relationship between food and drink service ratings.

Alcohol_Service_Status	Avg_Food_Rating
1 Full Bar	1
2 None	1
3 Wine & Beer	1

Query executed successfully.

**Figure 2.4.0** Here is the result of the query in **Figure 2.3.9**

- (iv) Using **Groupby**, **Having** and **OrderBy**, the TSQL query selects Restaurant\_ID and Name which was Alias as RestaurantName from the Restaurant table, uses the AVG() function on the service\_rating column to determine the average service rating and then aliased the column as 'AvgServiceRating' from the Ratings table. The query then performs a JOIN operation on Restaurant and Ratings tables, groups the result by Restaurant\_ID and Name to calculate the average service rating for each restaurant. Subsequently, it used the HAVING clause to filter out restaurants where the average service rating is less than 3. Finally, the result is ordered by the average service rating in descending order.

```

147 -- To find the avg service rating for each restaurant where avg service rating is less than 3
148 SELECT r.Restaurant_ID, r.Name AS RestaurantName,
149     AVG(rt.service_rating) AS AvgServiceRating
150 FROM Restaurant r
151 JOIN
152 Ratings rt ON r.Restaurant_ID = rt.Restaurant_ID
153 GROUP BY
154 r.Restaurant_ID, r.Name
155 HAVING
156 AVG(rt.service_rating) < 3
157 ORDER BY
158 AvgServiceRating DESC

```

**Figure 2.4.1:** The TSQL query to return Restaurant\_ID and Name with AvgServiceRating less than 3 and ordered by the average rating I descending order.

Results Messages

	Restaurant_ID	RestaurantName	AvgServiceRating
1	132584	Gorditas Dona Tota	2
2	132594	Tacos De Barbacoa Enfrente Del Tec	2
3	132603	Hamburguesas La Penca	2
4	132609	Pollo Frito Buenos Aires	2
5	132613	Cemitas Mata	2
6	132626	La Penca Hamburguesa	2
7	132650	Puesto de Gorditas	2
8	132572	Cafe Chaine	2
9	132705	Gorditas Dona Tota	2
10	132717	Tortas Hawaii	2
11	132723	Gorditas De Morales	2
12	132755	La Estrella De Dinas	2
13	132765	Mikasa	2
14	132767	Restaurant Familiar B Chino	2
15	132733	Little Caesar's	2
16	132834	Gorditas Dona Gloria	2

Query executed successfully.

DESKTOP-KOIZV4H (16.0 RTM) DESKTOP-KOIZV4H\USER (68) FoodserviceDB 00:00:00 | 130 rows

**Figure 2.4.2:** Here is the result from the query in **Figure 2.4.1**

## CONCLUSION

In today's data-fast moving corporate environment, businesses need to analyze huge amounts of information using SQL queries to identify patterns, trends, and areas for improvement. Automating data management can be a cost-effective and efficient solution, as stored procedures ensure consistency and consistency in data protocols. For instance, a stored process can update service ratings based on parking availability.

**Decision Support:** Making wise choices is crucial to succeed in the cutthroat restaurant business. Decision-makers can gain important information from SQL queries into a range of restaurant operations topics, such as menu planning, marketing campaigns, and operational improvement. For instance, restaurants can find-selling items, ascertain which menu items are failing, and modify their menu offers by employing SQL queries to analyze sales data.

**Data Integrity:** Ensuring the quality, consistency, and dependability of database information depends on maintaining data integrity. In order to maintain referential integrity between related tables and lower mistakes and inconsistencies in the database, primary and foreign key constraints are essential. A foreign key constraint, on the other hand, creates a relationship between tables and stops the insertion of erroneous data.

To sum up, stored procedures and SQL queries are essential for organizing and evaluating restaurant data, which helps companies create strategic plans and make wise judgments.