Hope Hospital Database

Introduction

The task is to develop a relational database for Hope hospital. The database is for the storing of information on the patients, doctors, medical records (diagnosis, medicines, medicine prescribed date, allergies), appointments, and departments. The hospital requested for a system that has the following functionalities:

- Patient registration
- Appointment booking/cancellation
- Review/feedback system for completed appointments
- Retain inactive patient's data

Microsoft SQL Server and SSMS were used to implement T-SQL statements and associated database objects in the development of the database.

Part 1

The first step was to design the database which was done using the hybrid approach and normalized to the 3NF (removal of transitive dependencies). This ensured that there are no redundancies (which could cause anomalies, memory wastage and data inconsistency) in the database.

The process undergone to design the database are:

1. Identification of Entities:
    - Patients
    - Doctors
    - Appointments
    - Departments
    - Medical Records
2. Identification of the relationships between the entities:
    - Patient - Doctor: many-to-many
    - Patient - Appointment: one-to-many
    - Patient - Medical record: one-to-one
    - Doctor -Department: one-to-one
    - Department - Doctor: one-to-many
    - Doctor - Appointment: one-to-many
3. The Entity Relationship Diagram (ERD) was drawn to foster better visualization of the design.

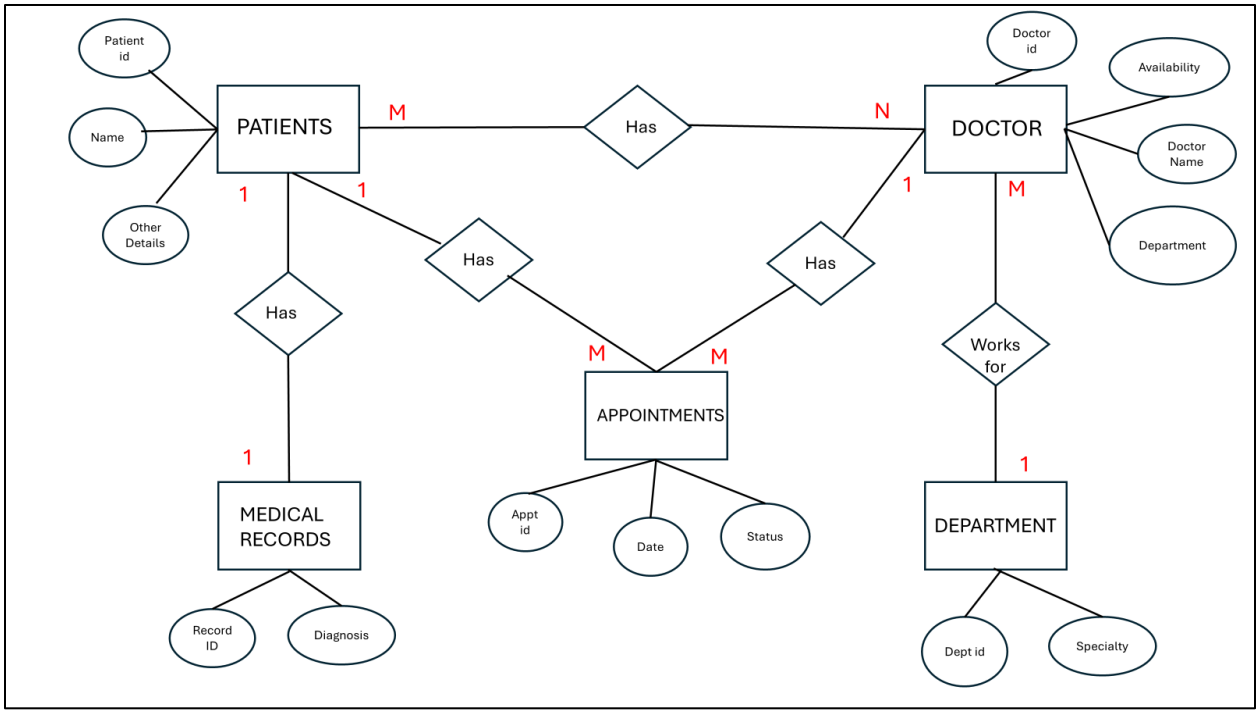

Figure 1: Entity Relationship Diagram for HopeHospital database (only some attributes shown)

4. Splitting of Entities into tables based on the ERD while considering normalization.
    - Patients: Patient, ArchivePatients
    - Doctors: Doctor, Available
    - Appointment: Appointment, PastAppointment
    - Department: Department
    - Medical Records: Diagnosis

The next step was to create the database, tables, and database objects to meet the client's requirements.

The steps followed in the creation of the 3NF database are outlined below.

1. The "HopeHospital" database was created using the CREATE DATABASE statement.

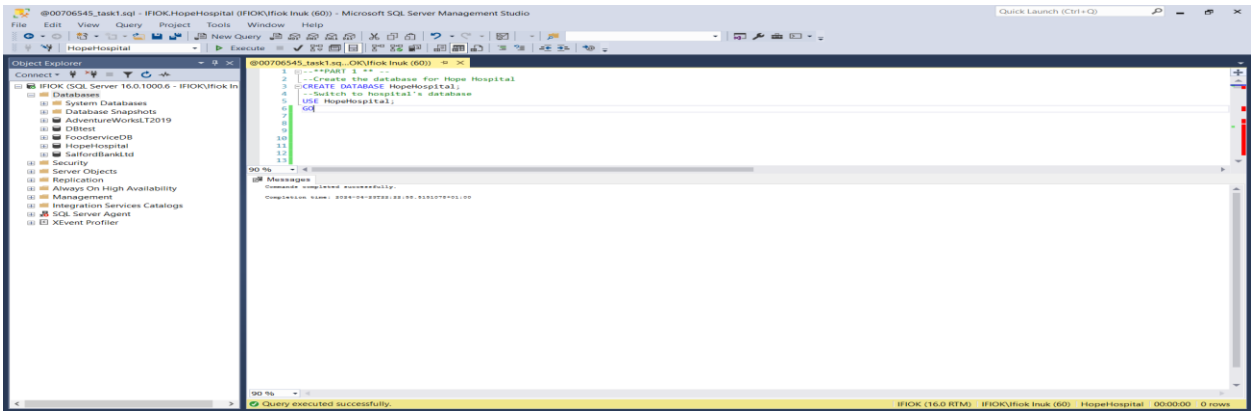

Figure 2: Creation of the database

2. Eight tables required for the database were created. The columns, datatype and justifications for each table are stated below.

Patients: contains all the personal details for the patients that are registered in the hospital.

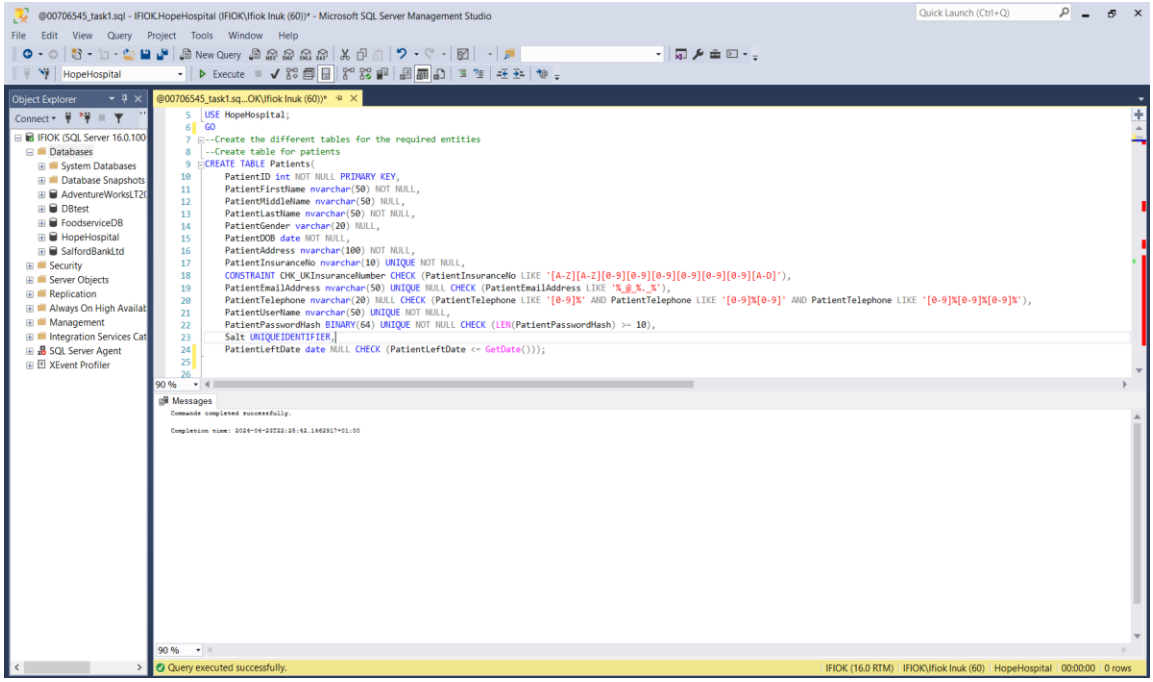

Figure 3: Creation of the Patients table

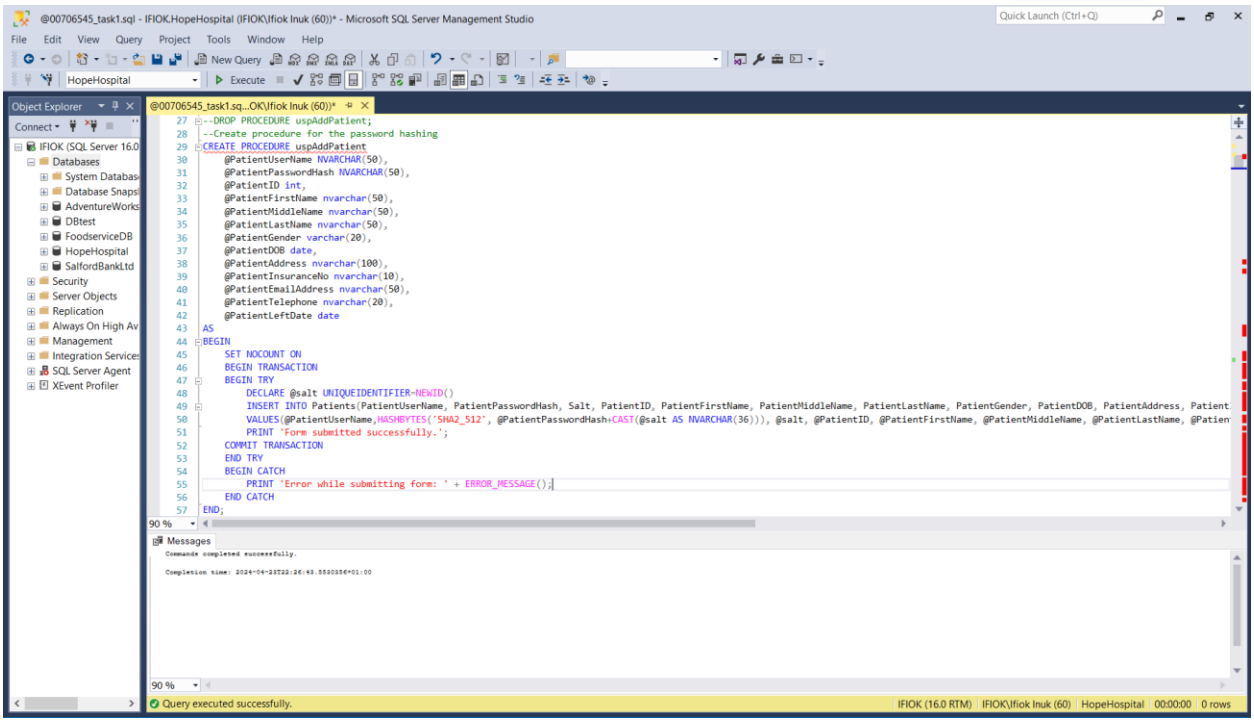

Figure 4: Creation of the uspAddPatient procedure

ArchivePatients: stores details of the patients that have left the hospital. This table is similar to the Patients table. It is populated by the execution of two triggers explained below.

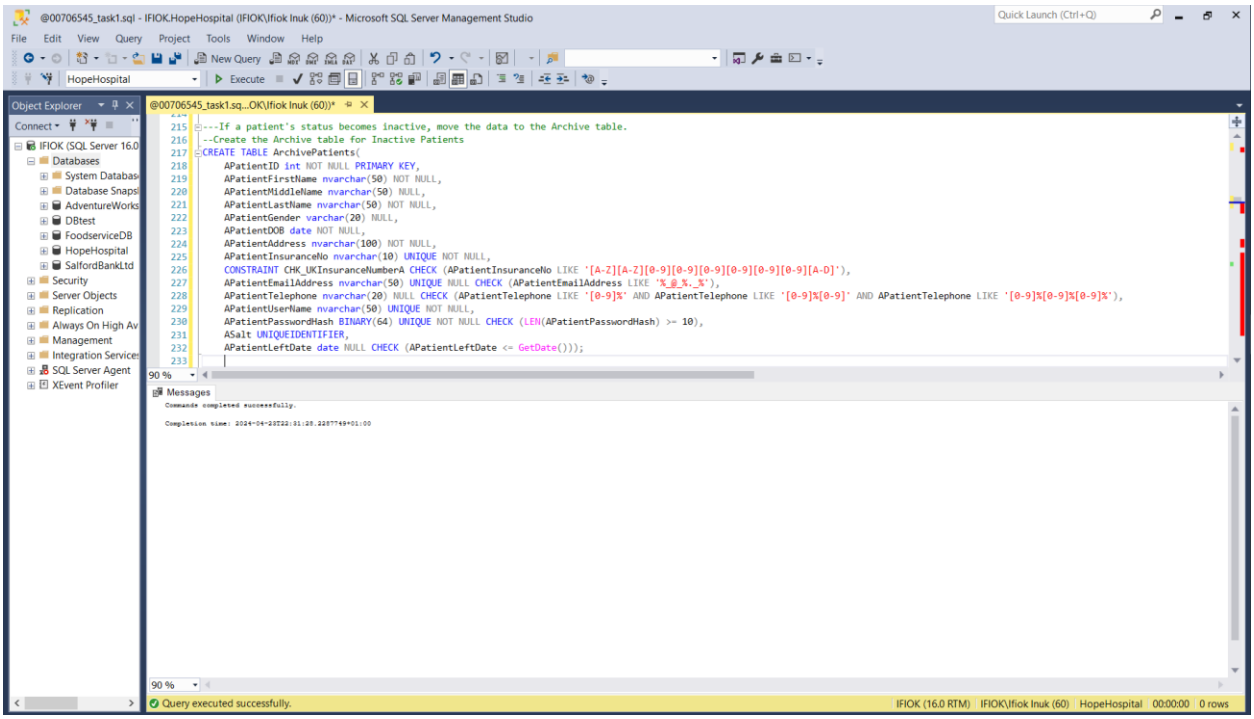

Figure 5: Creation of the ArchivePatients table

A trigger patient_update_status was created to delete the record once the PatientLeftDate (Patients table) is not null.

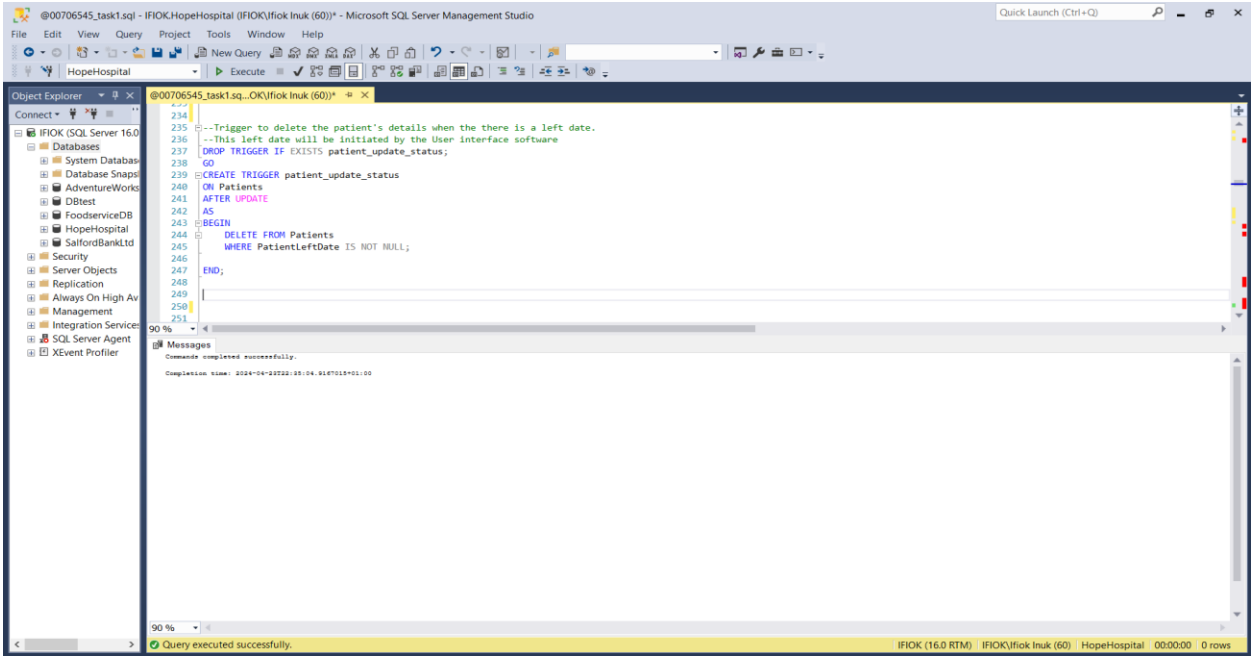

Figure 6: Creation of the patient_update_status trigger

Another trigger patient_left_archive was created to insert the deleted record into the ArchivePatients table.

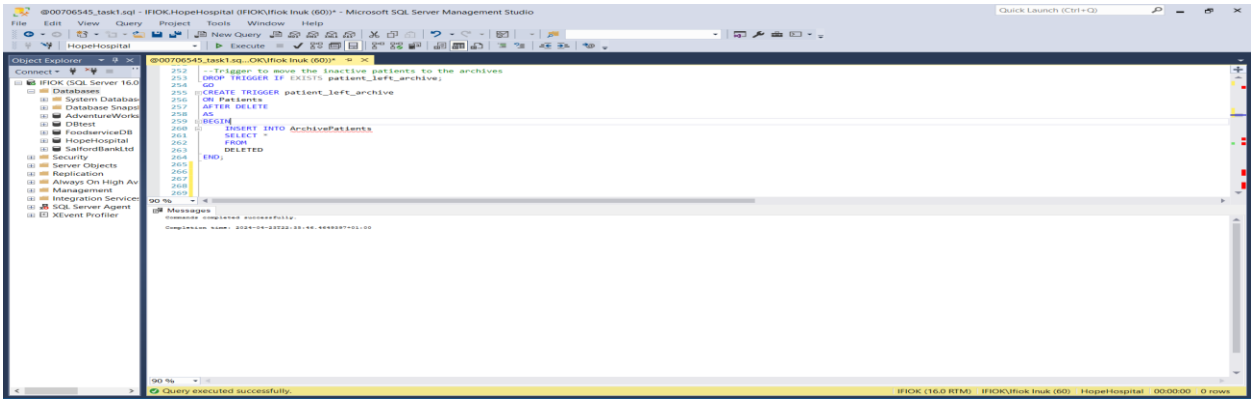

Figure 7:Creation of the patient_left_archive trigger

To confirm the transfer of records, the query below was executed.

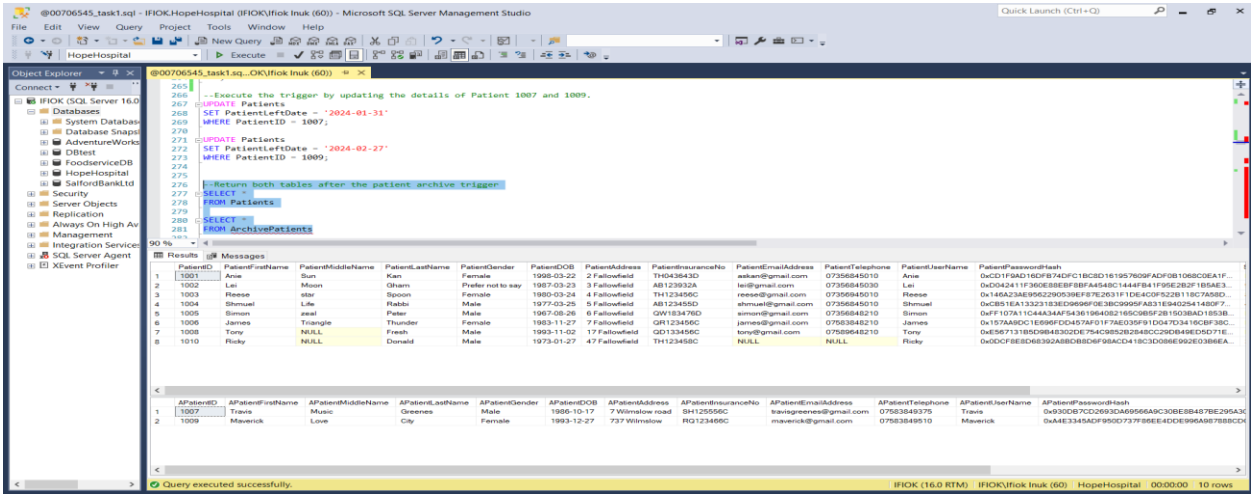

Figure 8: Results for archiving process

Department: contains the details for departments in the hospital. Columns are:

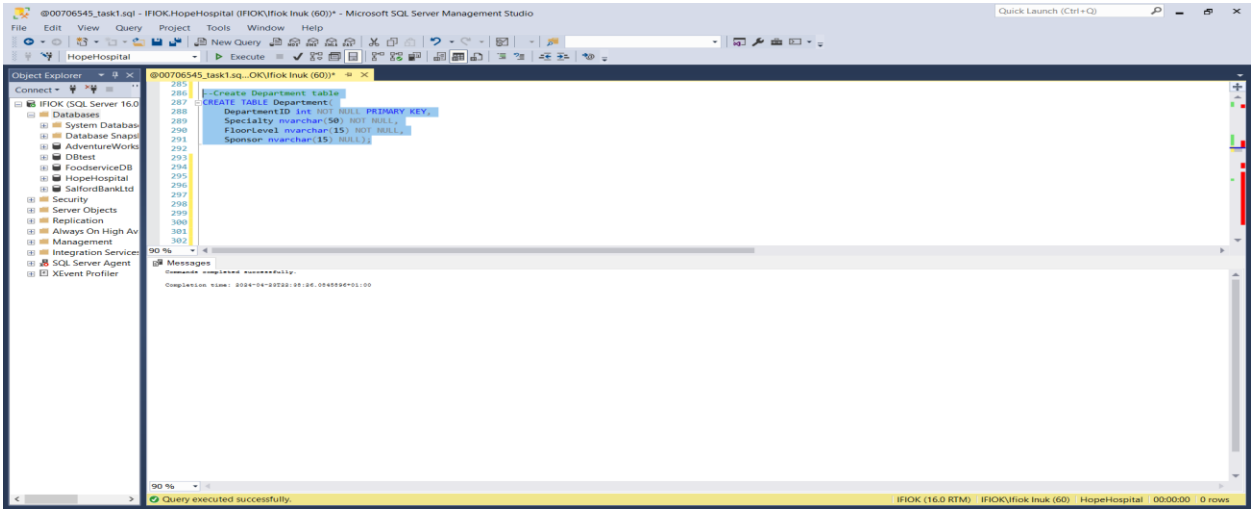

Figure 9: Creation of the Department table

Doctor: This table contains the attributes for the doctors in the hospital.

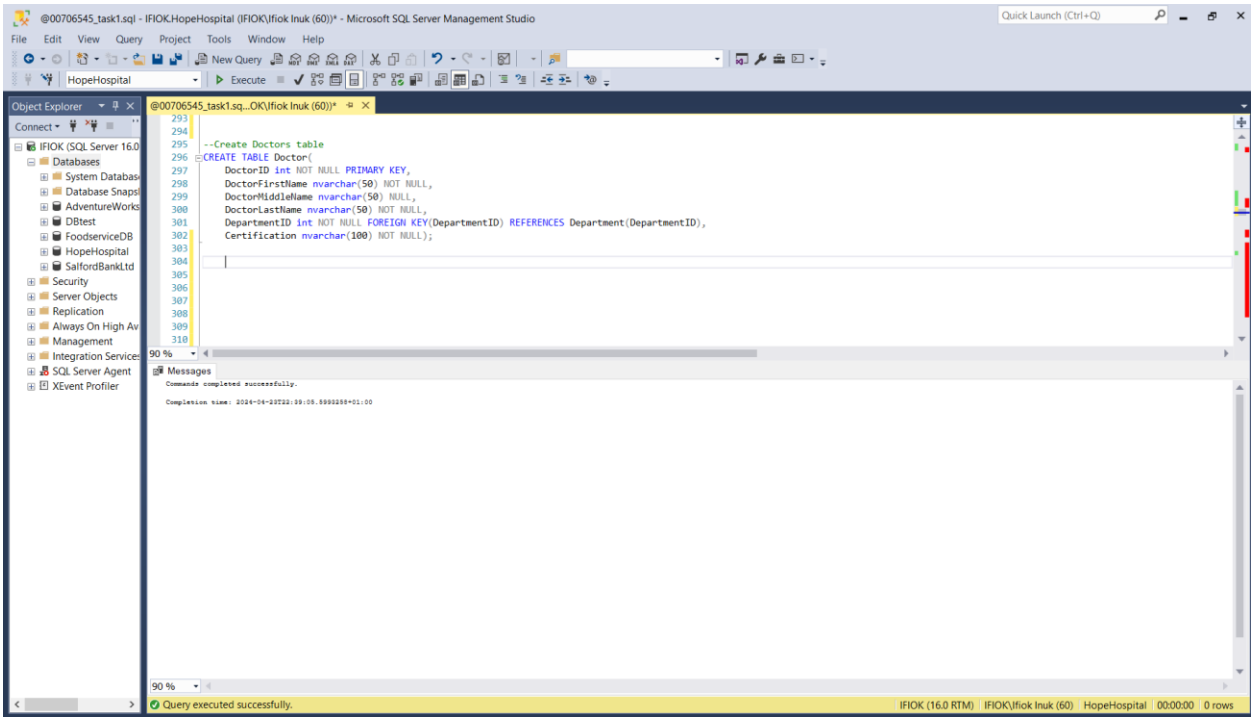

Figure 10: Figure 10: Creation of the Doctor table

Available: This table stores the doctors' availability.

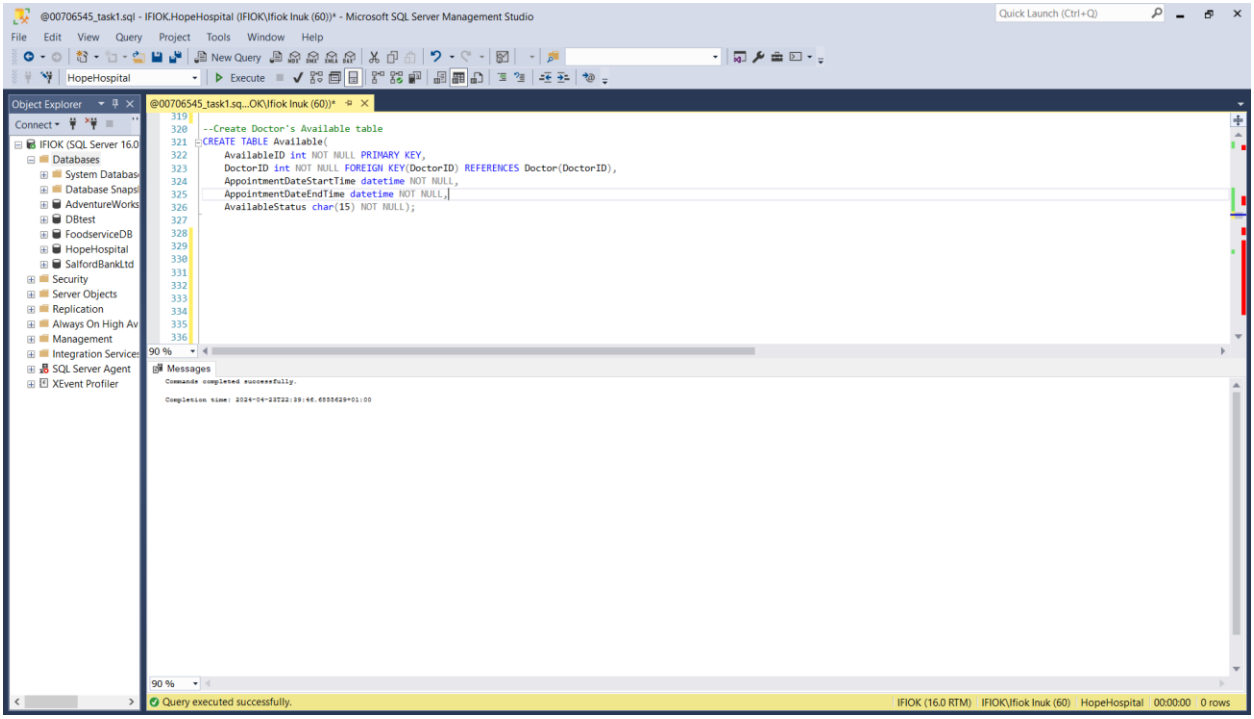

Figure 11:Creation of the Available table

Diagnosis: This contains the diagnosis details for each appointment that a patient has completed.

i. .

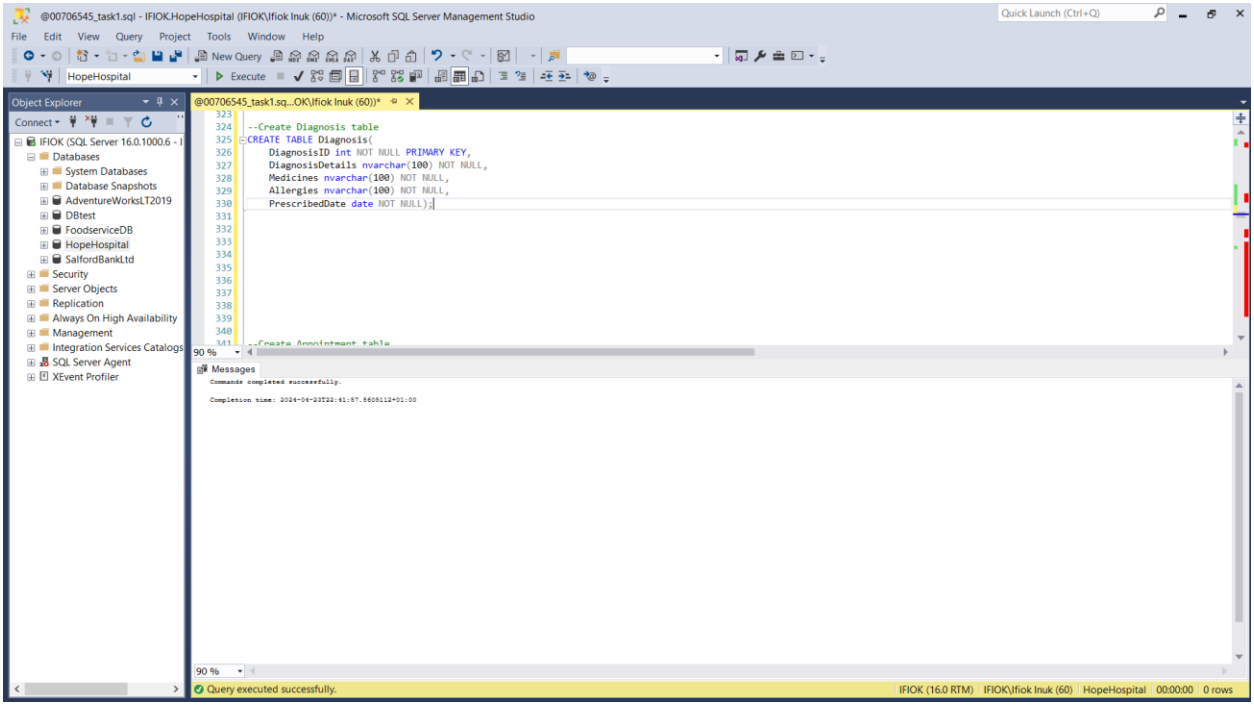

Figure 12: Creation of the Diagnosis table

Appointment: This stores the booked appointments for all the patients.

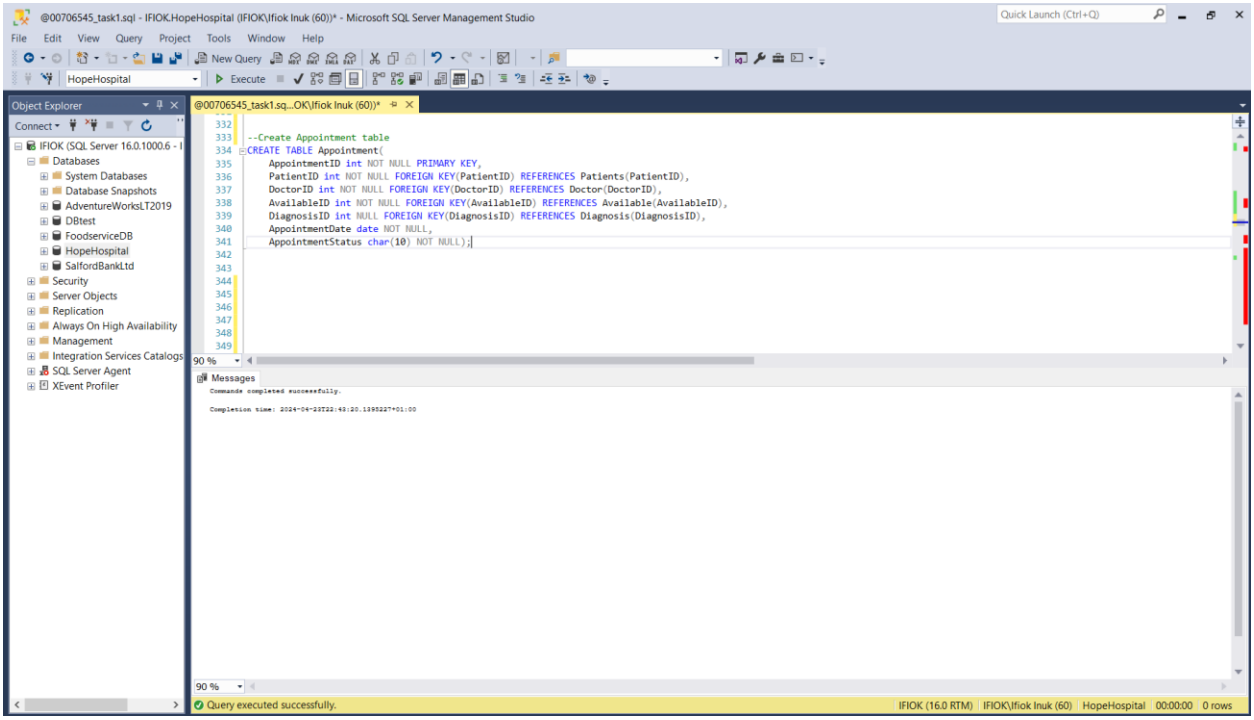

Figure 13: Creation of the Appointment table

PastAppointment: This table archives the completed appointments. It has the same columns as the Appointment table except for the last column that is replaced with the FeedbackDetails column. This column is filled once the patient provides feedback. The data type for the column is nvarchar(100) and has a **NULL** constraint.

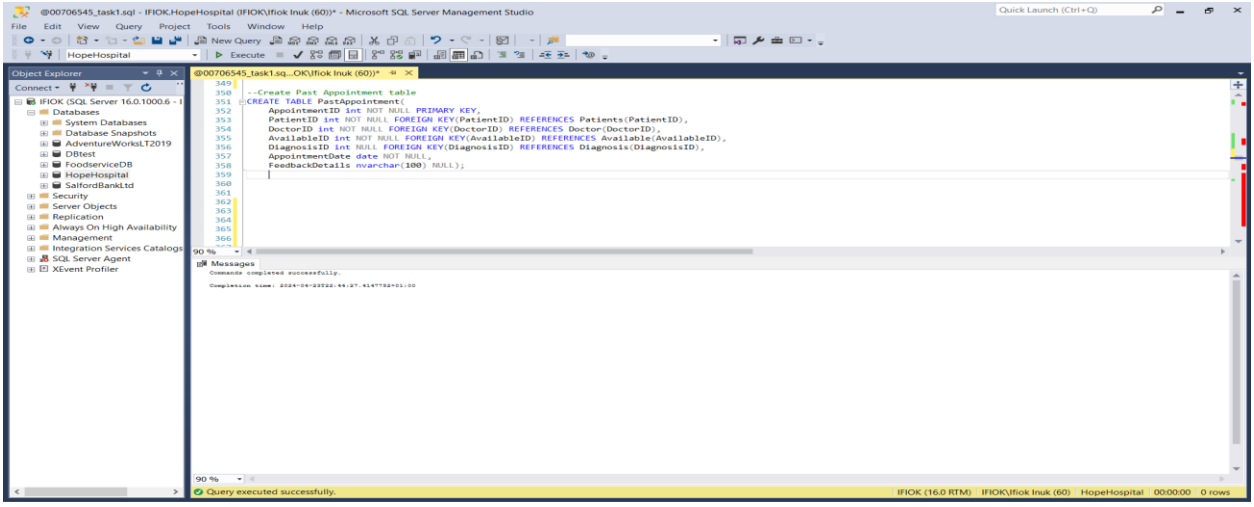

Figure 14:Creation of the PastAppointment table

The tables were populated using the INSERT INTO statement with test data.

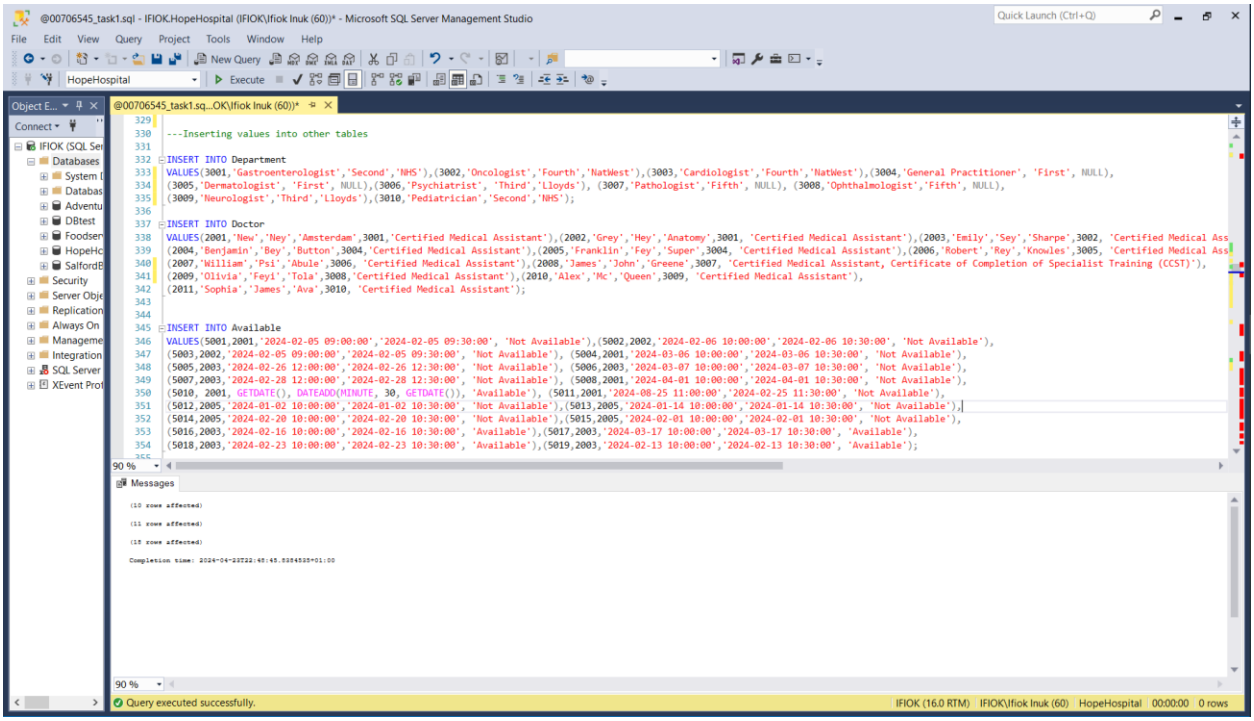

Figure 15: Manual insertion of data into tables(1)

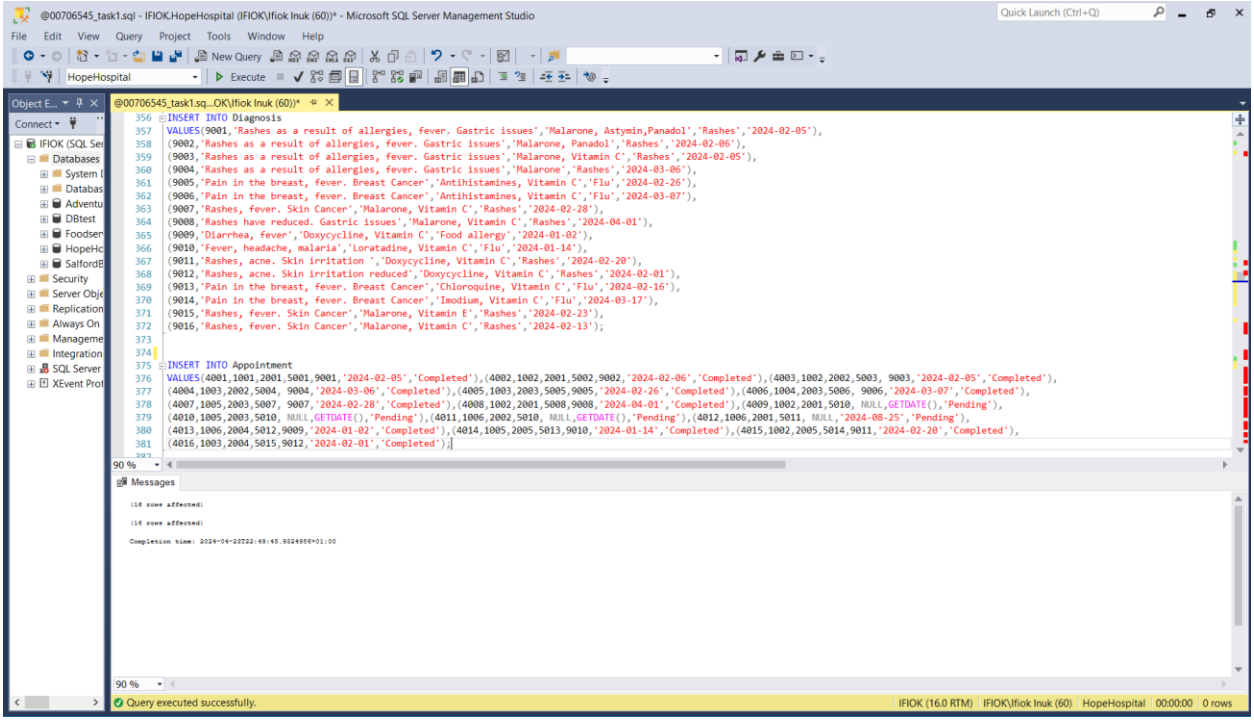

Figure 16: Manual insertion of data into tables(2)

A stored procedure-uspBookAppointment was created to simulate a new patient's booking and system check the doctor's availability.

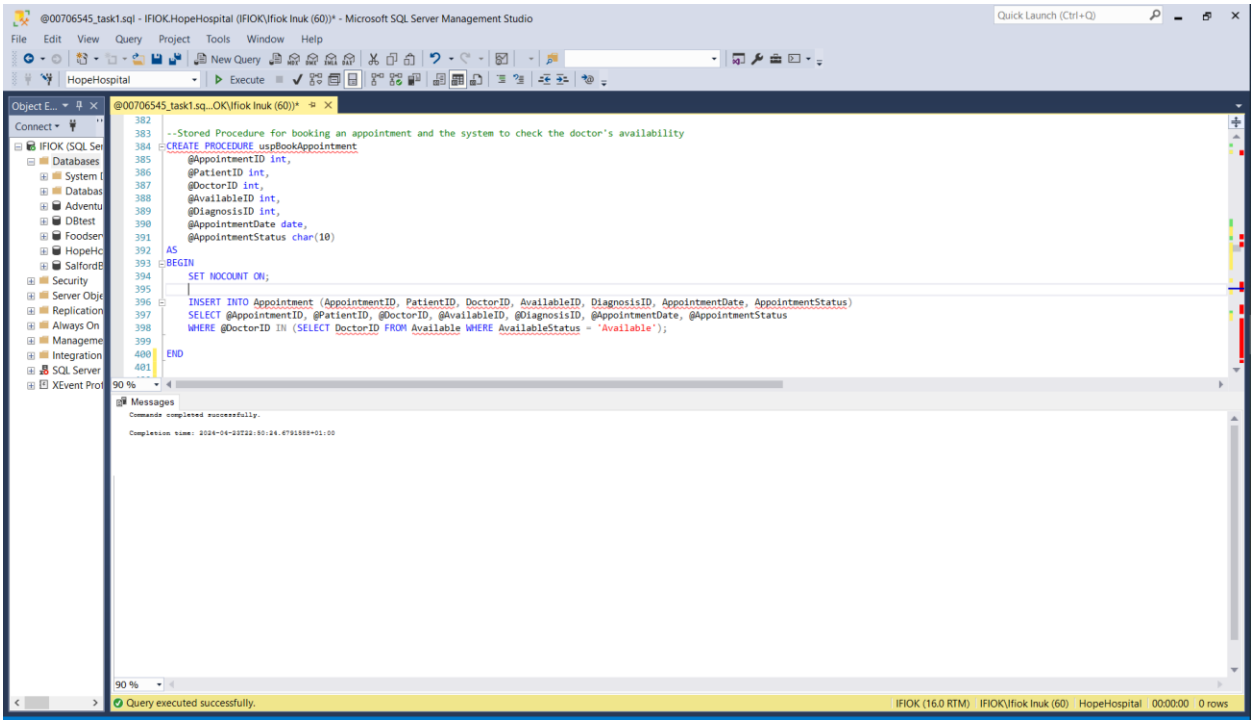

Figure 17: Creation of uspBookAppointment procedure

The procedure was executed.

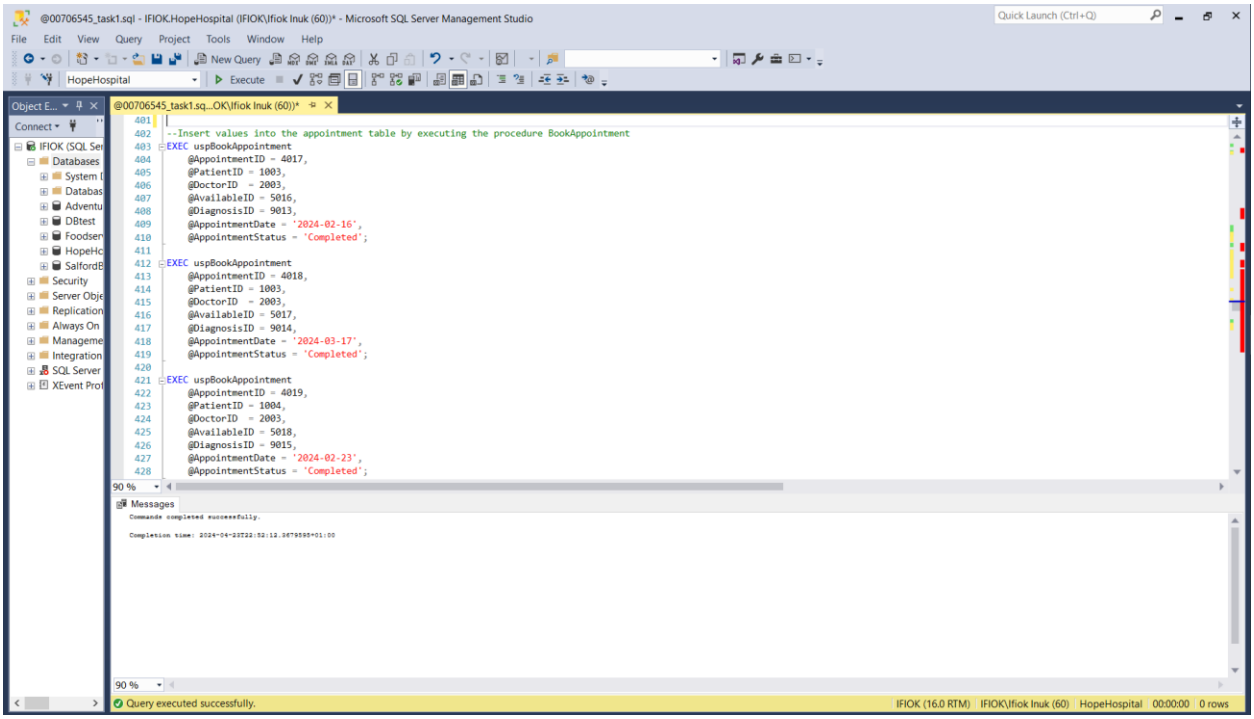

Figure 18: Execution of uspBookAppointment procedure

A SELECT statement was used to return the populated tables. See created tables and the database diagram showing the different relationships below.

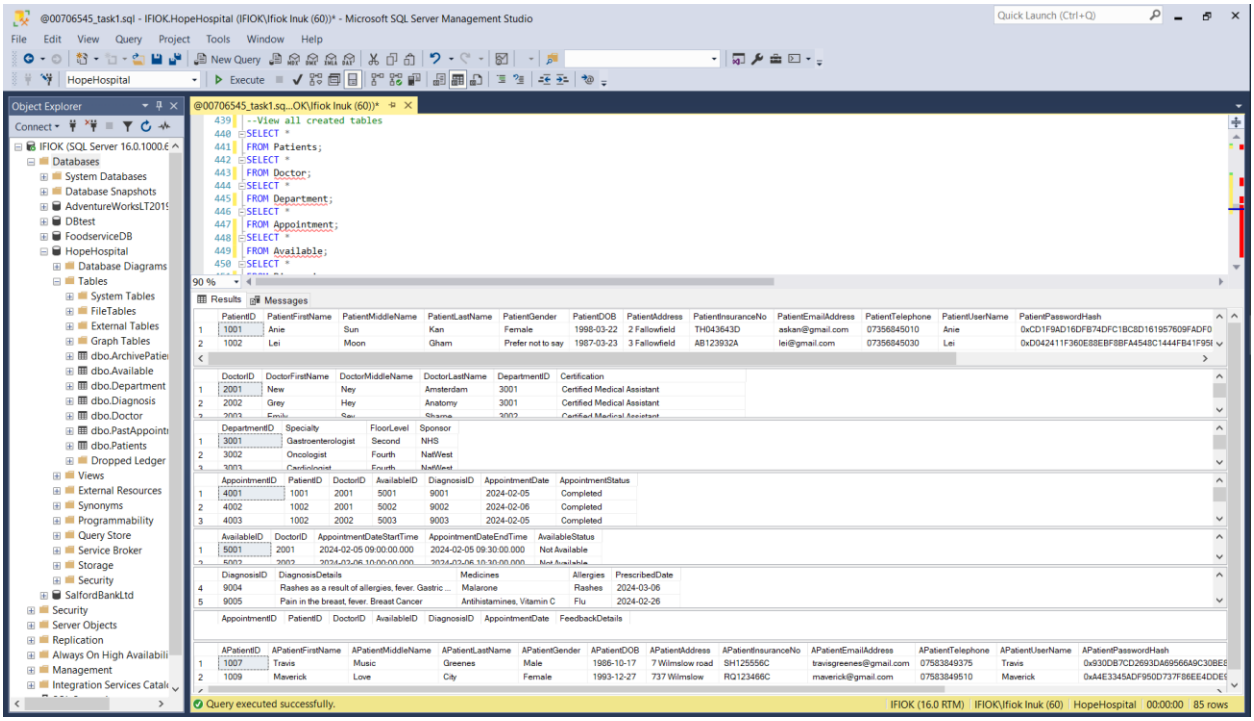

Figure 19: Populated tables

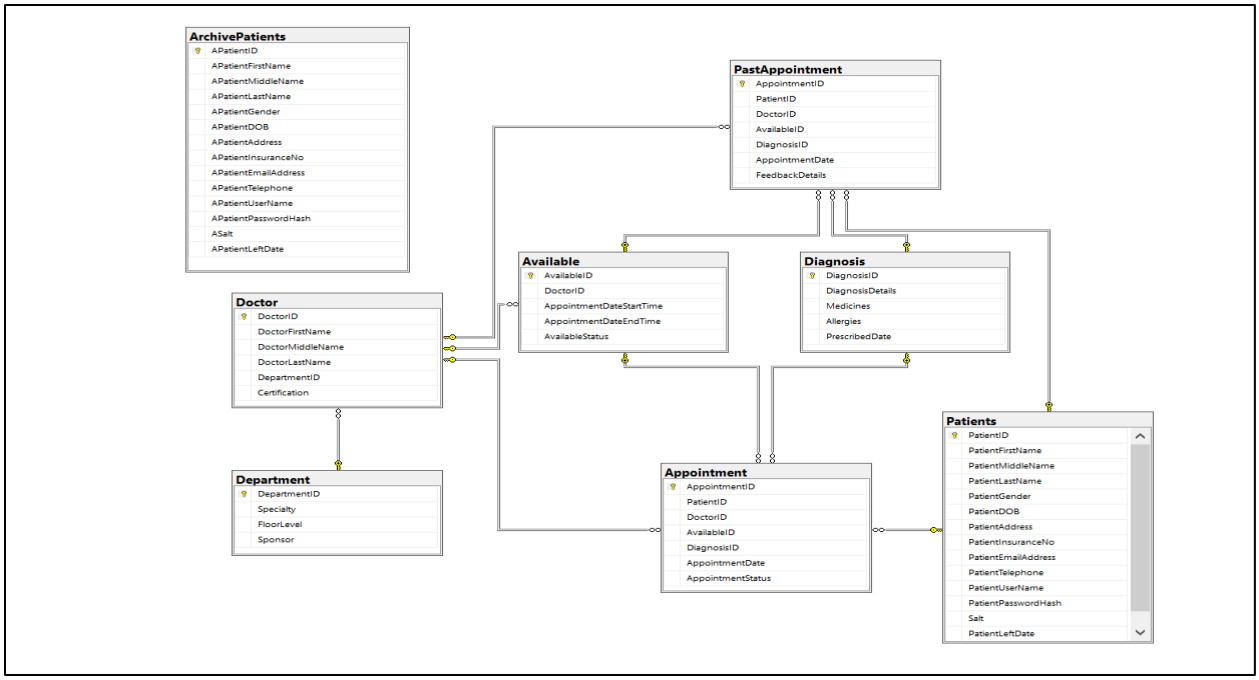

Figure 20: Database diagram for Hope Hospital

Part 2

<u>Question 2: Constraint to check that the appointment date is not in the past.</u>

The ALTER TABLE statement was used ADD the CONSTRAINT APPT_CURRENT which checks that the value in the AppointmentDate column is greater than or equal to the system time (GETDATE()),meaning now or in the future.

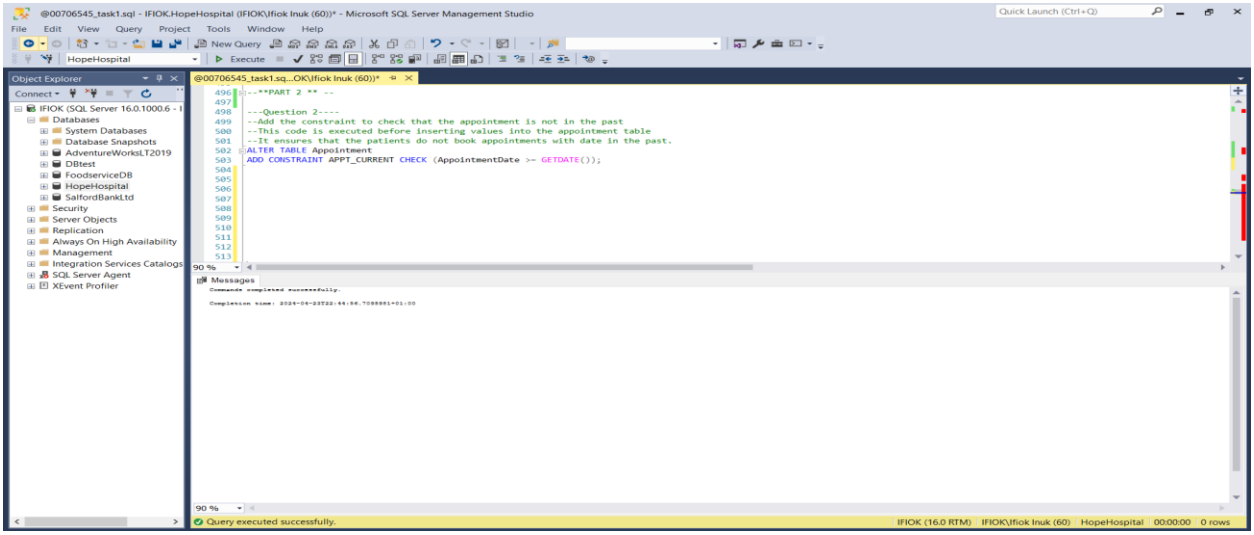

Figure 21: Constraint to check that the AppointmentDate is not in the past

It is important to note that Question 3, 4a and 4b were executed after the stored procedure in question 4d had been executed. Question 4d populates the PastAppointment table and deletes completed appointments from the Appointment table.

Question 3: Patients older than 40 and have Cancer in diagnosis.

The result shows that there are 7 patients over the age of 40 that have cancer in their diagnosis.

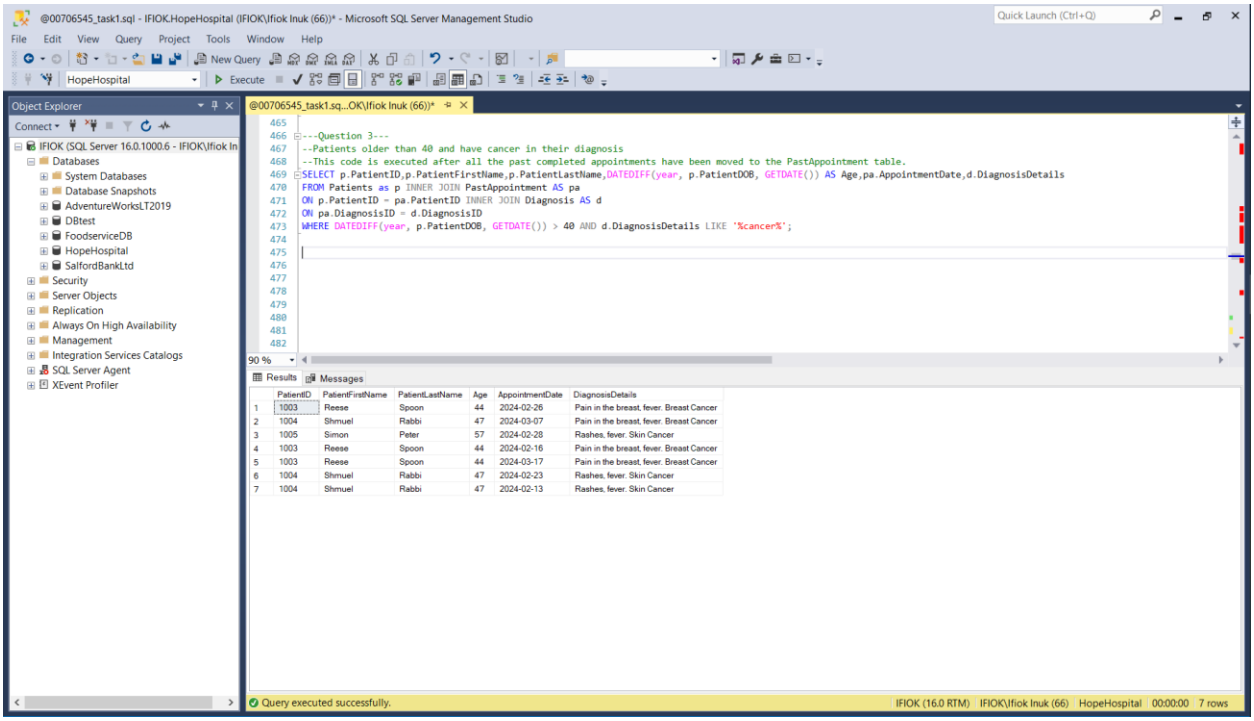

Figure 22: List of patients older than 40 and have cancer

Question 4a: Search the database by Medicine name.

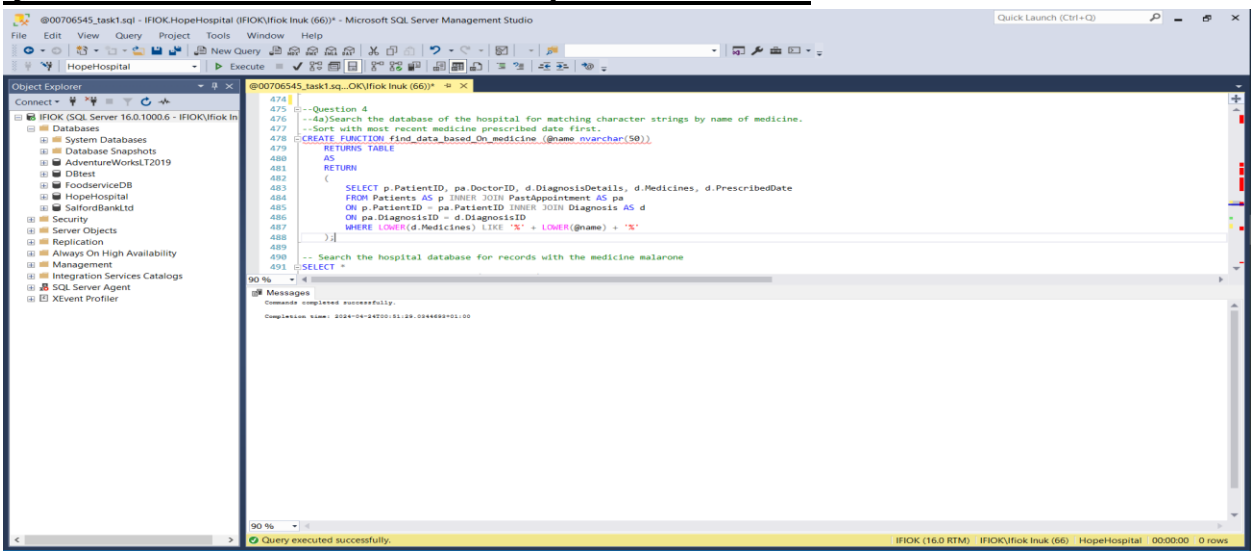

Figure 23: Creation of the find_data_based_On_medicine function

The result shows that we have 8 records with the medicine name malarone, and the most recent prescription was for a patient with id 1002.

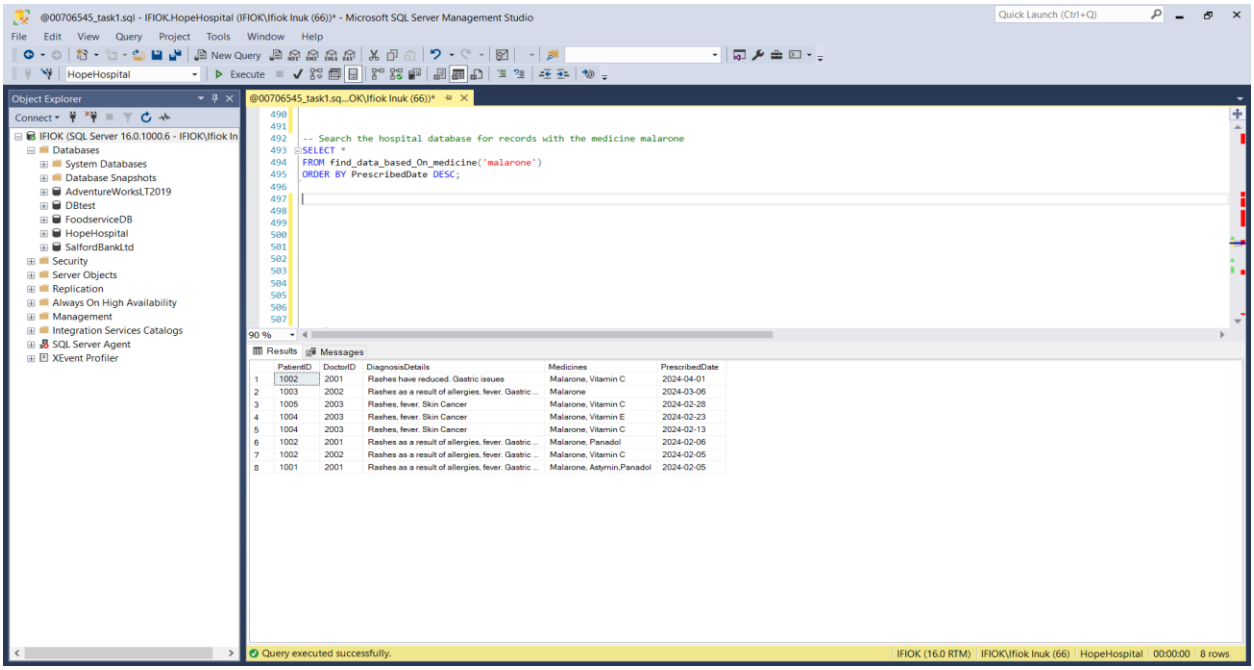

Figure 24: Execution of the function

<u>Question 4b: Full list of diagnosis and allergies for a patient that has an appointment today</u>

A stored procedure patient_medical_record with a parameter @Patientid was created. A SELECT statement was used to return specific columns from the resulting table.

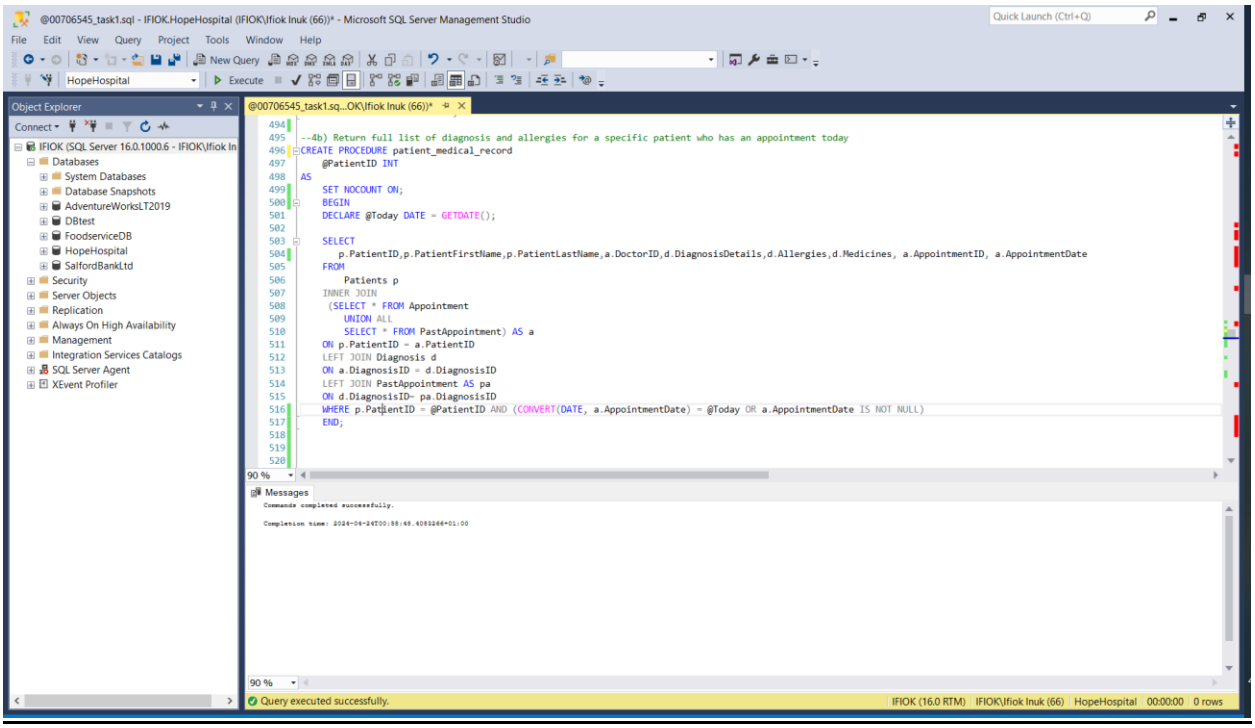

Figure 25: Creation of patient_medical_record procedure

The result shows the list of the patients (id:1002) past appointment, diagnosis and today's appointment (2024-04-23).

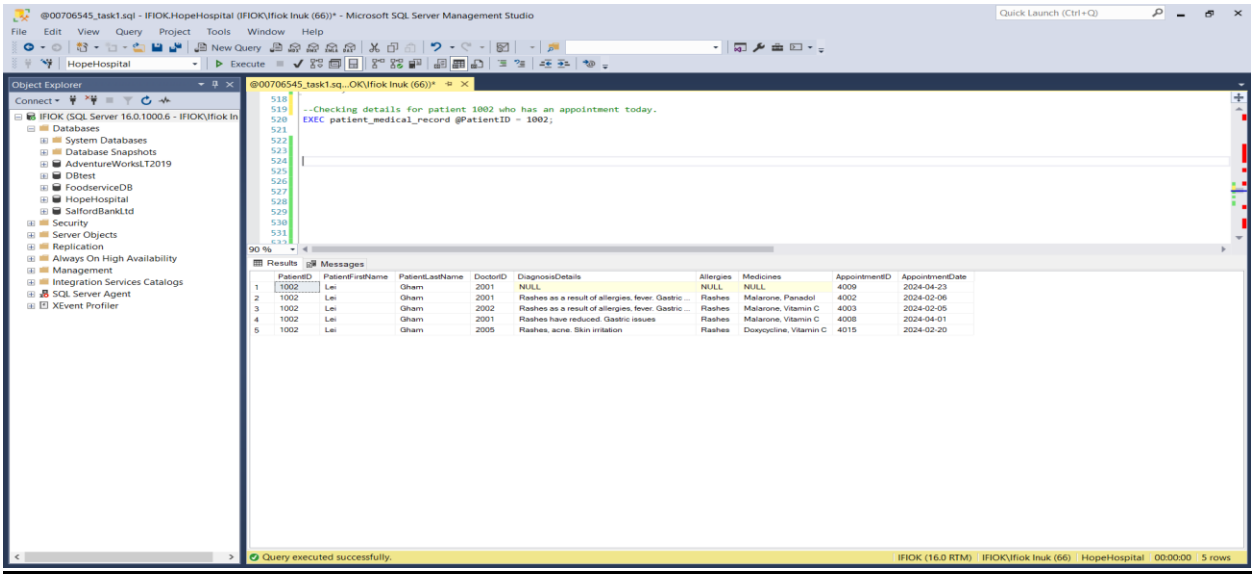

Figure 26: Full list of diagnosis and allergies for a patient 1002

Question 4c: Update the details for an existing doctor

This updates the certifications of the doctors when required. A stored procedure update_doctor with parameters @doctorid and @cert was created.

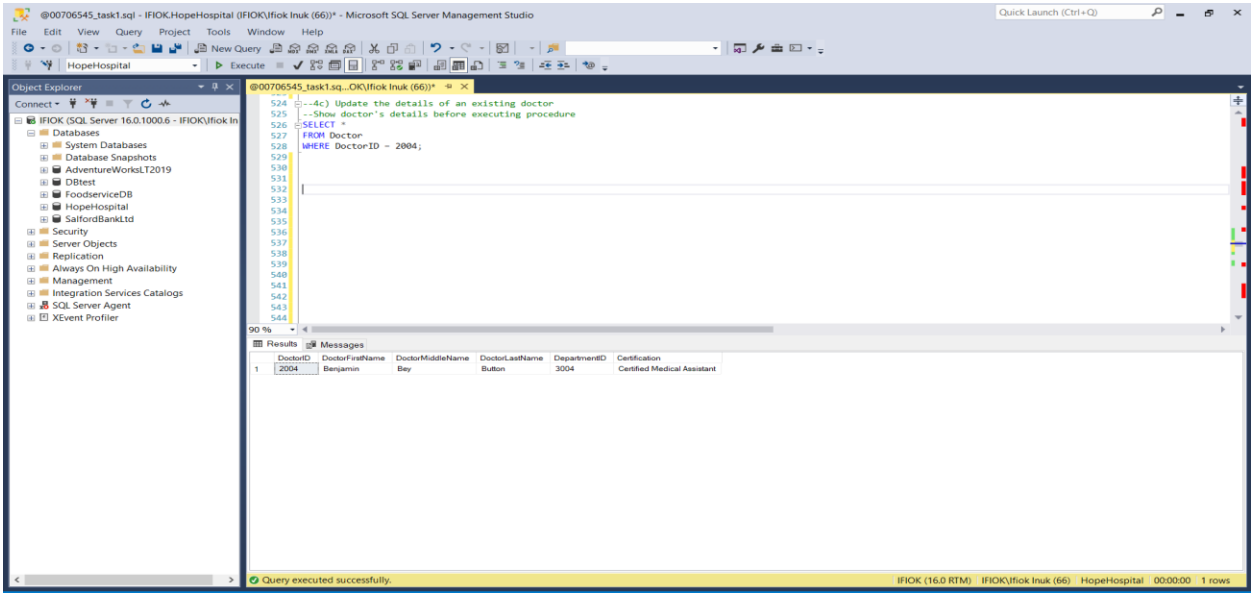

Figure 27: Test query before executing update_doctor

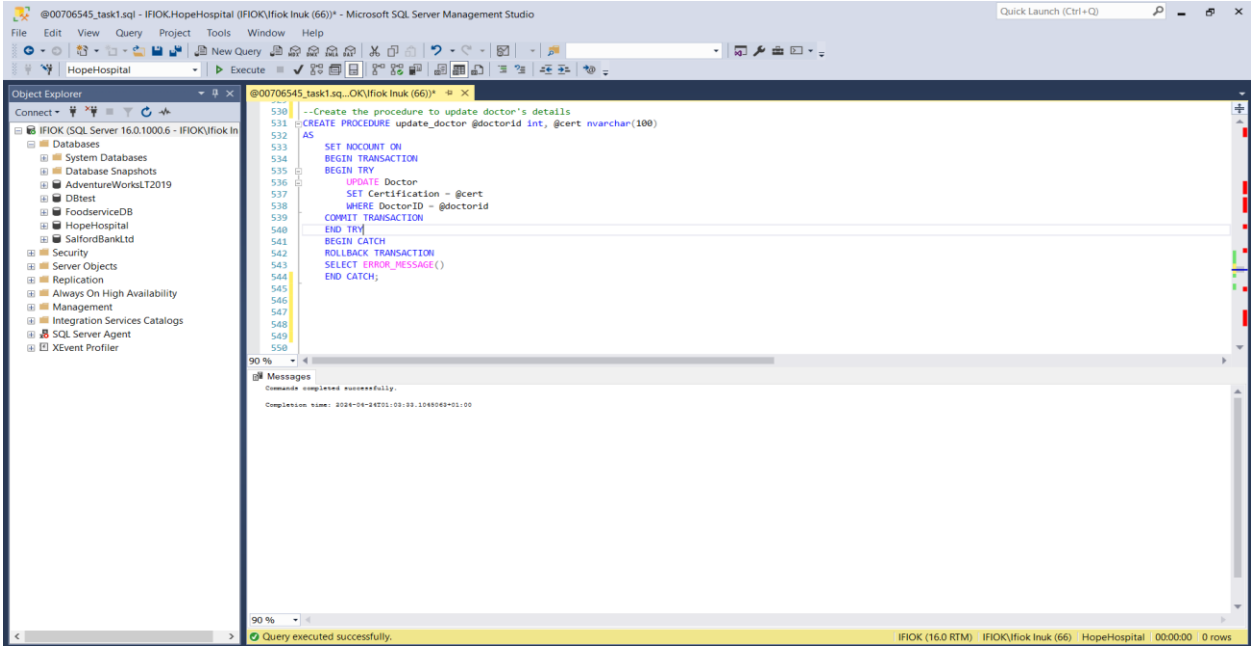

Figure 28: Creation of update_doctor

EXEC was used to execute the statement on the doctor with id 2004 and certification 'Certified Medical Assistant,Certificate of Completion of Specialist Training (CCST)'. A SELECT query was run before and after the procedure execution to confirm the successful update.

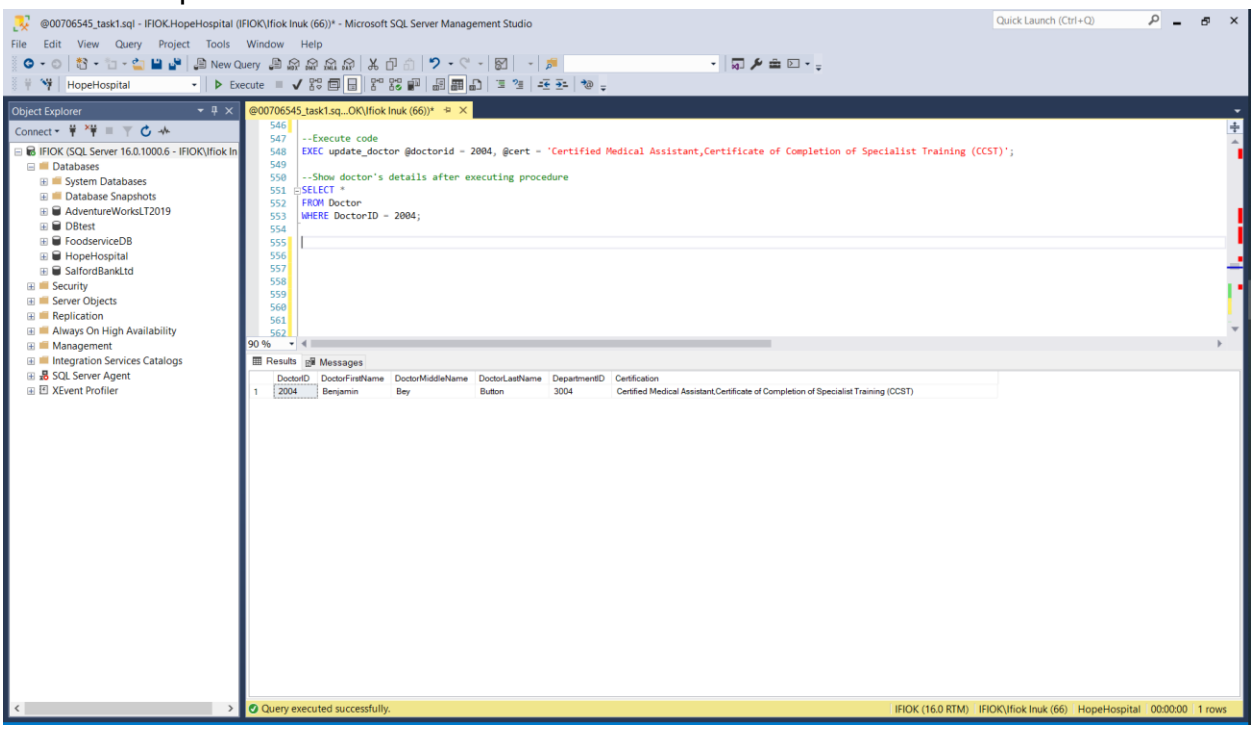

Figure 29: Execution of update_doctor

Question 4d: Delete appointment whose status is already completed.

This deletes appointments whose status is completed and saves the records in the PastAppointment table. A stored procedure (archiveAppointment) was created.

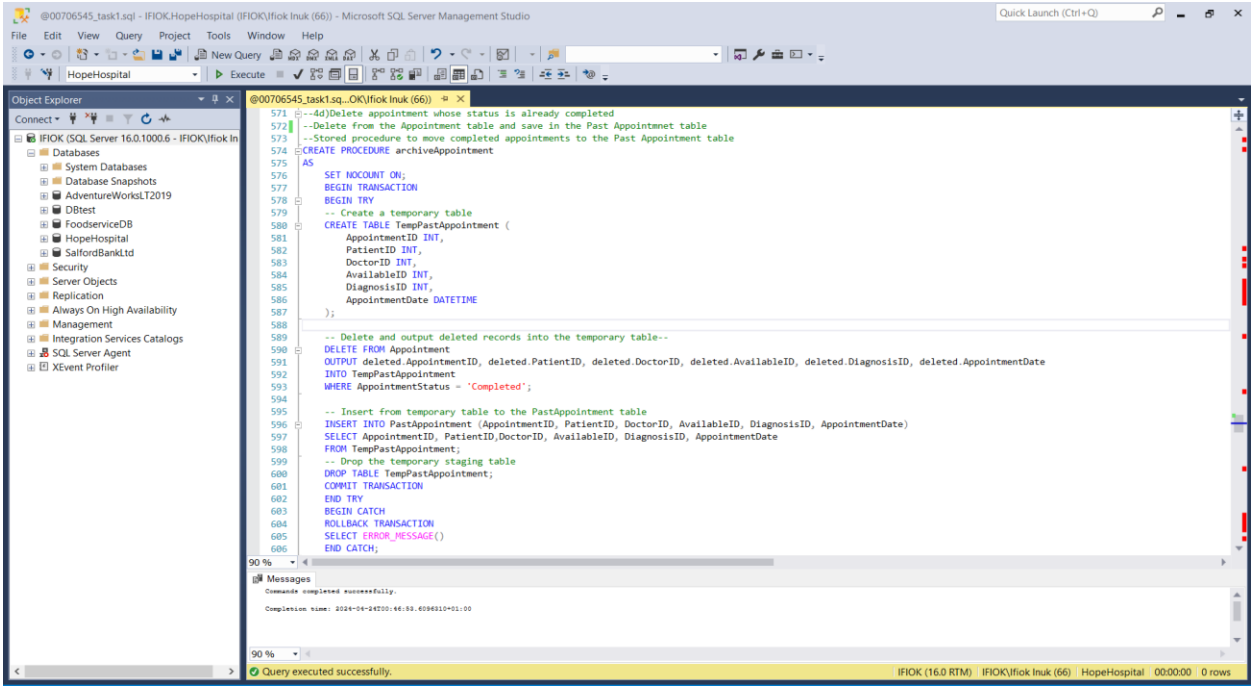

Figure 30: Creation of archiveAppointment

The procedure was executed.

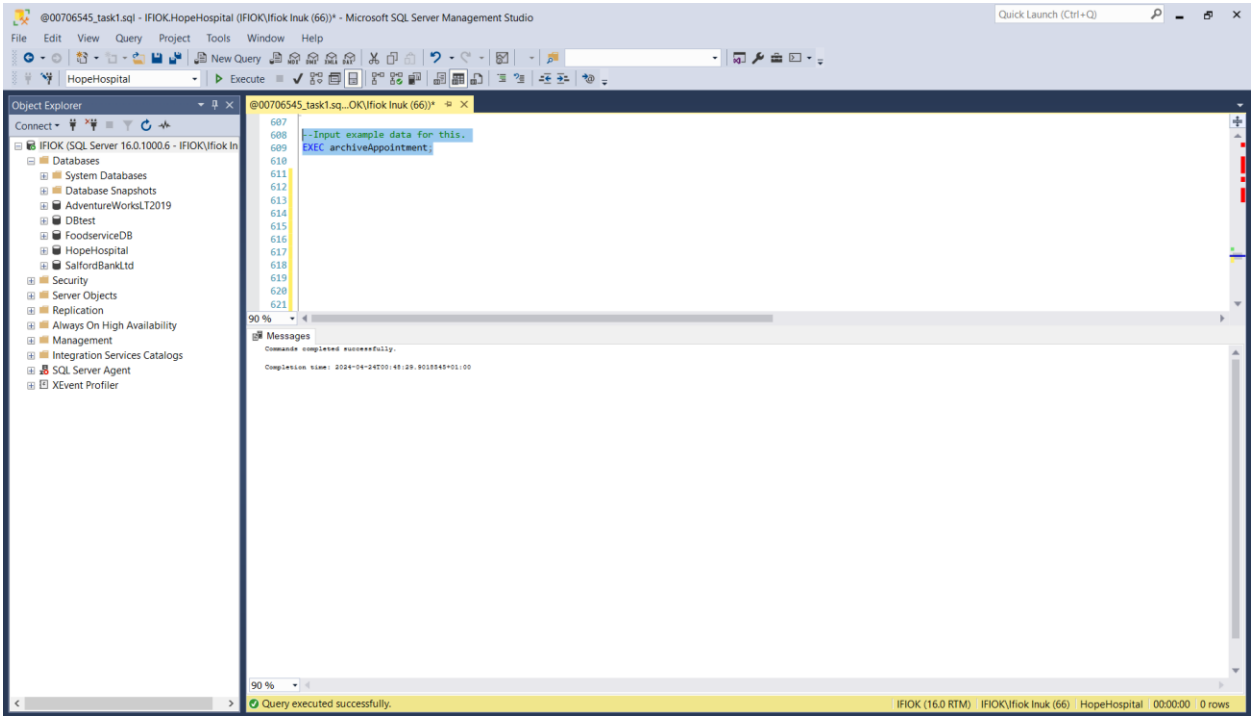

Figure 31: Execution of archiveAppointment

The result shows that the appointments with status-completed have been moved to the PastAppointment table.

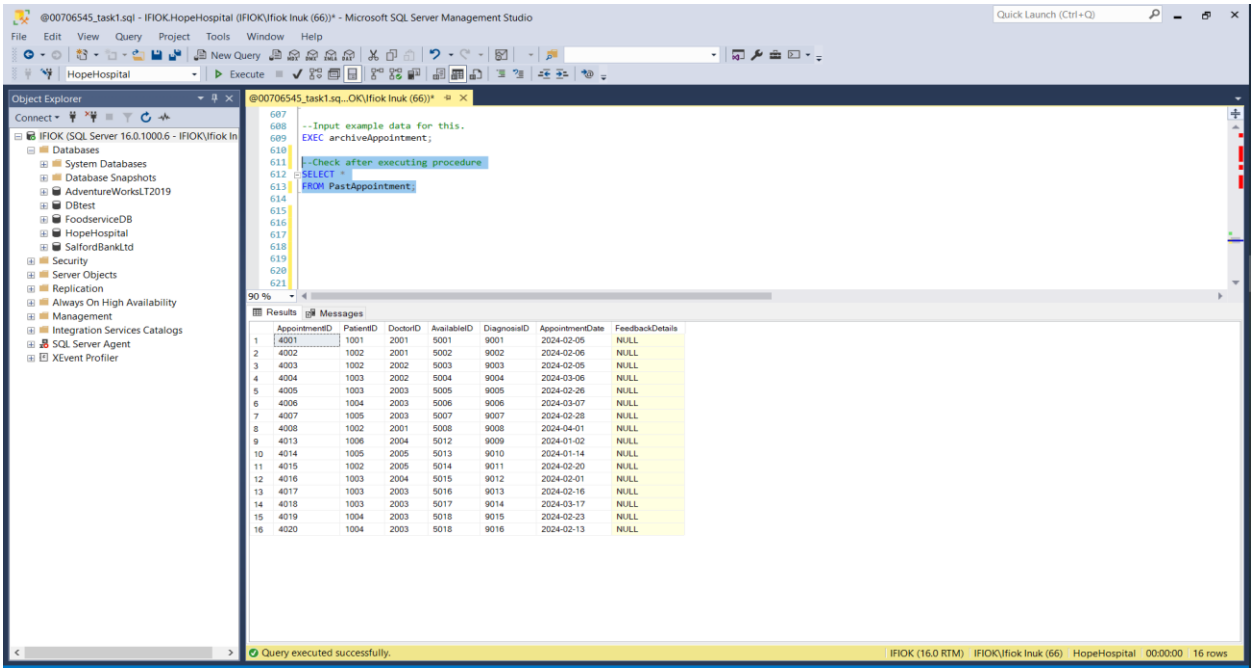

Figure 32: View the archived appointments

<u>Question 5: View for all doctor's appointments, reviews and departments</u>

A procedure (uspFeedback) was created to allow the patients to provide review once appointments are completed and the review is stored in the PastAppointment table.

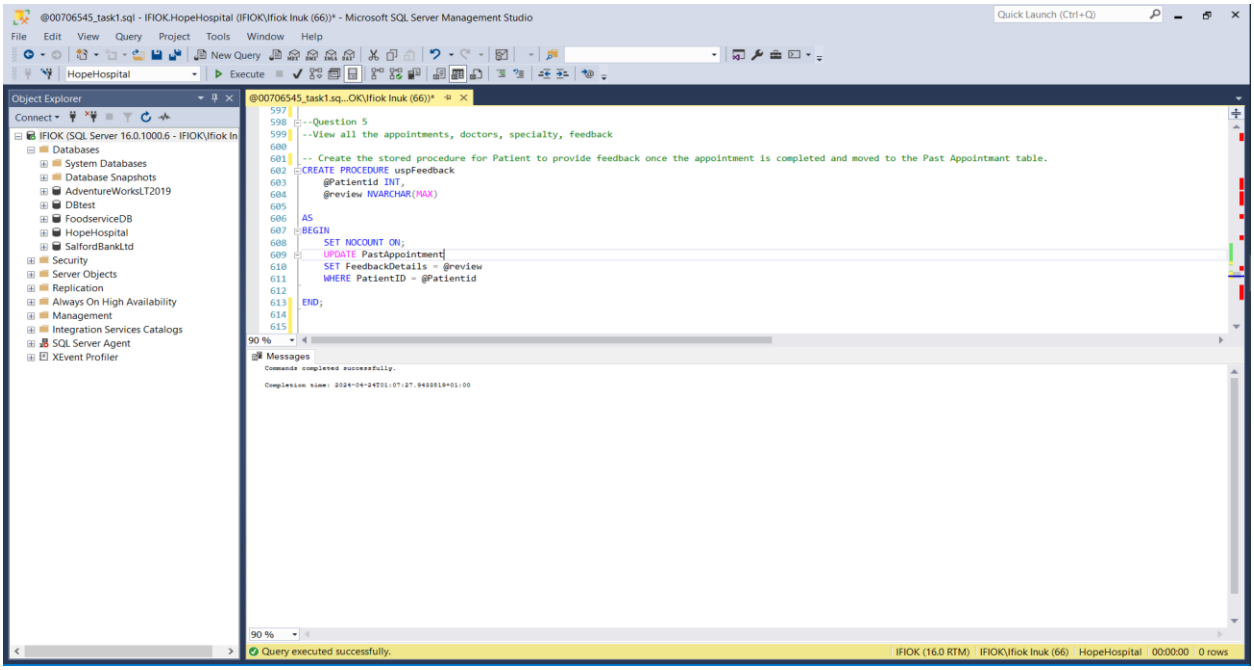

Figure 33: Creation of uspFeedback procedure

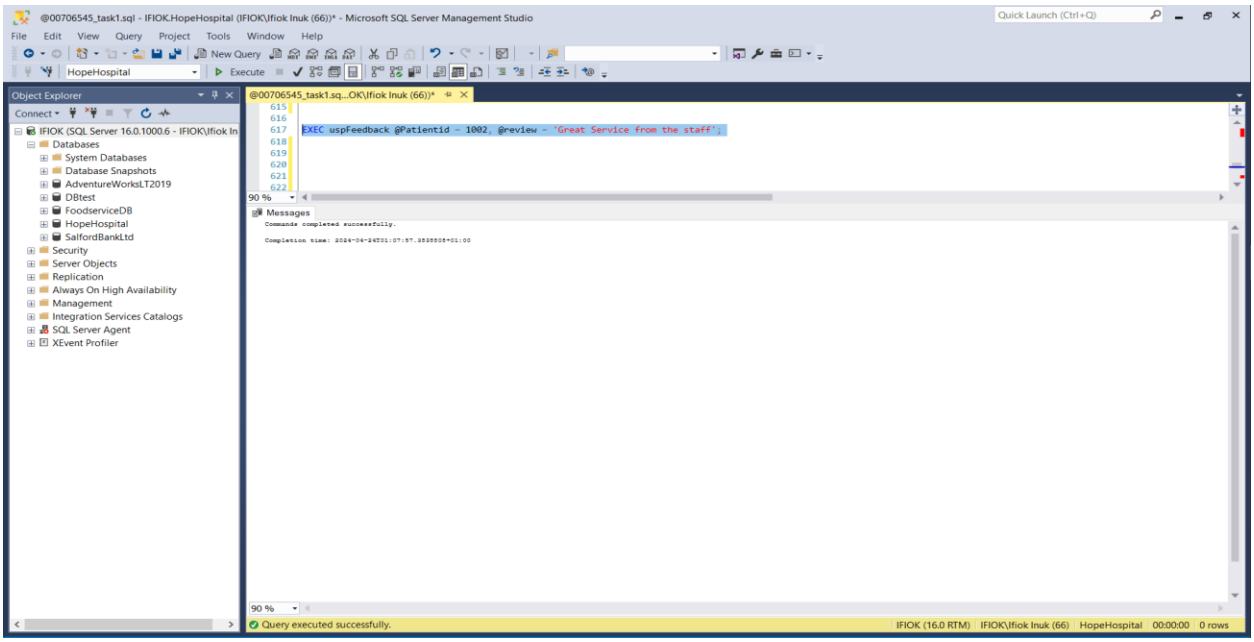

Figure 34: Execution of uspFeedback procedure

CREATE VIEW was used to create the allAppointmentDetails view for all doctor's appointments and other details. UNION ALL was used to combine the Appointment and PastAppointment tables. The Department, Doctor, Diagnosis, and PastAppointment tables were combined with the result of the union using LEFT JOINs. A SELECT statement was used to return specific columns from the resulting table.

A SELECT statement was executed on the view and the result is shown below.

There are 20 appointments so far in the database.

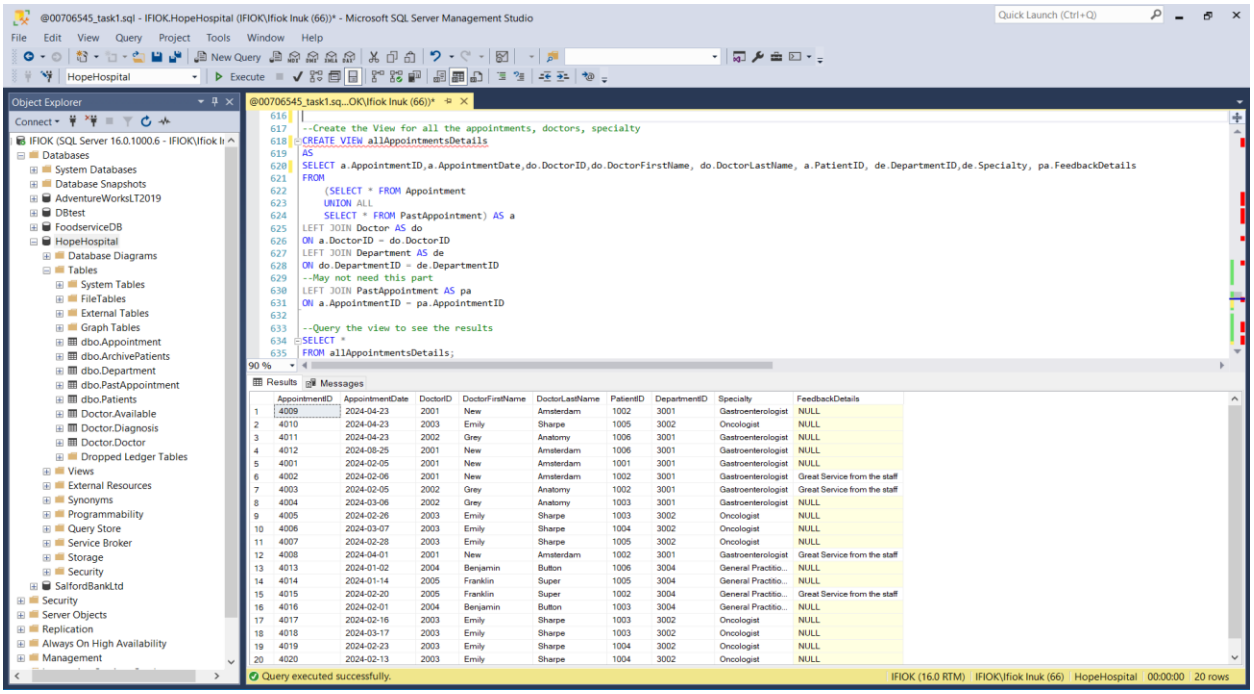

Figure 35: Create and view the allAppointmentsDetails

Question 6: Trigger so that the current state of an appointment can be changed to available when it is cancelled.

A trigger cancel_appt was created using the CREATE TRIGGER function. The DROP TRIGGER was run prior to delete if it already existed and prevent errors. The trigger was created ON the Appointment table and initiated AFTER an UPDATE has been performed on the table.

An IF condition was used to filter for updates WHERE the appointment status is 'Cancelled'.If this condition is true, an UPDATE is performed on the Available table to SET the status to 'Available'.

EXEC was used to execute the procedure. A SELECT query was run before and after the procedure execution to confirm the change.

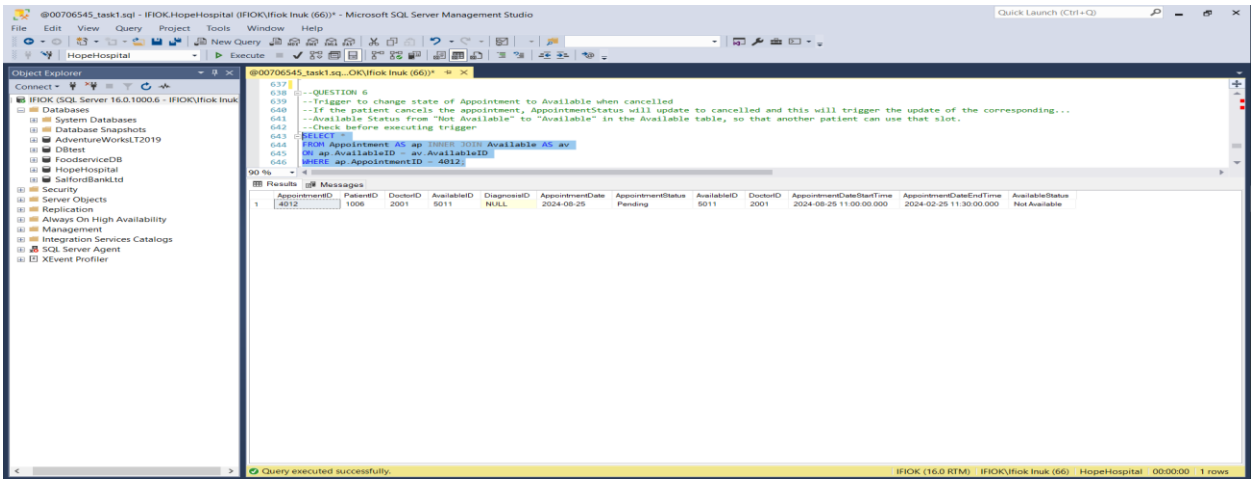

Figure 36: Test query before trigger

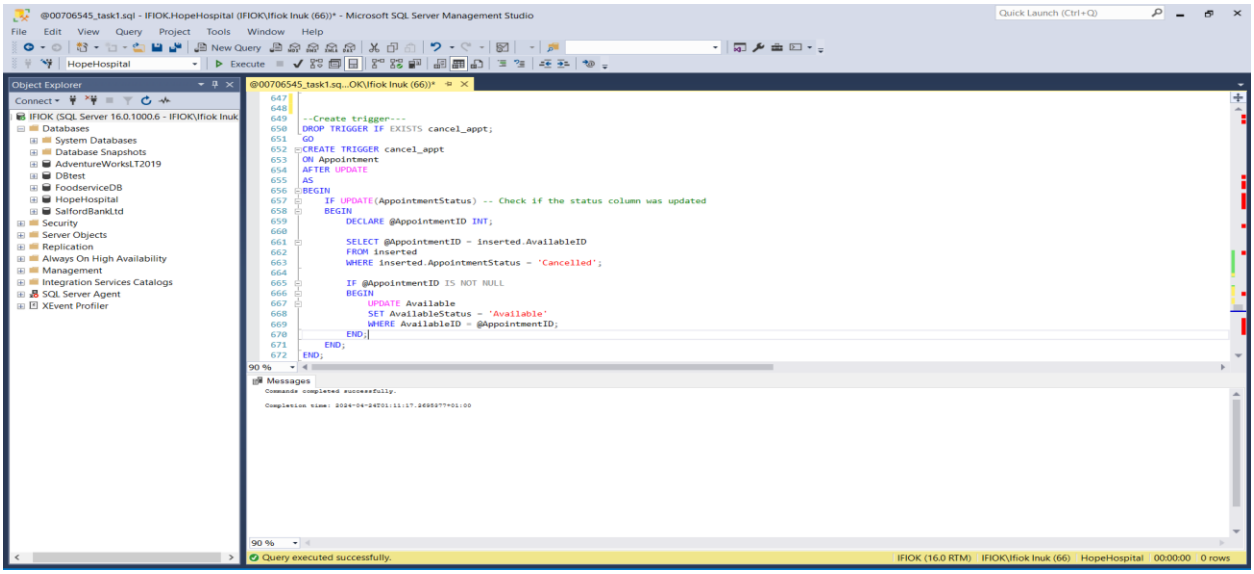

Figure 37: Creation of cancel_appt trigger

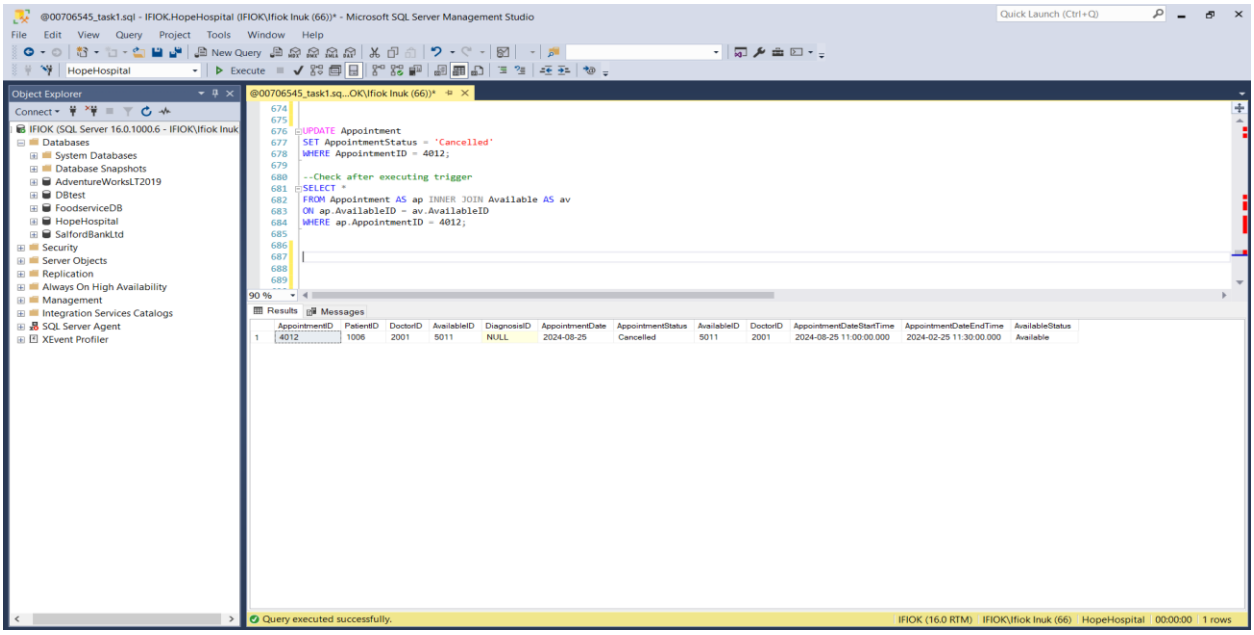

Figure 38: Update Appointment table and test query after trigger

<u>Question 7: Identify the number of completed appointments with the specialty of doctors as 'Gastroenterologist'.</u>

A SELECT statement returning the COUNT of all the records was executed on the resulting table from a subquery. The subquery was a SELECT statement on INNER JOINs between the PastAppointment, Doctor, Department tables filtered using WHERE for the 'Gastroenterologist' specialty. There are 5 completed appointments done by Gastroenterologist.

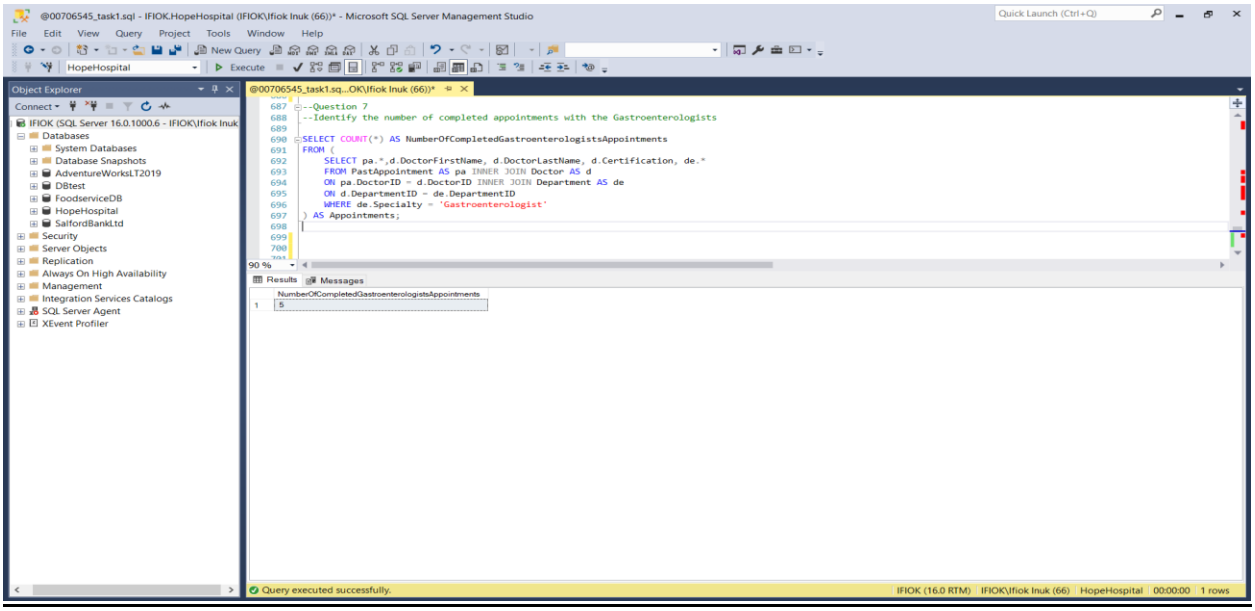

Figure 39:Completed appointments done by Gastroenterologist

Question 8: Additional Recommendations

a) Data Integrity and Concurrency:

Data integrity was ensured by uniquely identifying each record in the tables using primary keys. Foreign keys to correctly define the relationships between the tables. Implementing the correct data type and constraints also maintains data integrity. For example, checking the insurance no, email address, appointment date is not in the past, patient left date is not in the past, use of NOT NULL and UNIQUE in the tables creation. Another application is the use of TRY…CATCH in the uspAddPatient stored procedure. This ensures that the patient's details are entered correctly or returns an error message. Also, triggers like the patient_update_status and patient_left_archive ensured that the patient archiving transaction obeyed the atomicity and consistency rule.

b) Database Security:

Database security is an important concept that needs to be implemented in relational databases; as it will ensure the confidentiality and protection of the data.For this database, some controls were added to secure it and they include:

i)Use of the salted hash method to protect the patient's password. This was implemented as the uspAddPatient stored procedure.

ii)Use of Schemas to optimize user's access and permissions. Patient and Doctor Schema were created, and the necessary database objects were moved to the schemas.

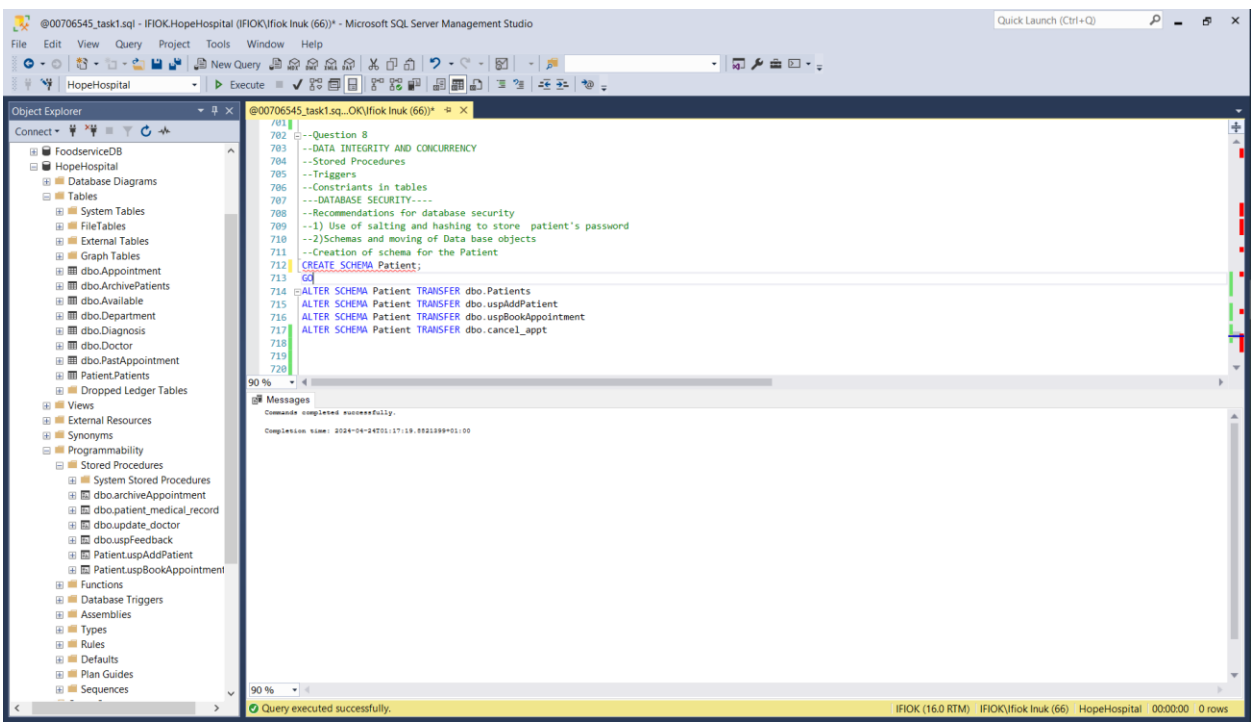

Figure 40: Creation of Patient schema

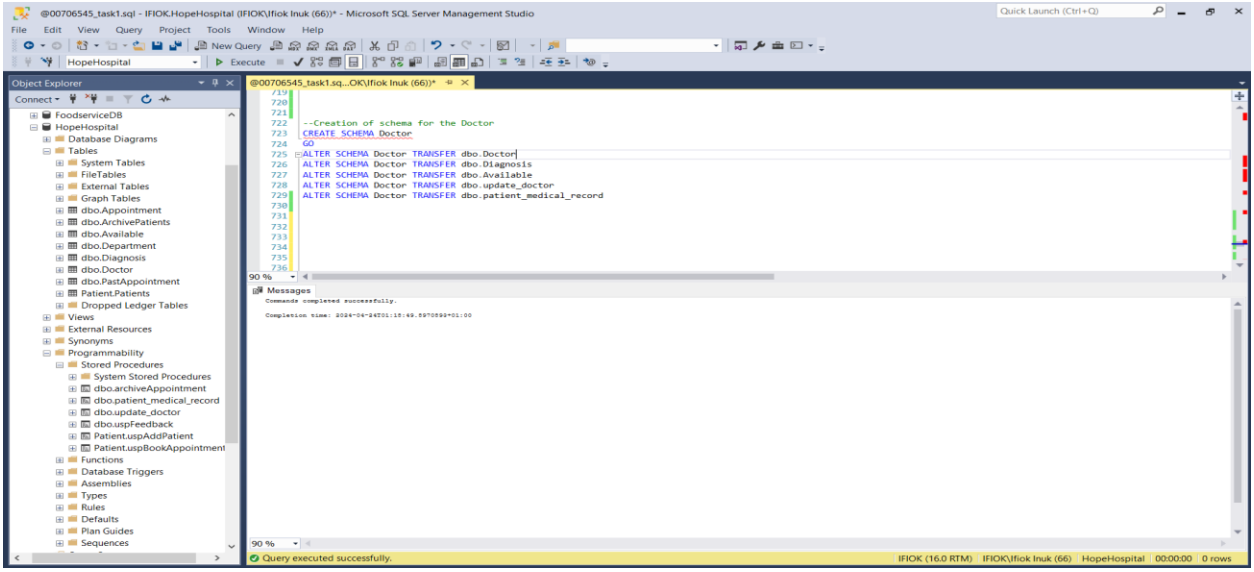

Figure 41: Creation of Doctor schema

Once Schema has been created, schema needs to be referenced when writing queries. For example, the updated query to return the Patients table is shown below.

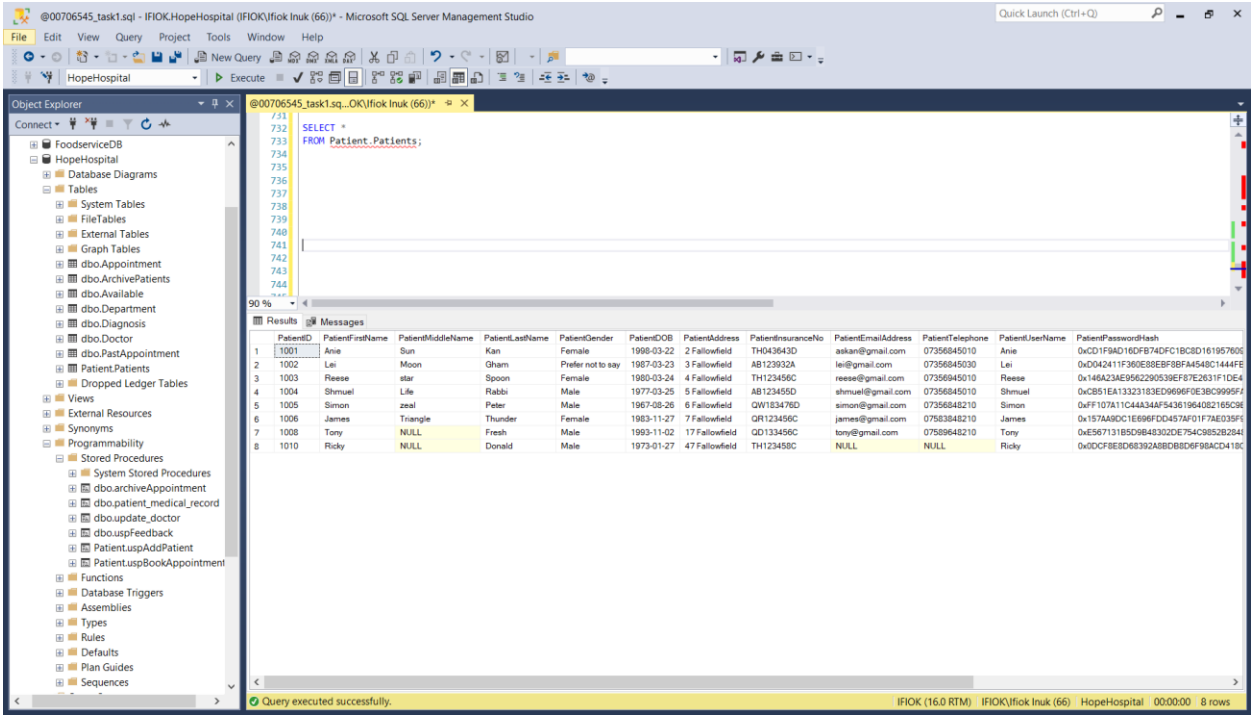

Figure 42: Sample statement after Schema creation

ii) Users, Roles and Privileges: Roles were created for the patients and doctors and privileges were granted as required. Also, login details will be created for the patients, see below for Anie. This ensures that the right restrictions are in place for the different users.

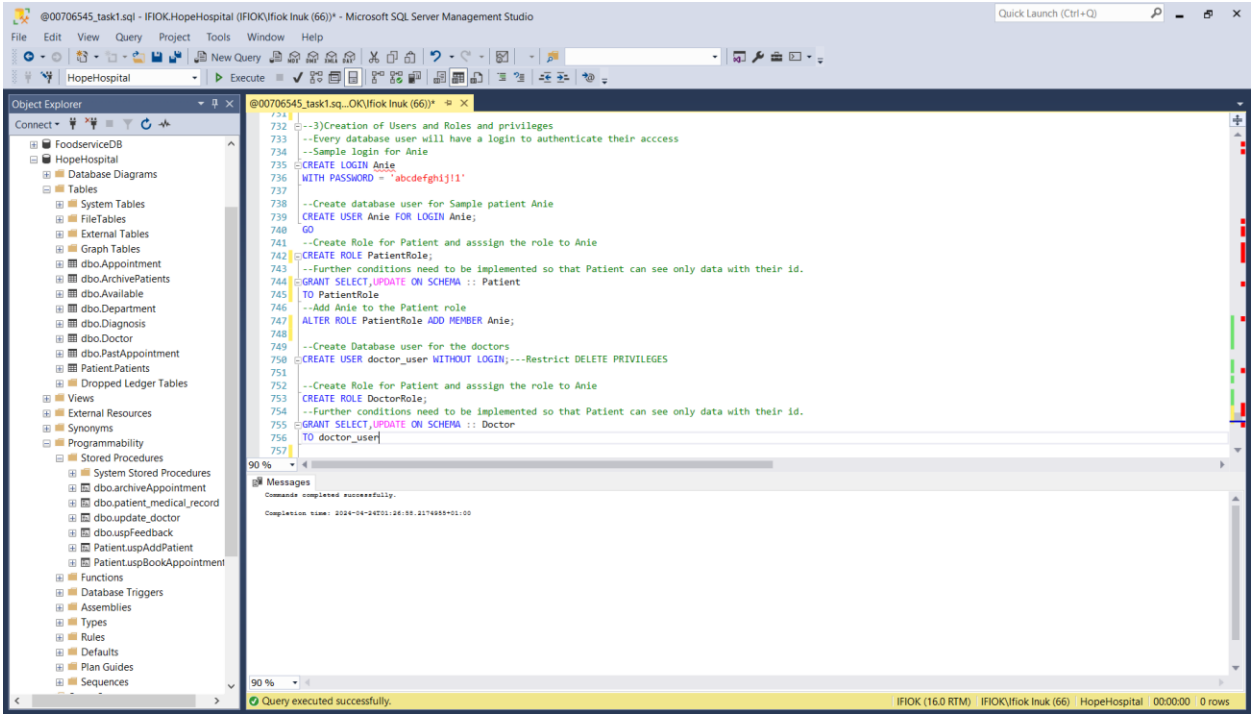

Figure 43: Creation of user, roles, login and privileges

Database security procedures should be reviewed periodically to ensure that the database is protected from illegal activities and data integrity is preserved.

c) <u>Database Recovery and Backup</u>

This is vital in the design of the database as this ensures data integrity in case of a transaction or system failure. It helps to achieve the atomicity (only complete committed transactions affect the database) and durability (no loss of committed transactions) part of the ideal ACID transaction. For the HopeHospital database, a full backup was performed to ensure all the transactions meet this requirement.

The following steps were performed:

i) Creation of the Database backup: A full backup was done and saved to the disk. The backup file name is HopeHospital_@00706545_FullBackup.bak.

ii)Creation of the maintenance plan: A plan was created to perform full backup, perform integrity check and log the report daily.

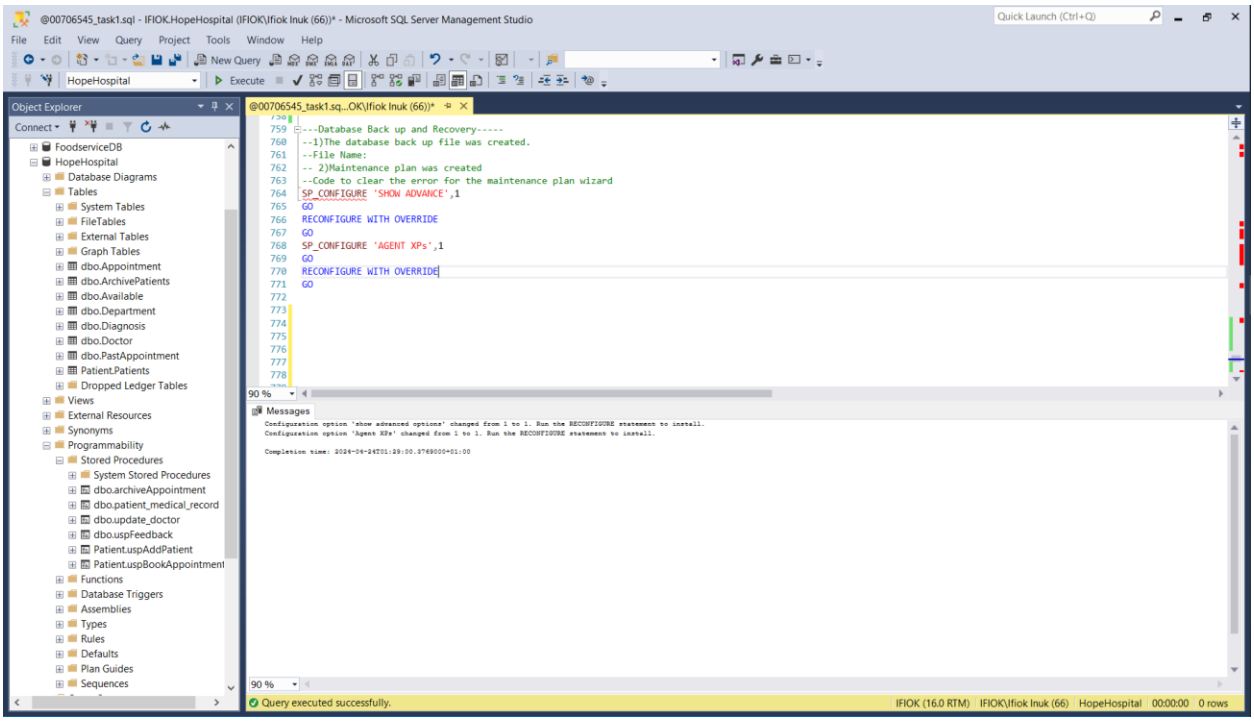

Figure 44: Code to clear error

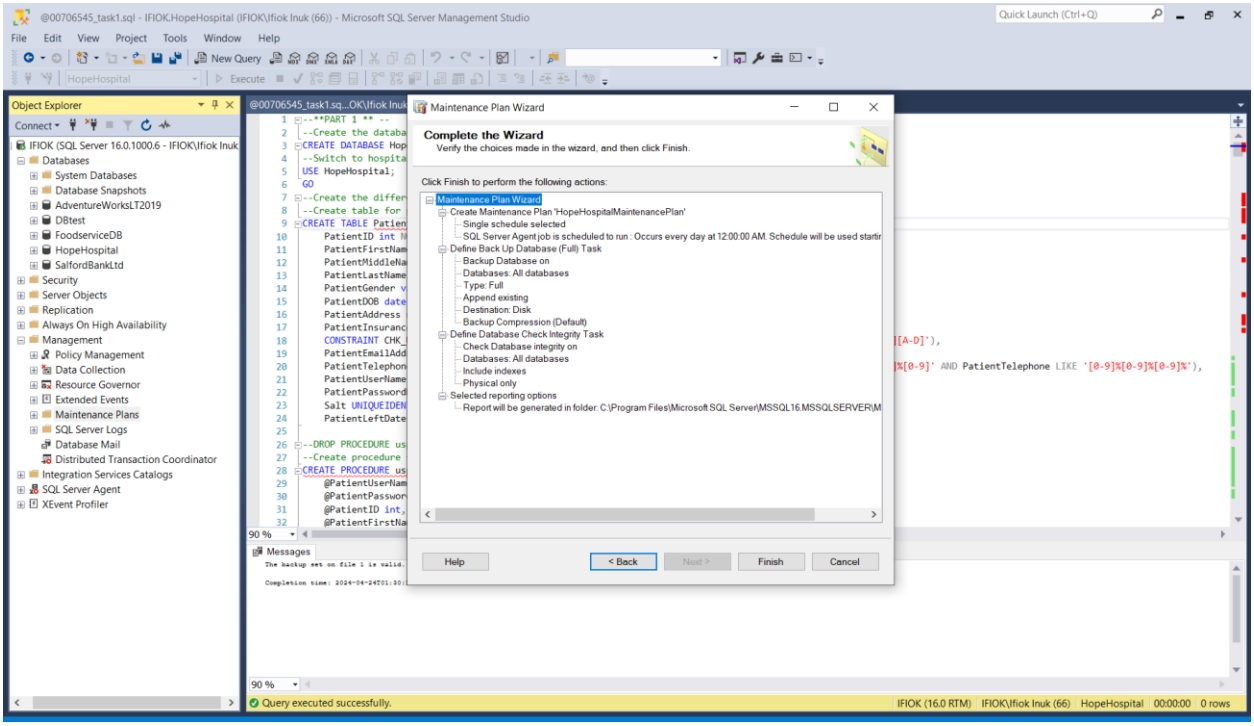

Figure 45: Nightly Maintenance plan

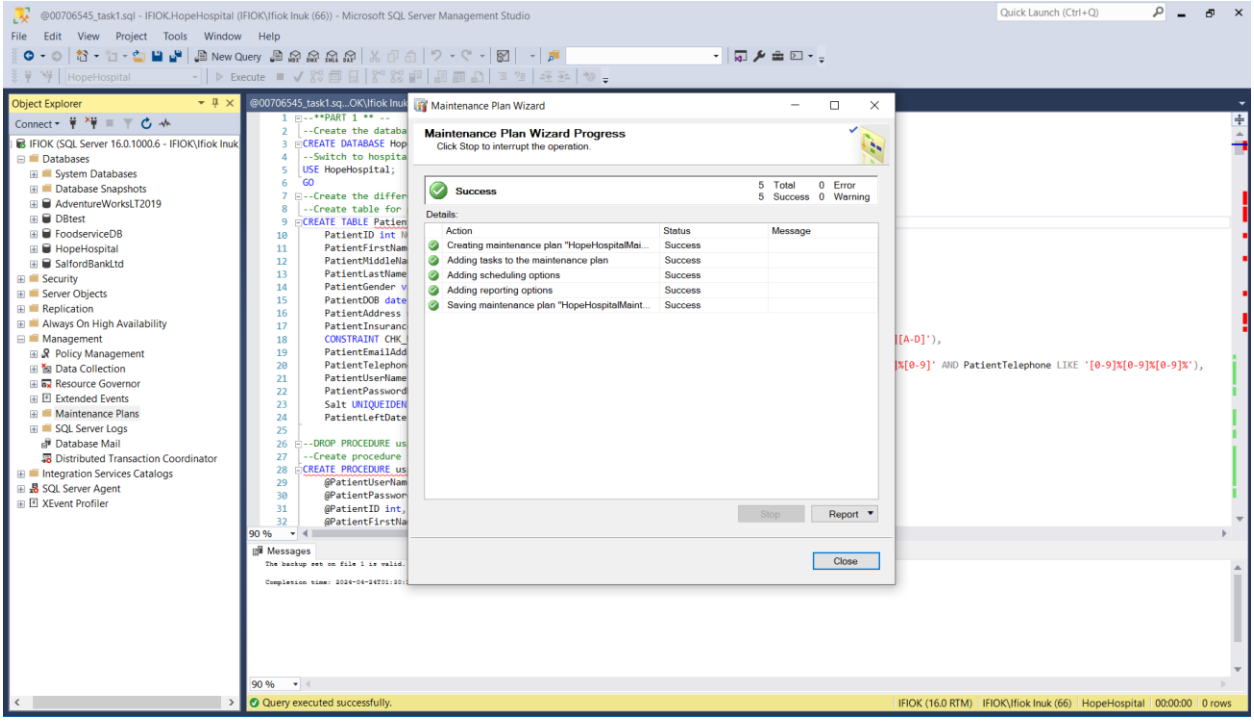

Figure 46: Successful creation of plan

iii) CHECKSUMs were executed to ensure that the database backup is not corrupted and can be restored.

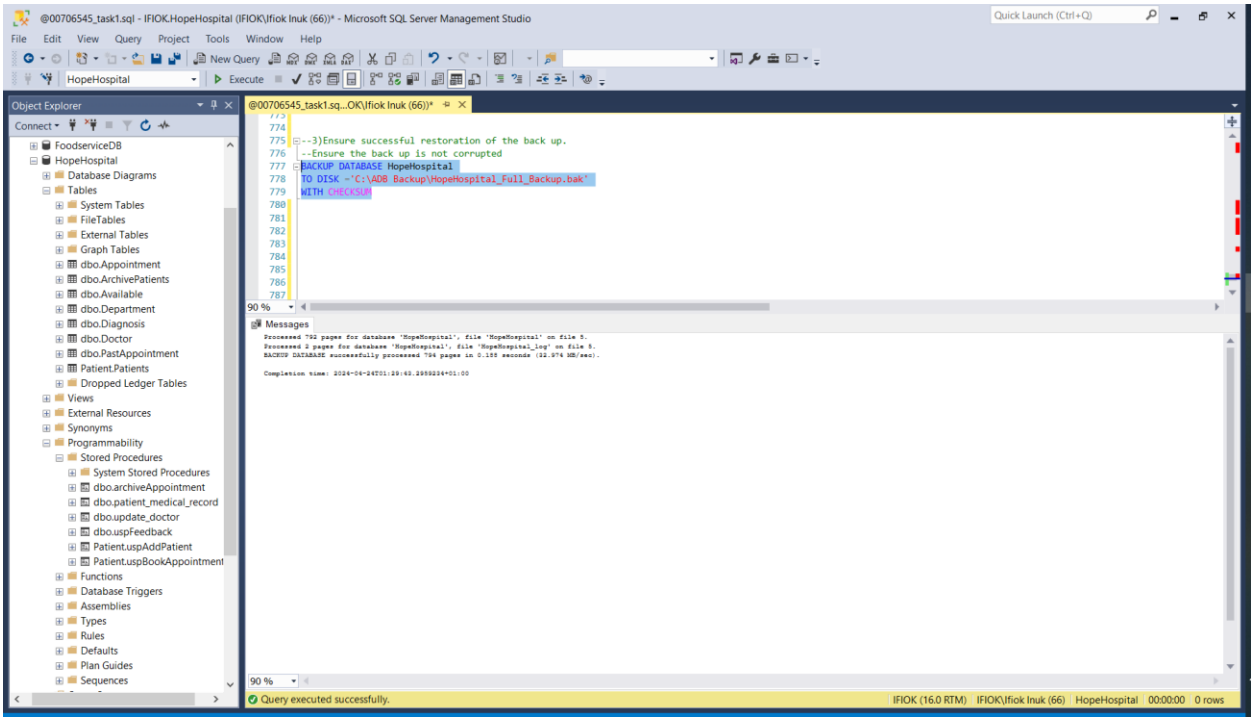

Figure 47: Check that backup is not corrupted

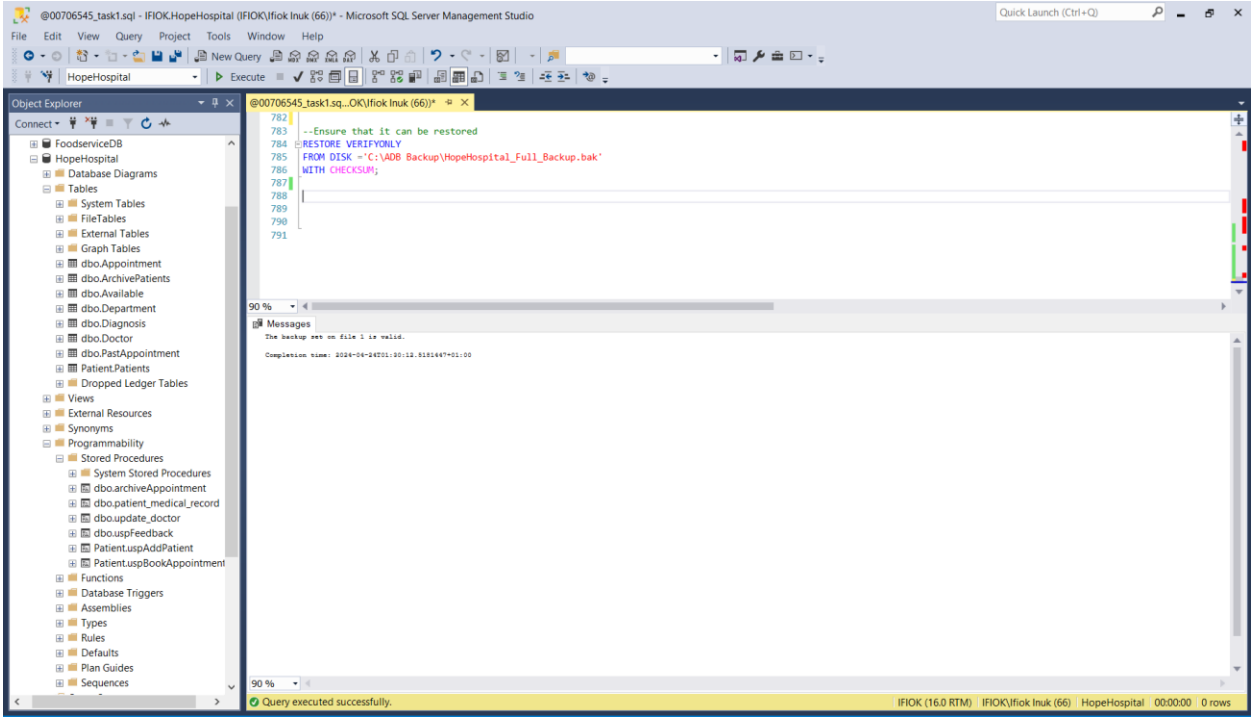

Figure 48: Check that backup can be restored

Conclusion

The HopeHospital database was designed in 3NF to store and analyze the hospital's data. Eight tables were created with the appropriate data type, constraints, and other database objects to ensure data integrity and concurrency. Database security was also implemented through secured passwords, use of stored procedures, schemas, roles and privileges. The database was also fully backed up and a recovery plan was implemented to ensure data integrity. Some of the key functionalities of the database are:

- Storing of patients and doctors' details
- Checking for data integrity during updates of the tables
- Retaining of data on inactive patients and completed appointments
- Easy retrieval of hospital data using stored procedures
- View all the hospital's appointments
- Insights into the system and improvement opportunities.

Food Service Database Report

Introduction

As a database consultant, I was consulted to develop a database for a food service company. The purpose of the database was to store, retrieve and analyze data on the restaurants, the types of cuisines offered, the consumers and their ratings of the different restaurants. The task is to analyze the data and provide valuable insights to the stakeholders and promote informed business decisions. Four csv (restaurants.csv, ratings.csv, consumers.csv, restaurant_cuisines.csv) were provided by the client. My plan to tackle the task involved understanding the provided data, finding the relationships between the datasets, designing the database and objects, and executing queries for analysis.

Microsoft SQL Server and SSMS were used to implement T-SQL statements and create database objects for the database.

Part 1

The steps taken to design, create the database, import the four csv files and create four tables are stated below.

1. The database was designed based on the following thought process:
    a. Entities (Restaurant and Consumer)
    b. Relationships between Entities: Many - to - Many
    c. Splitting into tables (Restaurant, Consumer, Ratings, Restaurant_Cuisines)
2. The database named “FoodserviceDB” was created using the “CREATE DATABASE” statement.

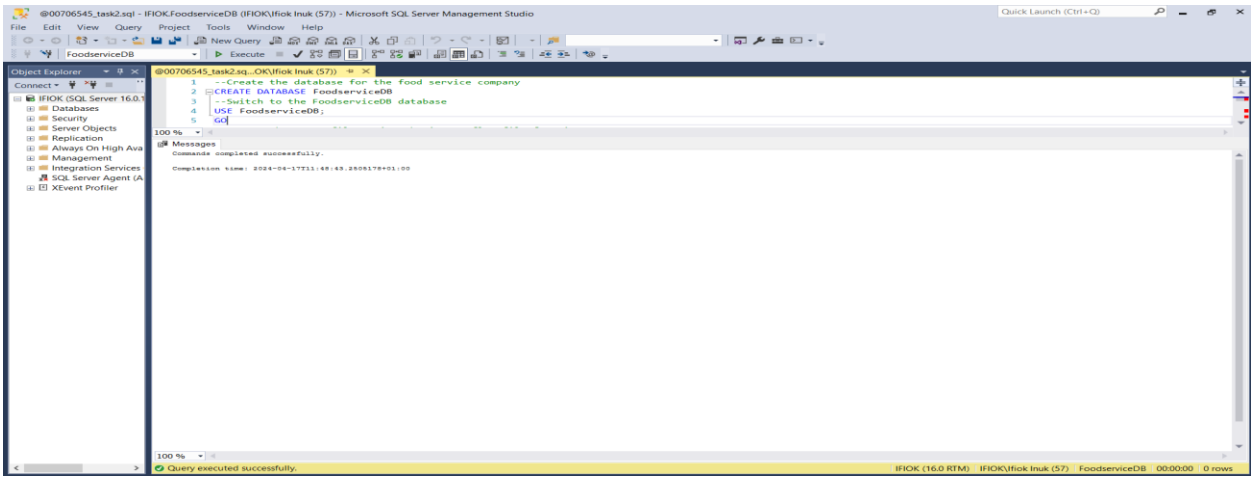

Figure 49: Create Foodservice database

3. The provided csv files were imported into the FoodserviceDB database using the “Import Flat File” function. This was achieved by expanding the database in the object explorer as shown below.

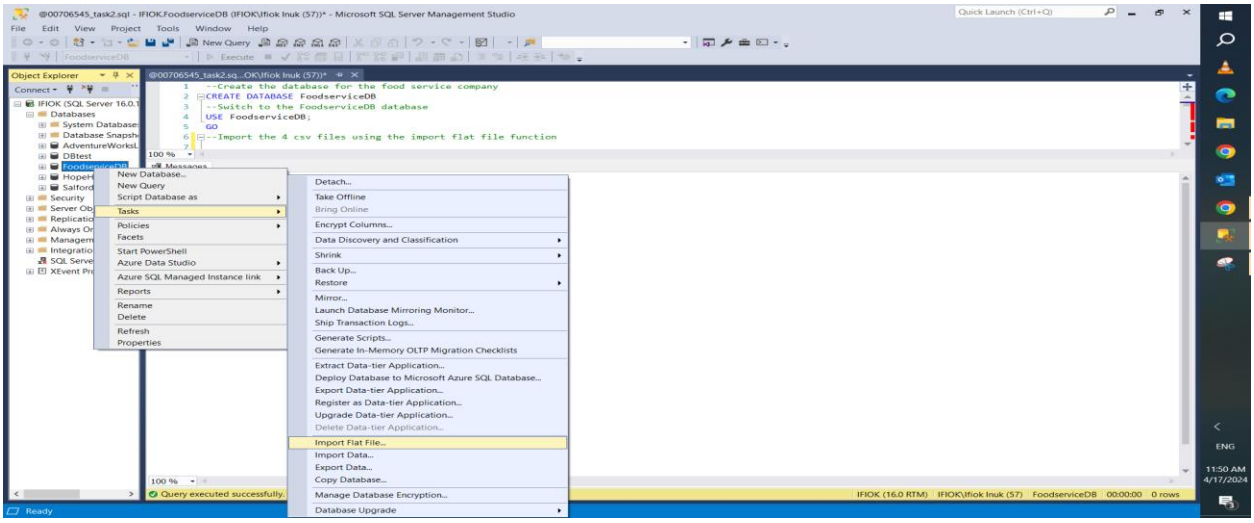

Figure 50: Import csv file(1)

4. The file to be imported (restaurants.csv) was selected.

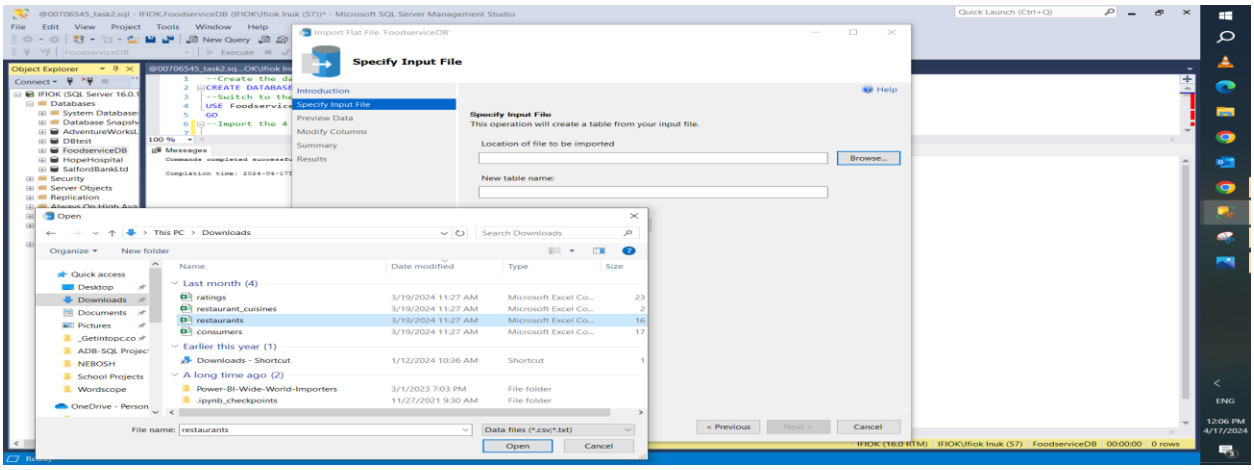

Figure 51: Import csv file(2)

5. For each of the file, the data was previewed, and columns modified before importing, to ensure successful insertion into database tables and maintain correctness of the data. The columns for the restaurants.csv file was modified as follows:
    - Restaurant_ID was checked as the primary key
    - Zip_Code data type was changed from int to nvarchar(10)
    - Latitude and Longitude data type were changed from float to decimal(10,8) and decimal(11,8)
    - Franchise data type was changed from bit to varchar(10)

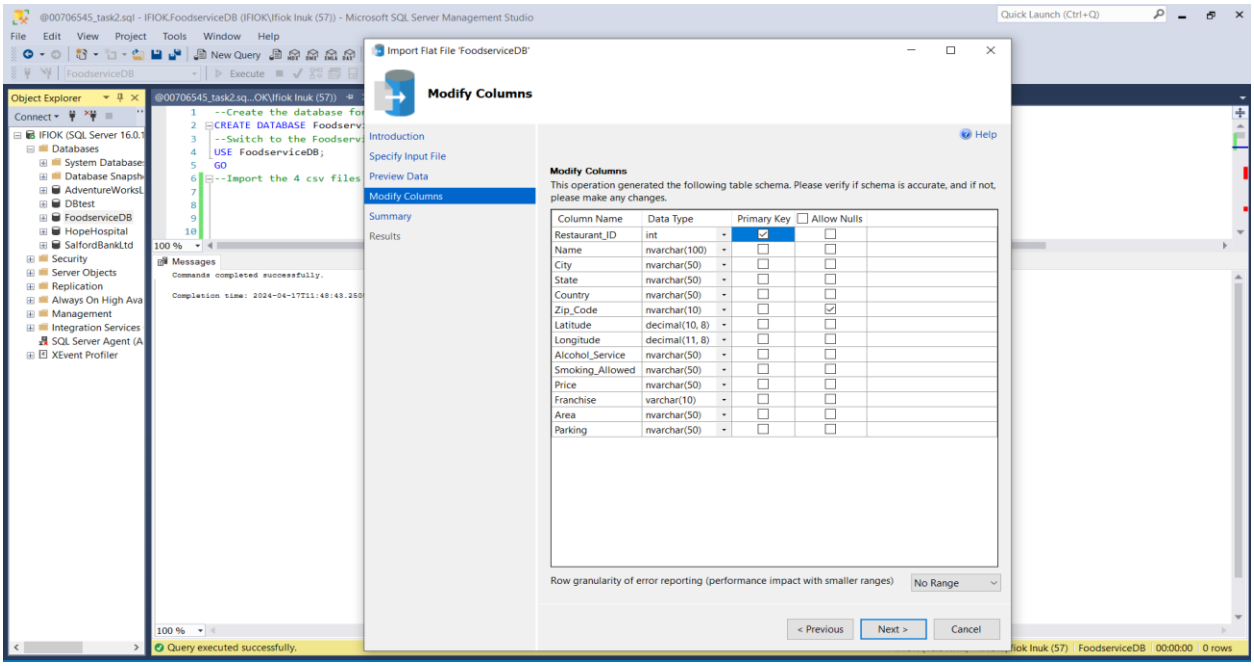

Figure 52: Modification of restaurants columns

6. The other three csv files were also imported using steps 2 to 5. Columns were modified as appropriate.

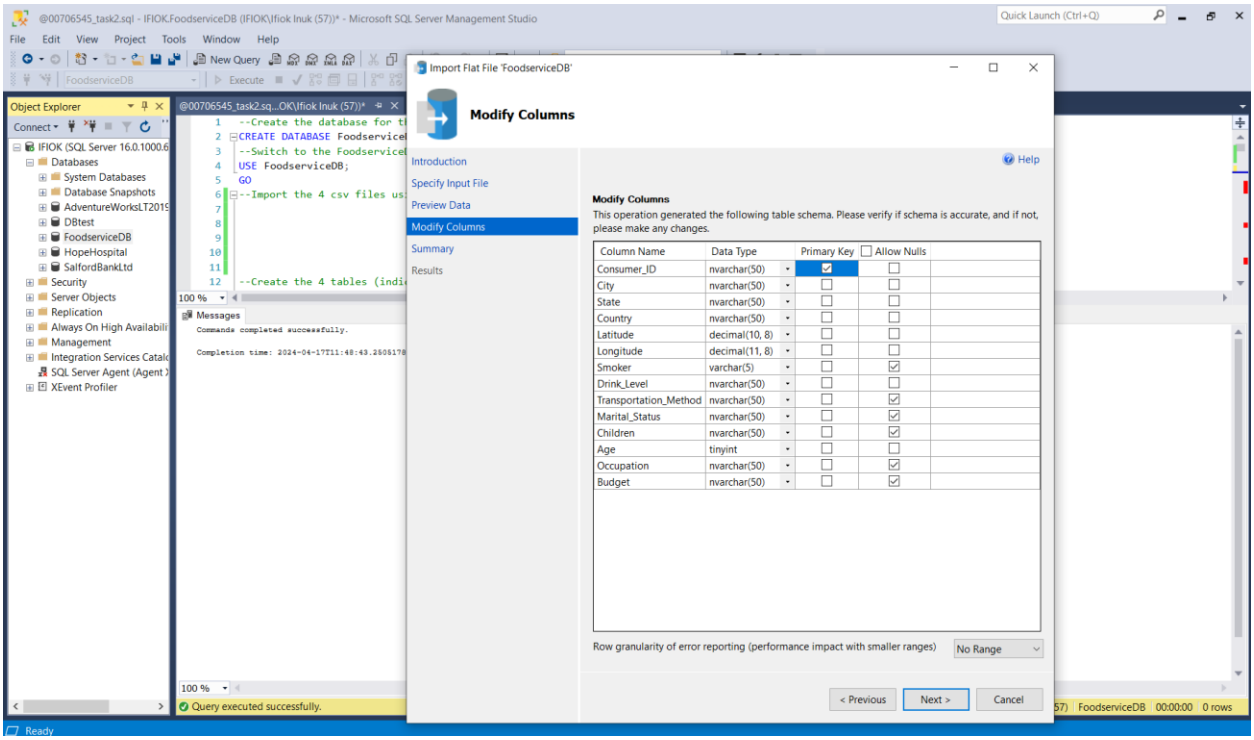

Figure 53: Modification of consumers columns

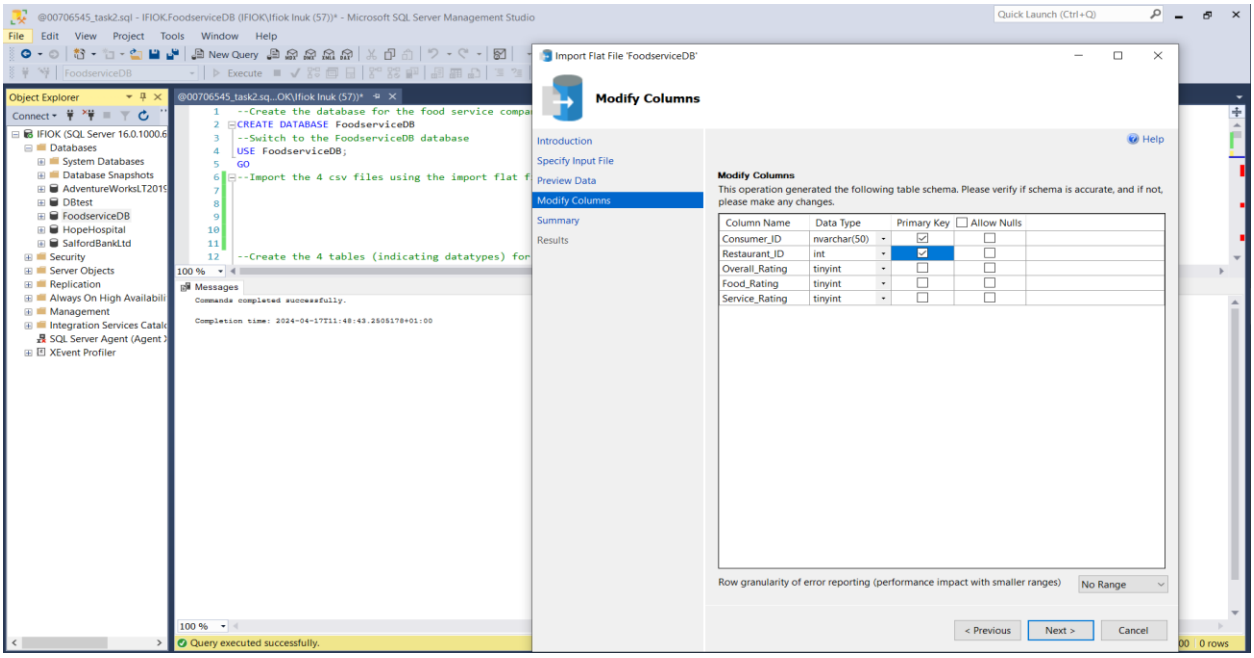

Figure 54: Modification of ratings columns

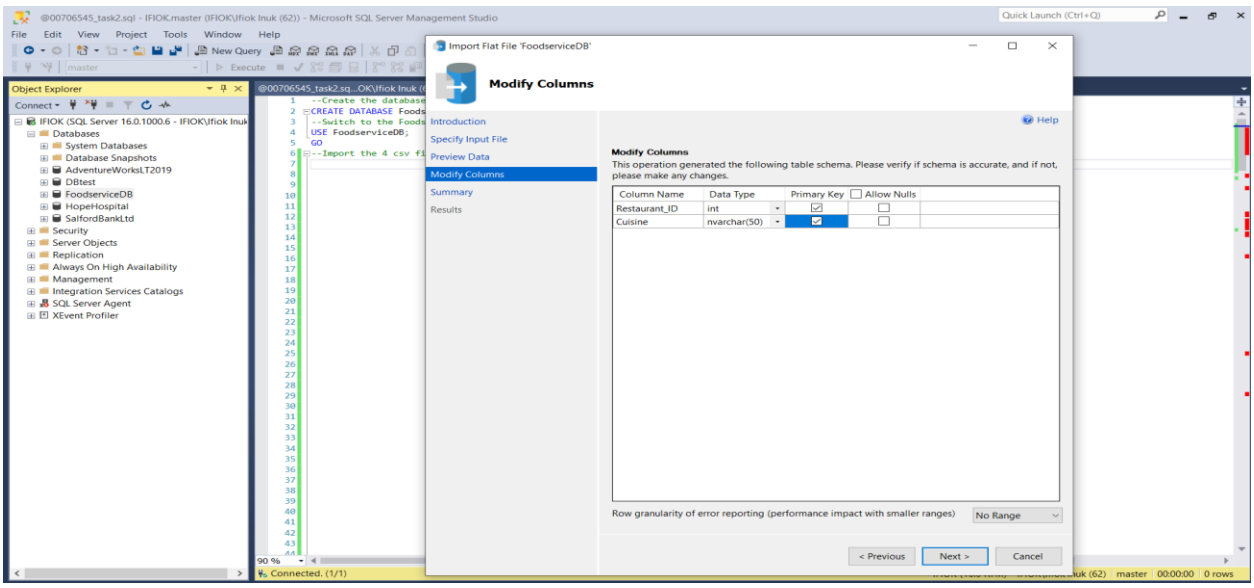

Figure 55: Modification of restaurant_cuisines columns

7. Once the files were imported successfully, the tables for the database were manually created. This was done to ensure program reusability. If a different set of data is provided, minimal changes will be made to the program.
    a) Restaurant table was created using the CREATE TABLE statement. The table contains information about the restaurants.

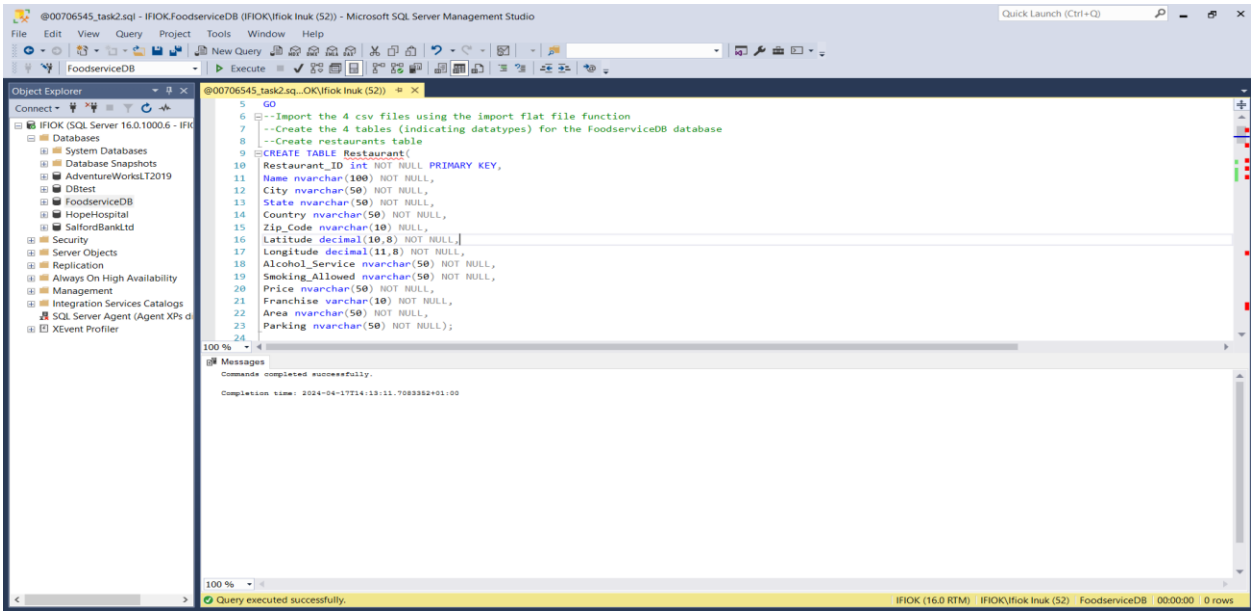

Figure 56: Creation of restaurants table

b) The Consumers table: It contains information on the consumers. The primary key was assigned to the Consumer_ID column with ***nvarchar(50)*** as the datatype because it contains non-digits. Latitude was assigned the ***decimal(10,8)*** and longitude was assigned the ***decimal(11,8)*** datatype because this allows for precision which is important in this case. Age was assigned the ***tinyint*** datatype because it contains small integers (0 - 255). Smoker and Drink_Level were assigned ***varchar(n)*** because they contain letters. Other columns were assigned the ***nvarchar(n)*** data type as they contain unicode characters. ***NULL*** and ***NOT NULL*** constraints were added as required.

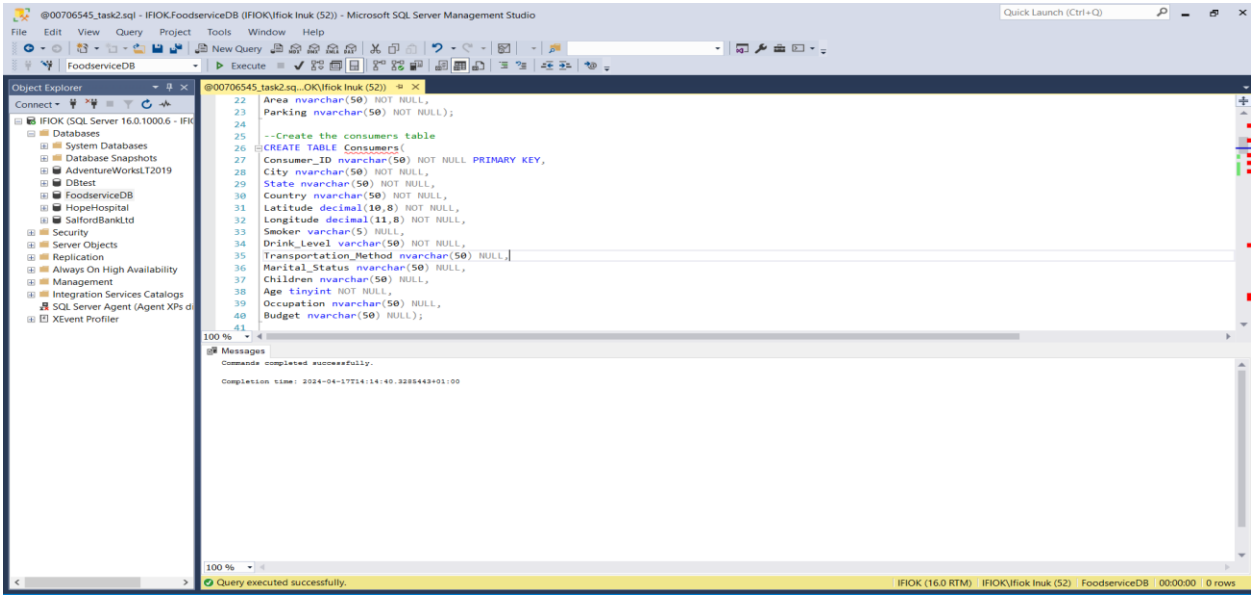

Figure 57: Creation of consumers table

c) Ratings table contains information on the consumers rating on the restaurants.

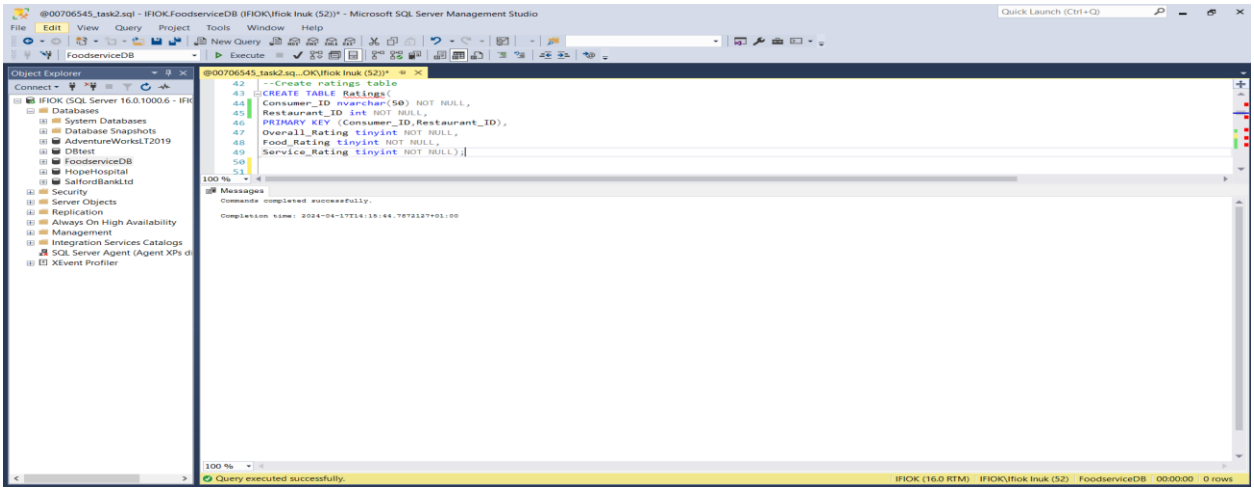

Figure 58: Creation of Ratings table

d) Restaurant_Cuisines table contains information on the types of cuisine offered by the restaurant.

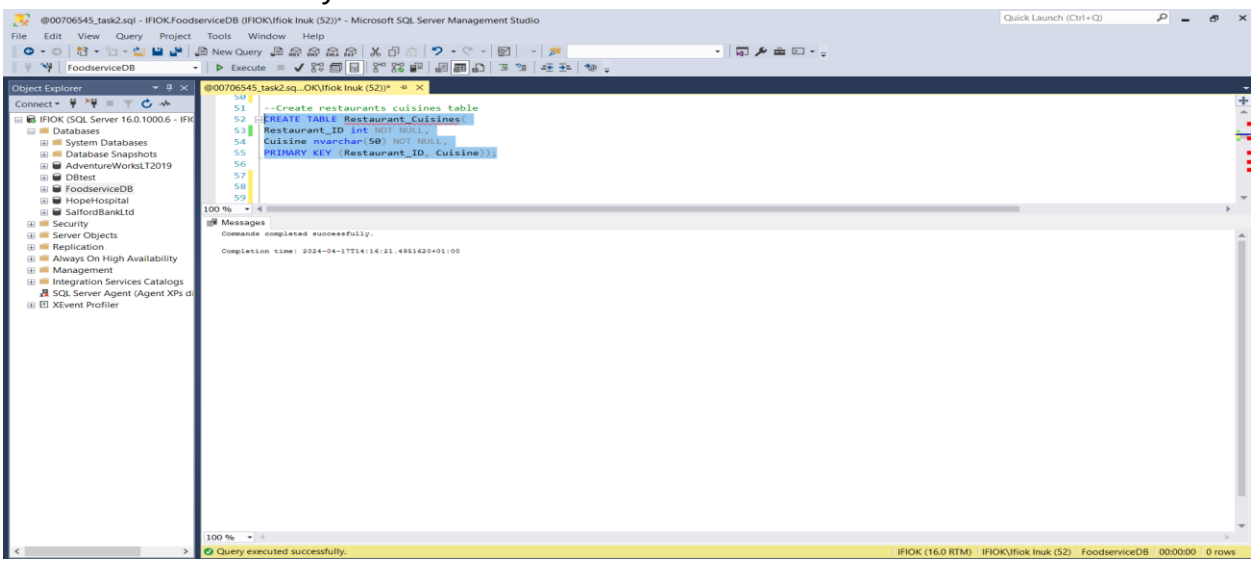

Figure 59: Creation of Restaurant_Cuisines table

8. The Restaurant_Cuisines and Ratings tables were altered to add the constraints for the foreign keys and connect the tables.

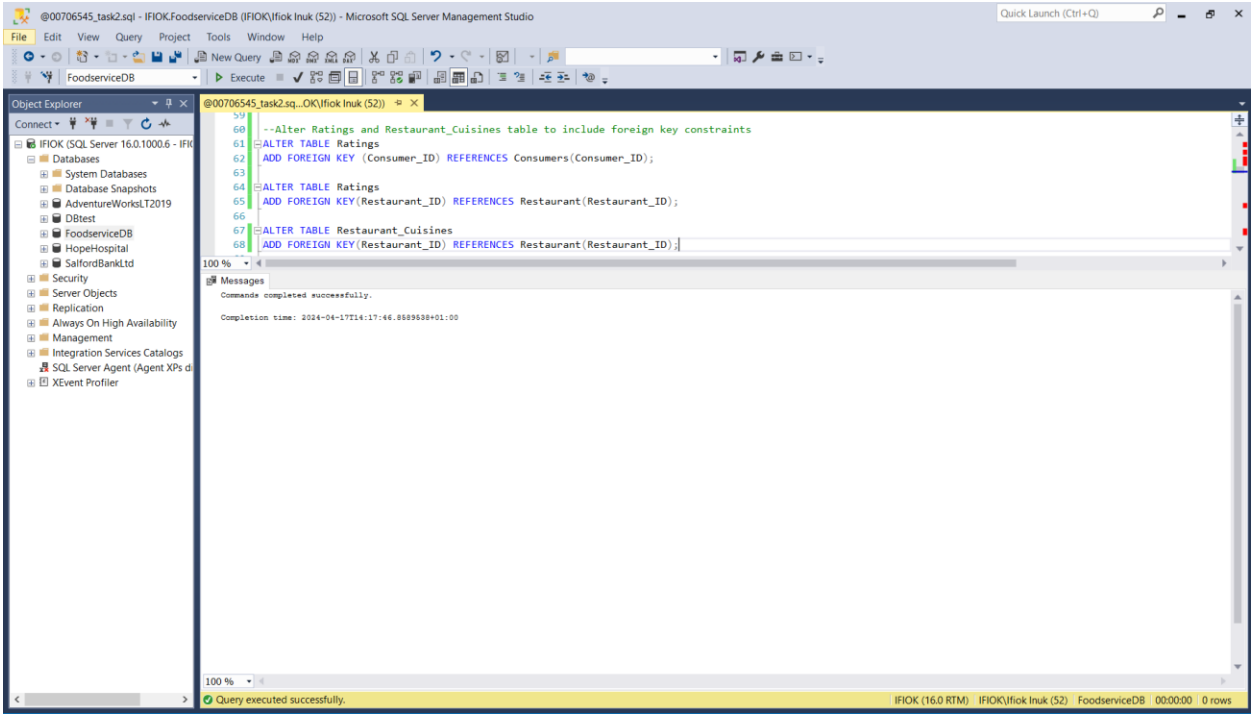

Figure 60: Alter tables to add foreign keys

9. The database diagram was generated to visually compare the relationships between the tables. This was achieved by expanding the FoodserviceDB in the object explorer, then right clicking on “Database Diagrams”, then click “New Database Diagram”. The four tables were then selected.

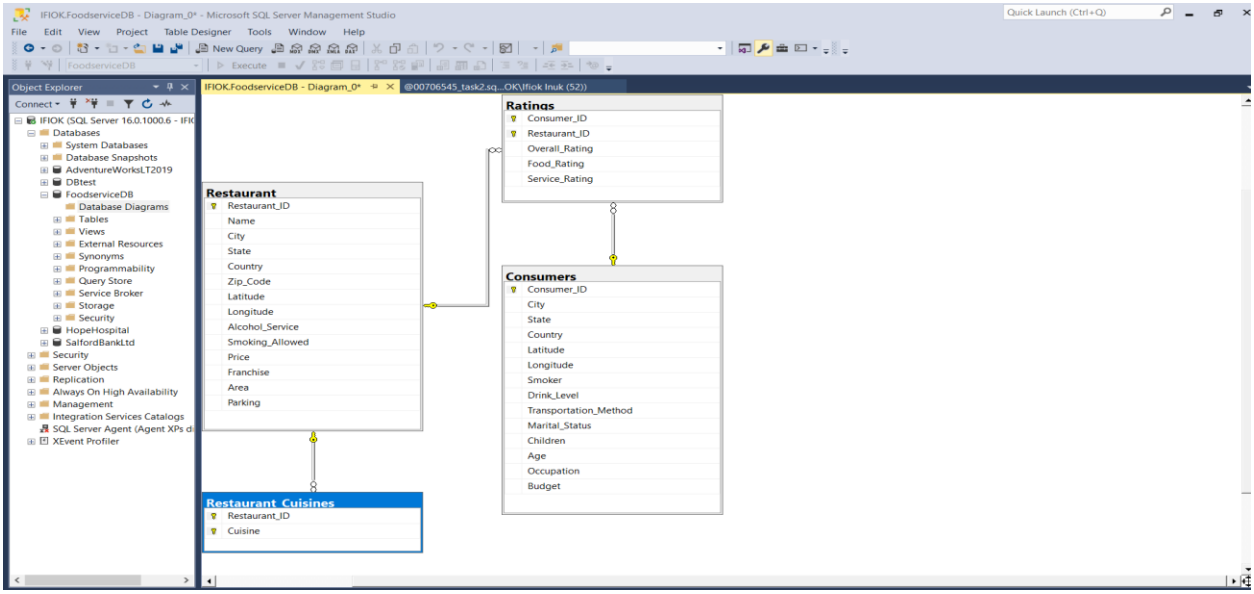

Figure 61:Database diagram

10. Once the four tables were created, the data from the imported csv files were inserted into them.

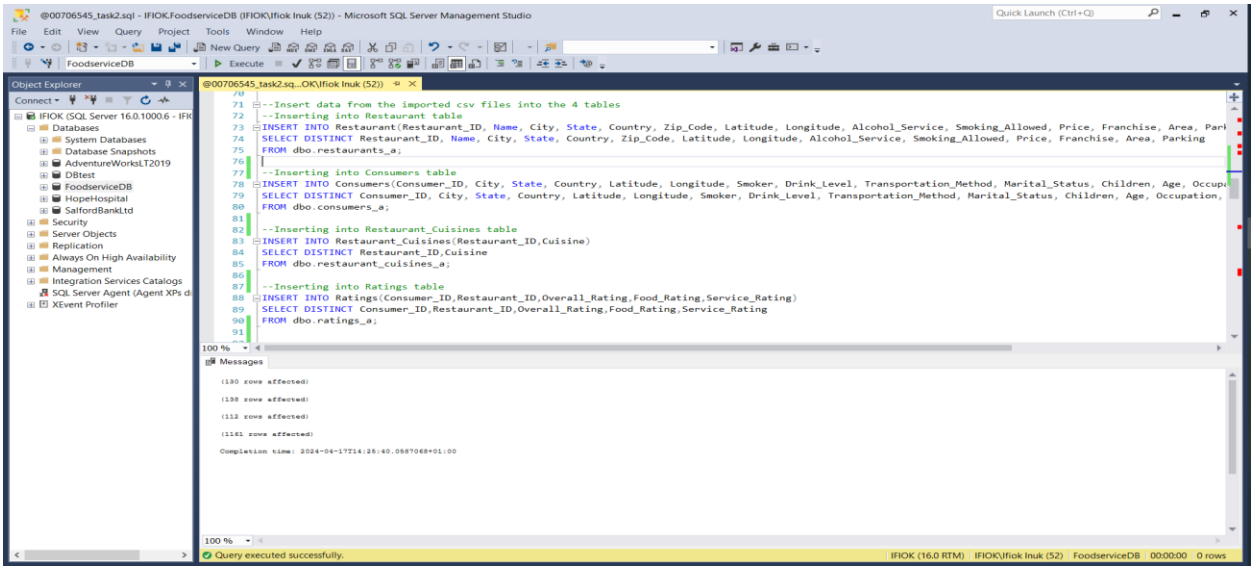

Figure 62: Insert data from the csv files into the database tables

11. SELECT queries were written to confirm that the tables have been populated accordingly.

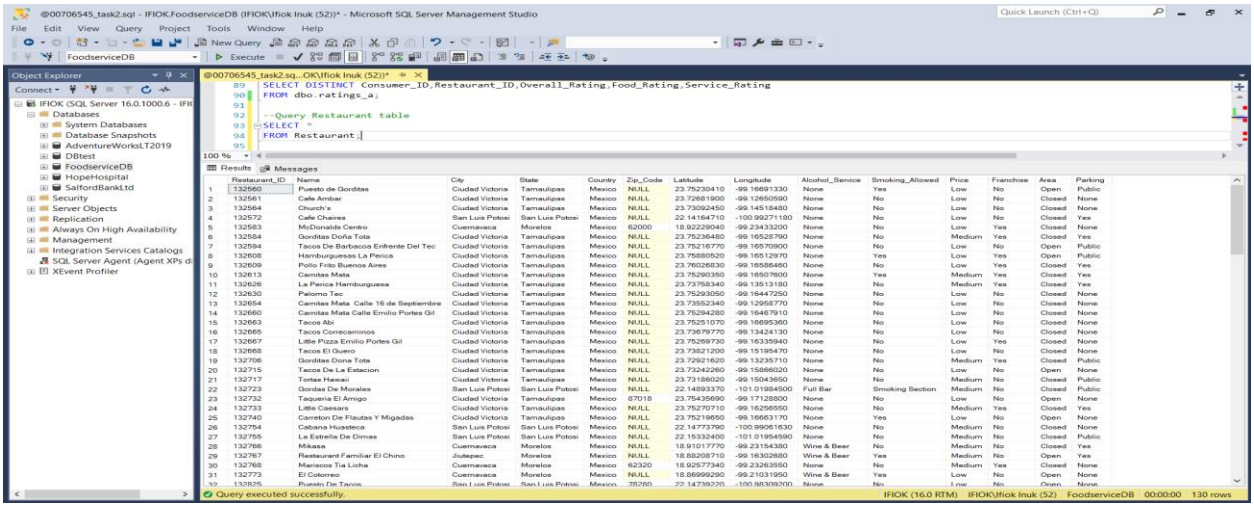

Figure 63: Populated Restaurant table

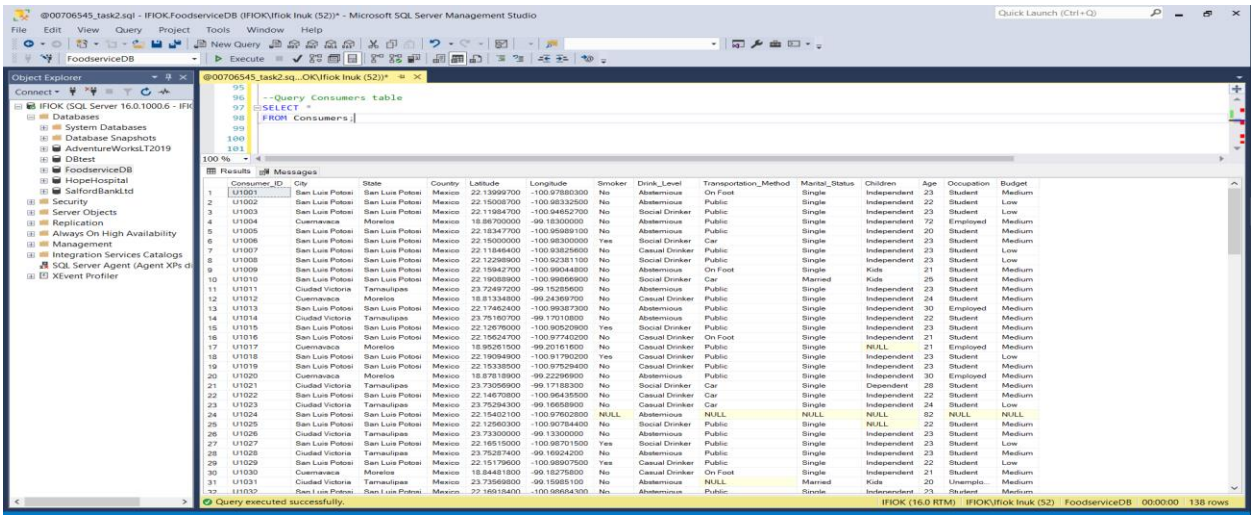

Figure 64: Populated Consumers table

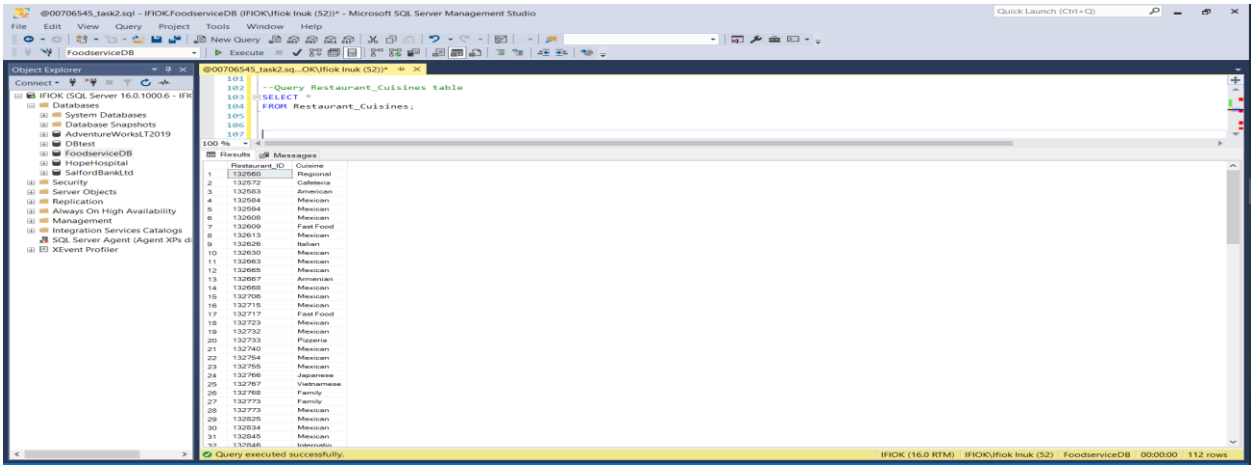

Figure 65: Populated Restaurant_Cuisines table

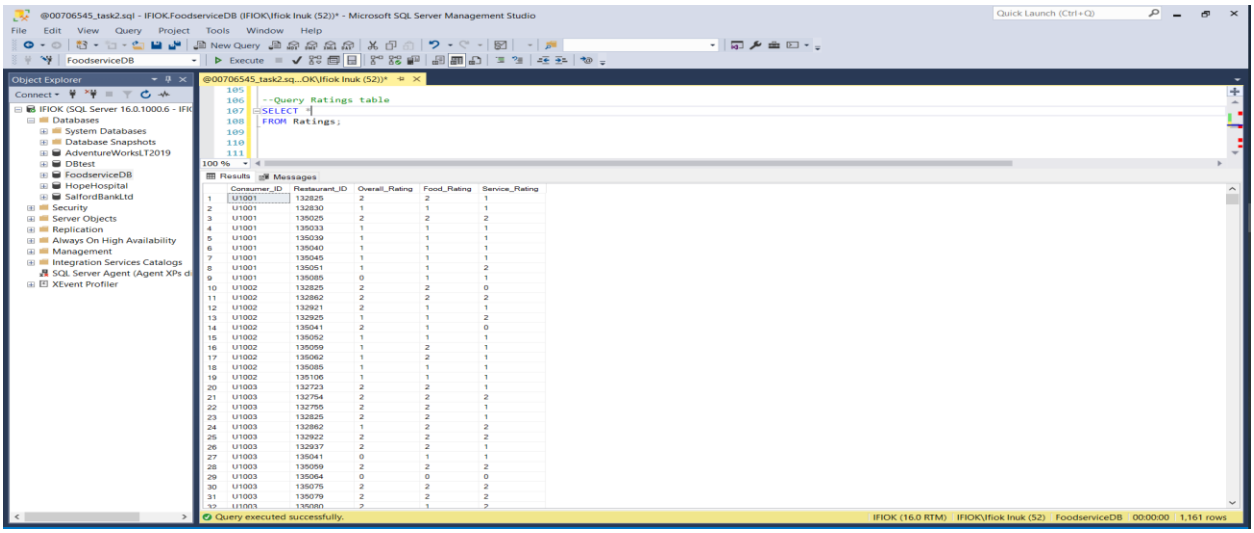

Figure 66: Populated Rating table

12. The tables created from the imported csv files were dropped to optimize memory.

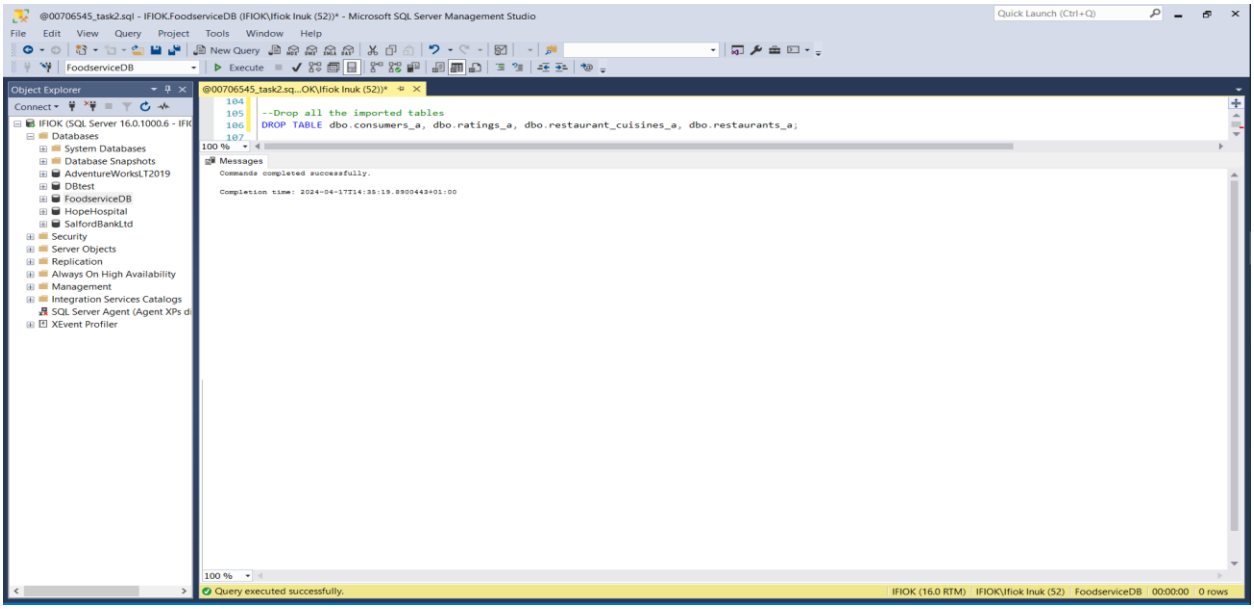

Figure 67: Delete temporary tables from imported csv files

Part 2

Question 1

The statement below lists all the restaurants with a medium range price, open area and serving Mexican food. The result from the join was then filtered using the WHERE clause to return the names of the restaurants with Mexican cuisine, open area and medium price.

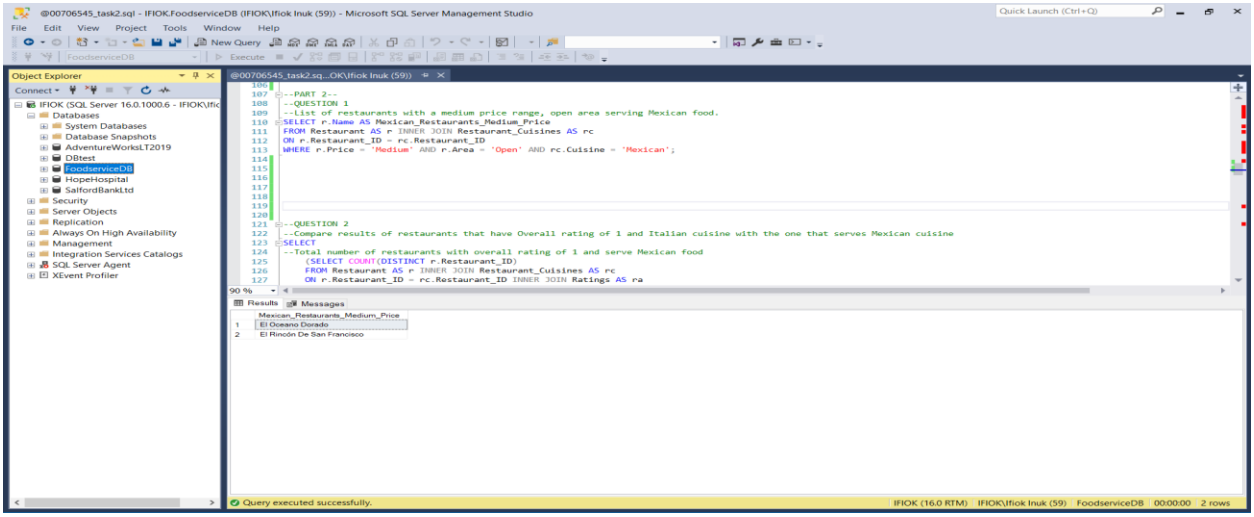

Figure 68: Names of the restaurants with Mexican cuisine, open area and medium price

Question 2

The statement below compares the total number of restaurants that have an overall rating of 1 and serve Mexican dishes to the total number of restaurants that have an overall rating of 1 and serve Italian dishes.

Analysis:

The results shows that there are 27 Mexican and 4 Italian restaurants with an overall rating of 1, which is assumed to be a medium rating (considering 2 = high and 0 = low). The restaurants are all located in Mexico, consumers live in Mexico and there are more Mexican (28) than Italian (4) restaurants. It is only natural for the results to show that there are more medium rated Mexican restaurants than Italian.

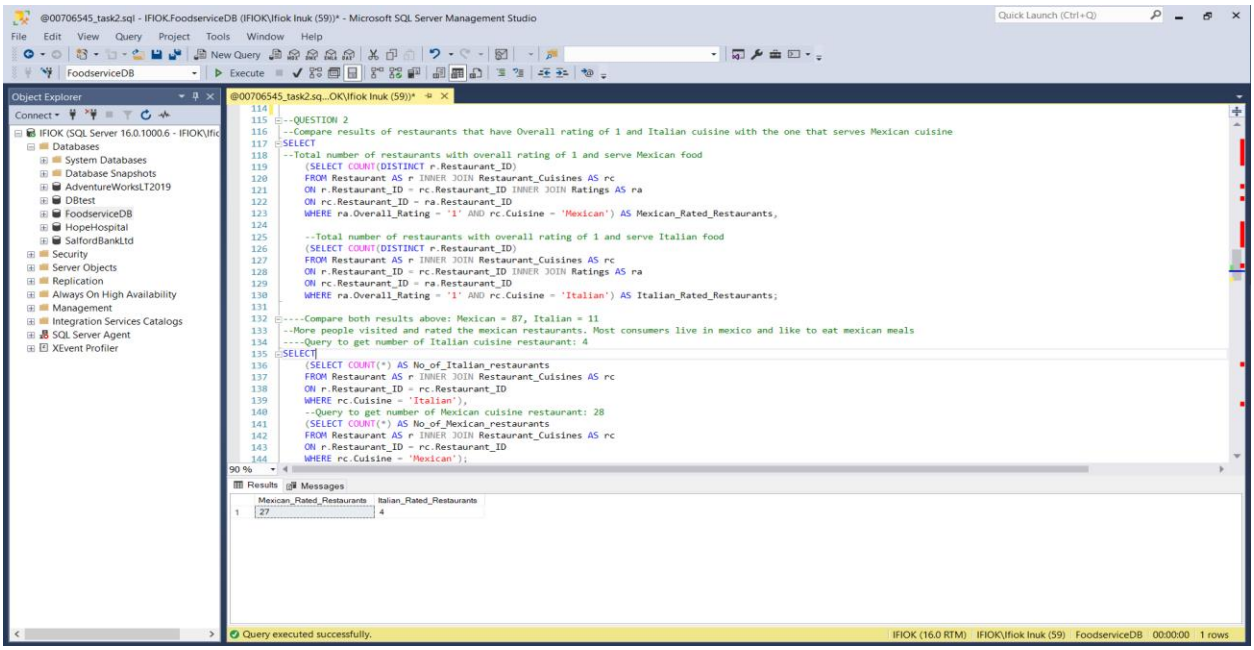

Figure 69: Analysis of the restaurants with rating of 1

Question 3

The statement below shows the calculation of the average age of consumers who have given a 0 Service rating. A SELECT query was used to return the result of an INNER JOIN between the Consumers and Ratings table. The tables were joined on the Consumer_ID column and filtered using the WHERE clause for Service rating of 0. The Average of the Age column was returned using the AVG() aggregate function. The ROUND() function with a value of 0 was also used to round off the Age to the nearest whole number. The result of the query is that the average age of the consumers who give 0 service rating is 26.

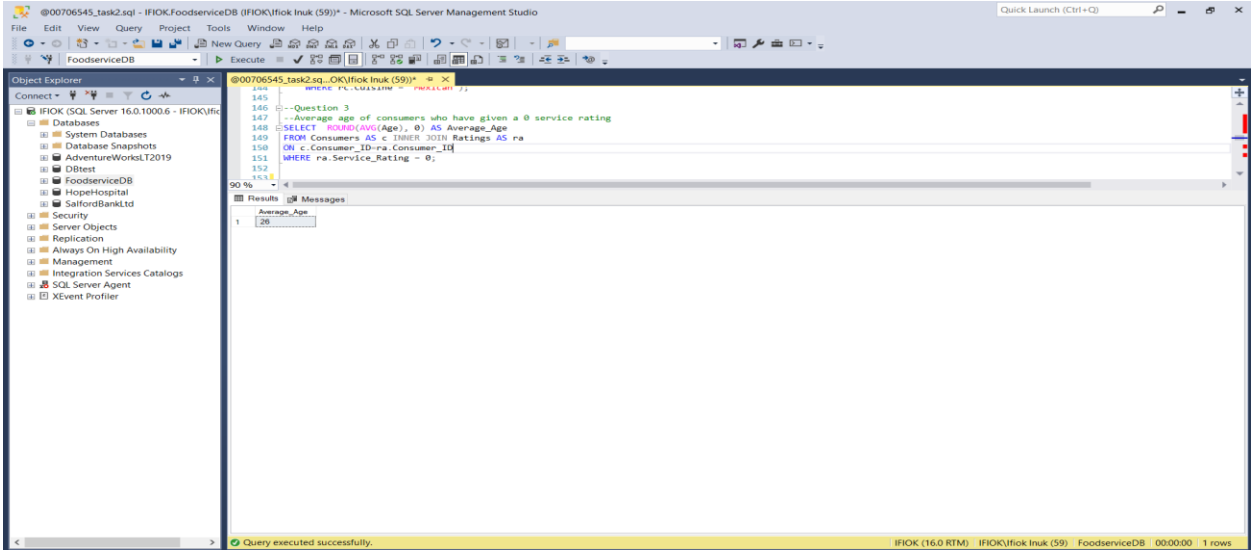

Figure 70: Average age of the consumers who give 0 service rating

Question 4

The statement below returns the list of restaurants rated by the youngest consumer.

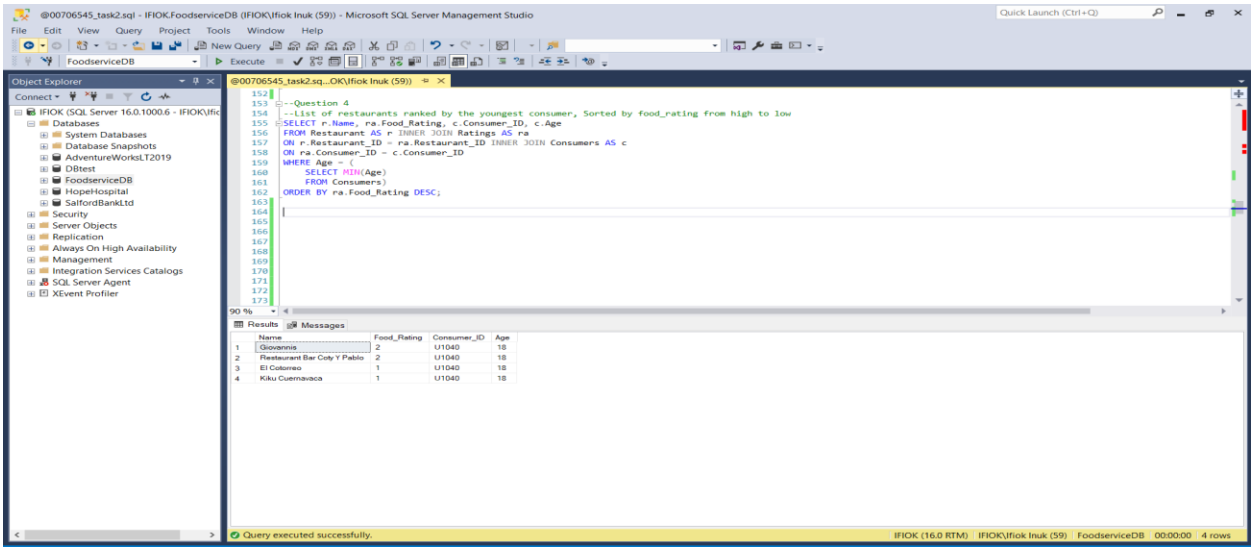

Figure 71: Restaurants rated by the youngest consumer

Question 5

Before creating and executing the stored procedure, I counted the number of restaurants that did not have a service rating of 2 but had parking as yes or public, which was 376.

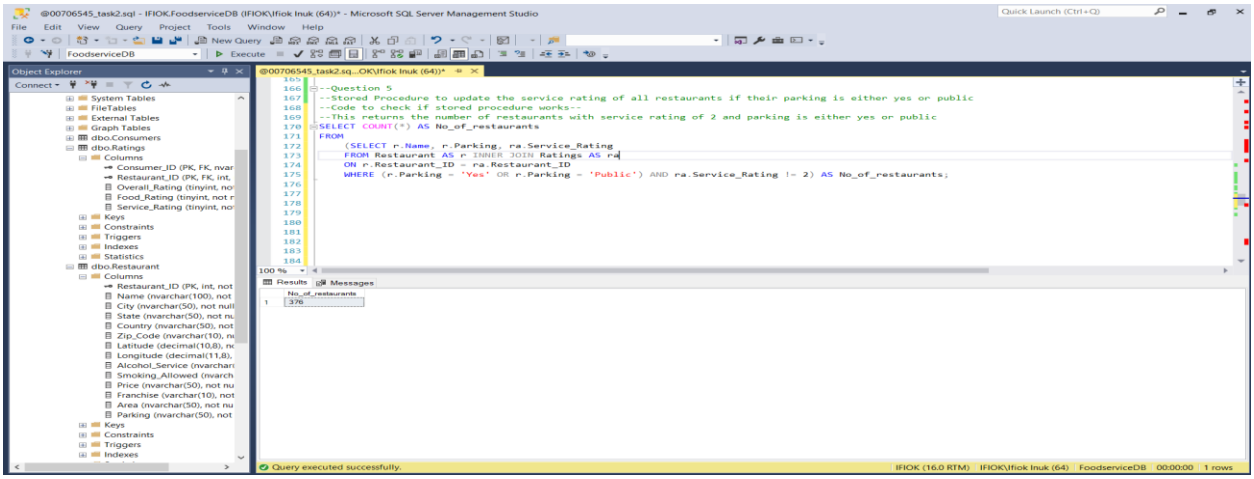

Figure 72: Test query before executing the procedure

The stored procedure for updating the service rating of these restaurants to 2 was created as shown below.

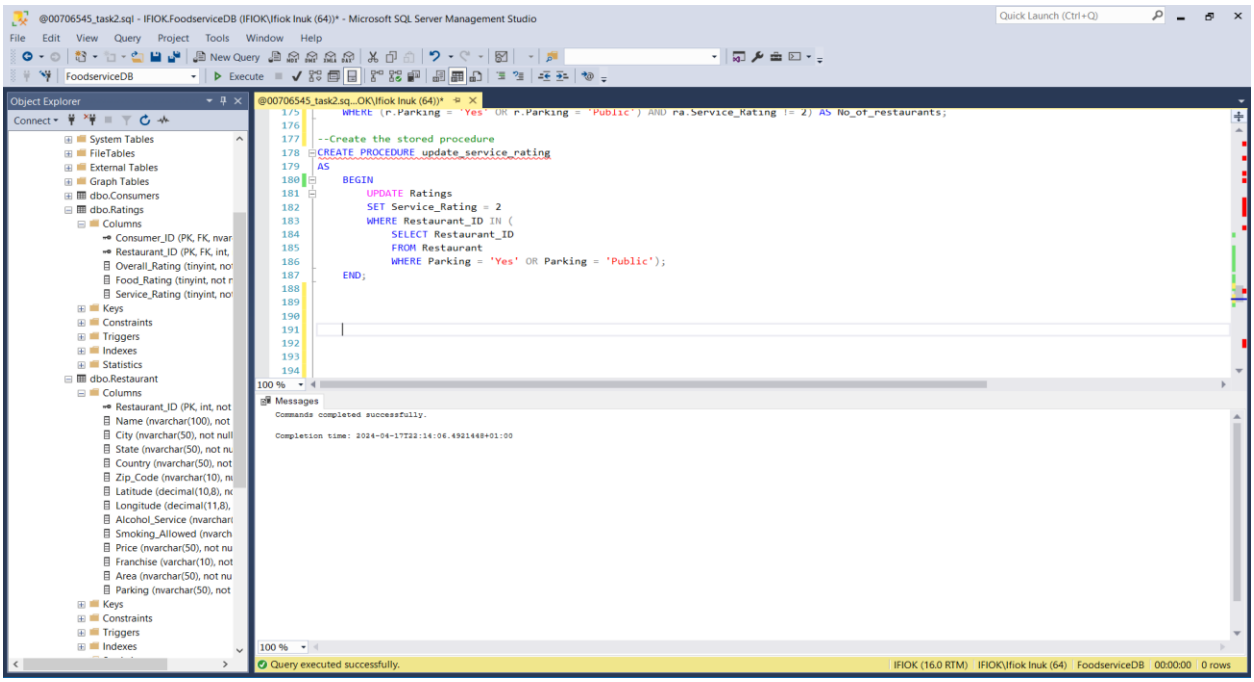

Figure 73: Creation of the procedure

The procedure was executed as seen below.

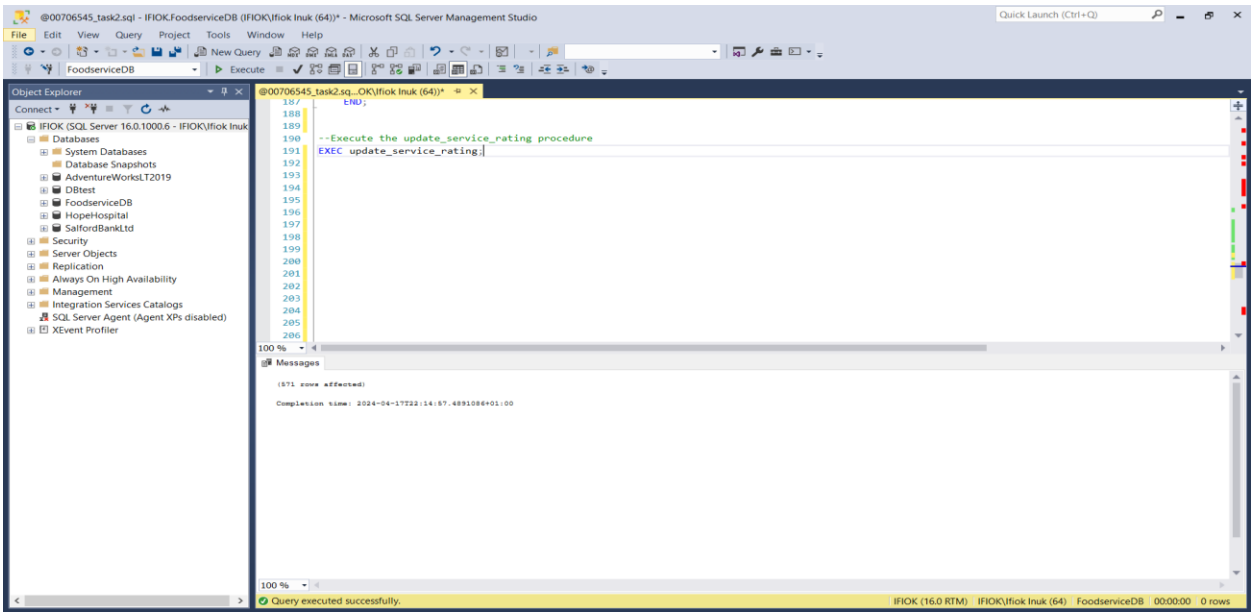

Figure 74: Execution of the procedure

To confirm that the procedure executed correctly, the test query was executed again. The result showed 0, meaning all the restaurants were updated.

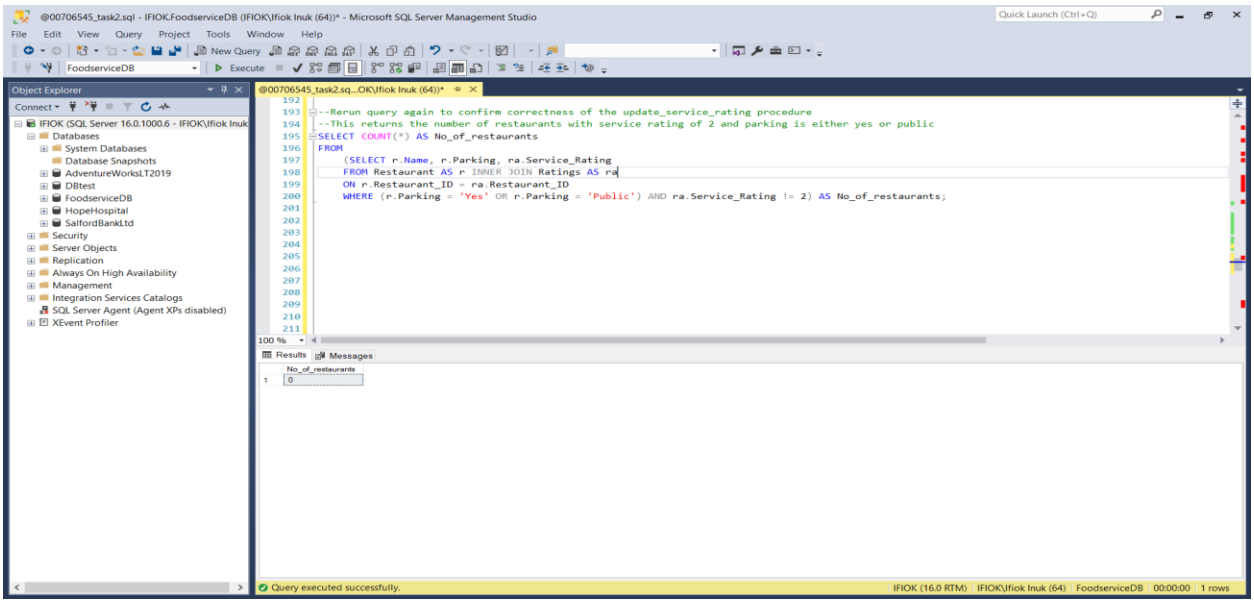

Figure 75: Test query after executing the procedure

Question 6 - PERSONAL QUERIES

Question 6a

A stored procedure was created to return a restaurant's ratings. The procedure accepted an argument (@restaurant data type int). clause was added to filter for the restaurant id that was passed to the procedure.

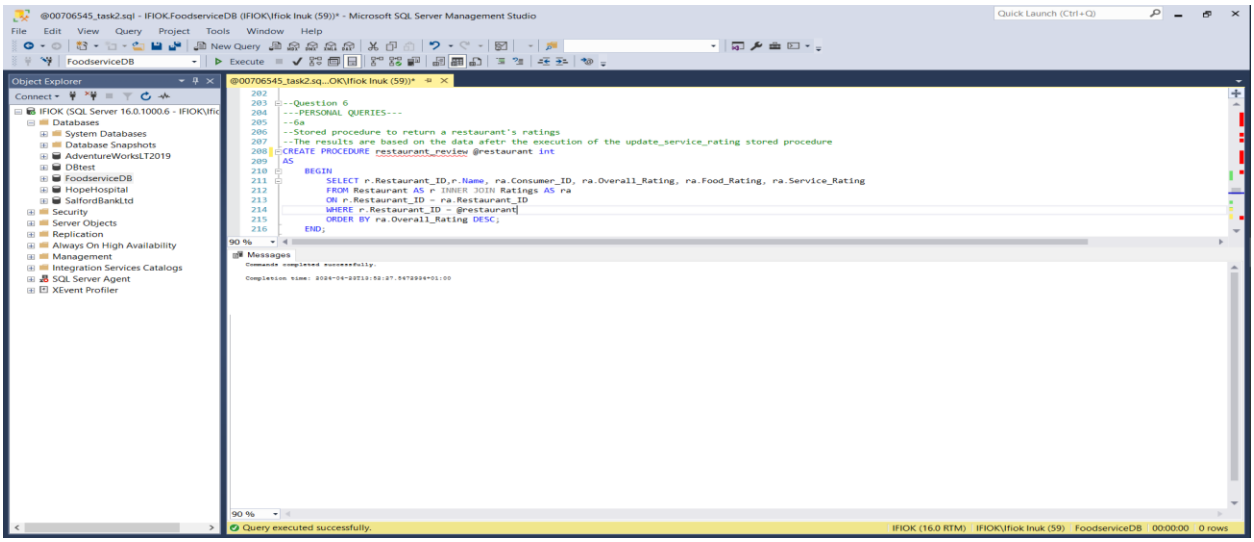

Figure 76: Creation of the restaurant_review procedure

The procedure was executed for two restaurants with ids of 132560 and 132583.

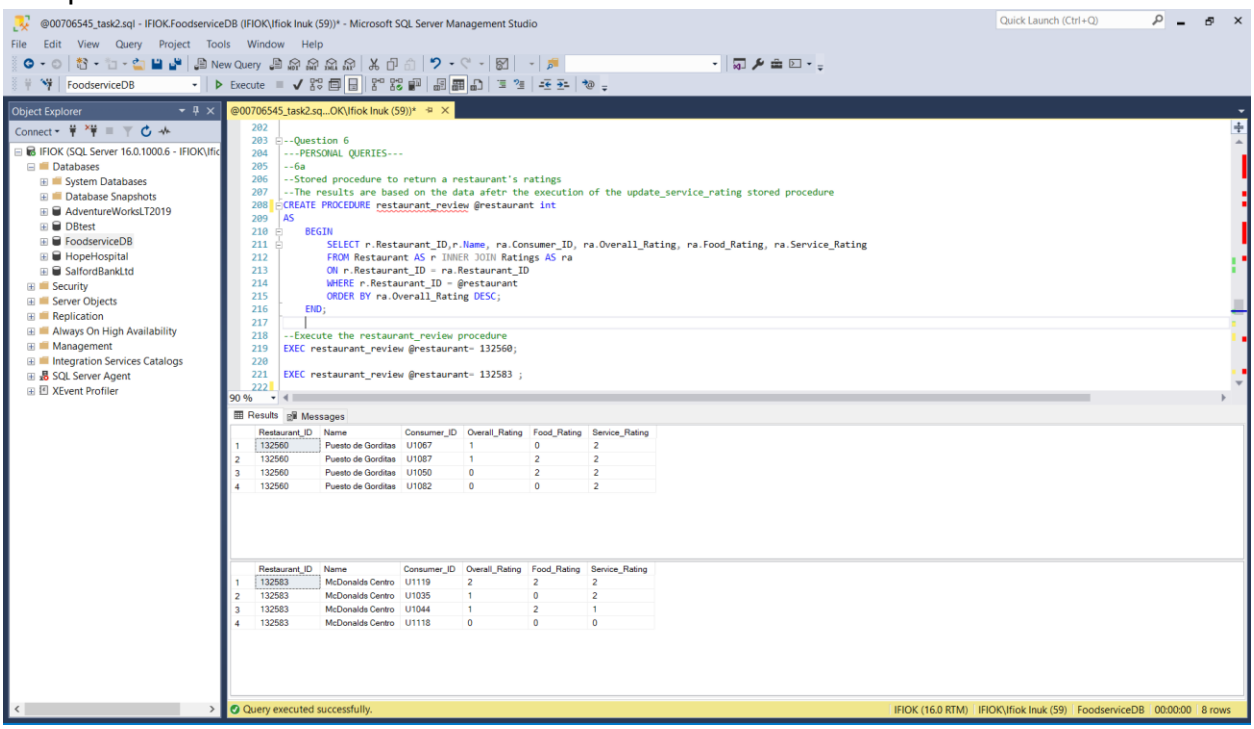

Figure 77: Creation of the restaurant_review procedure for restaurants with ids of 132560 and 132583

<u>Question 6b</u>

The statement below returns a list of the restaurants that need to have a kid's menu. We have 50 restaurants that need to have a kid's menu.

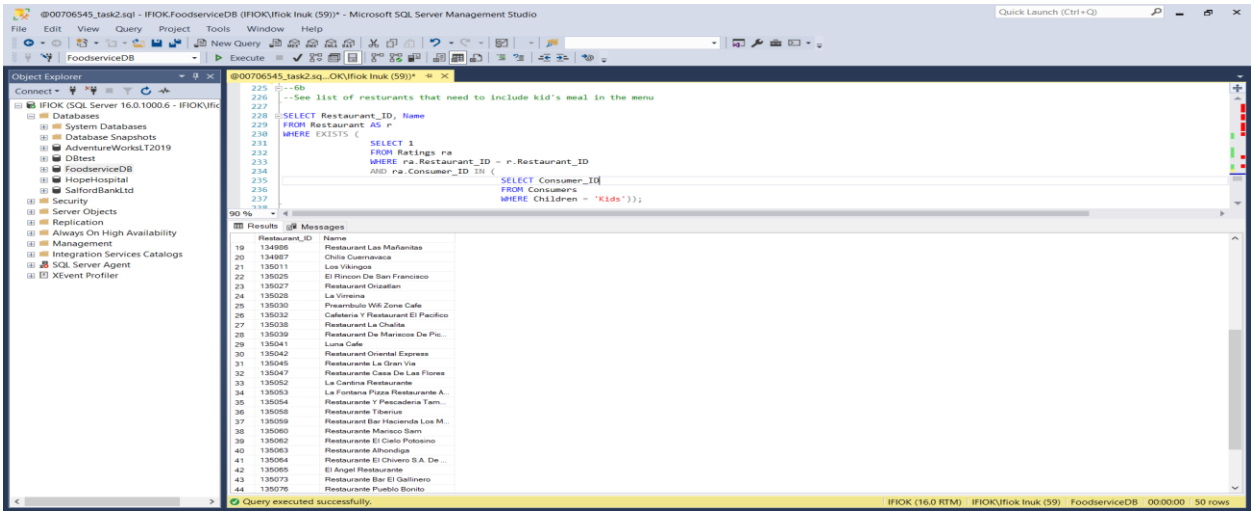

Figure 78: List of the restaurants that need to have kid's menu

Question 6c

The statement below returns a list of the restaurants that have good food rating but poor service rating and need to improve on service. The result shows that there are 4 restaurants with good food but poor service.

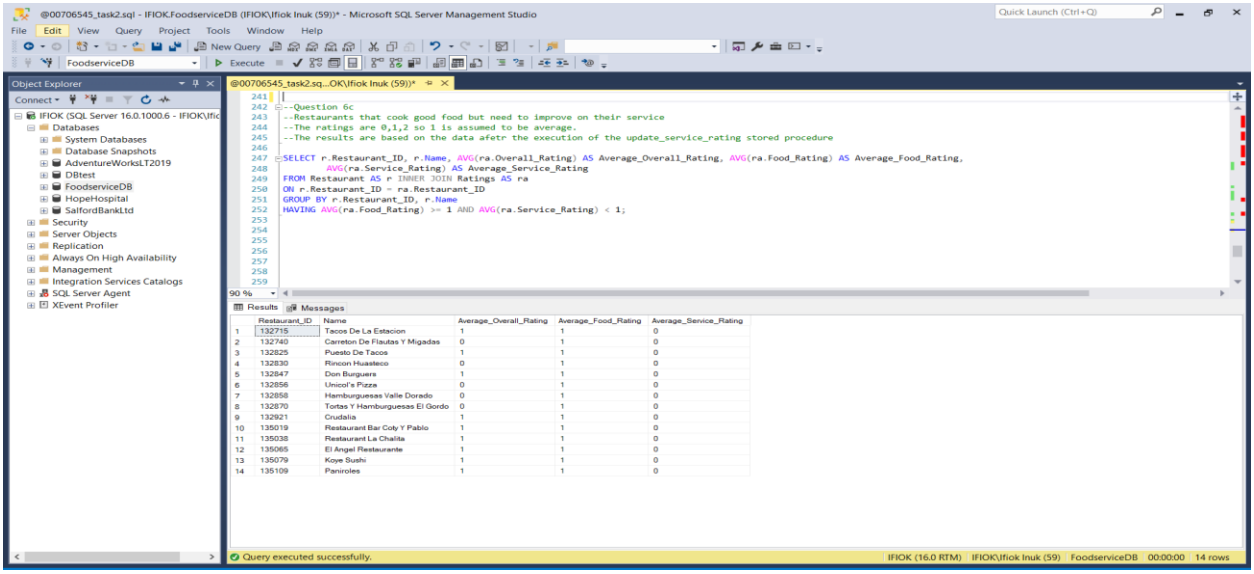

Figure 79: List of the restaurants that have good food rating but poor service rating

Question 6d

The statement below counts the number of smokers that use the smoking allowed restaurants. The results show that only 6 smokers use the smoking allowed restaurants so we can assume that this is not important factor to consider when improving a restaurant's service.

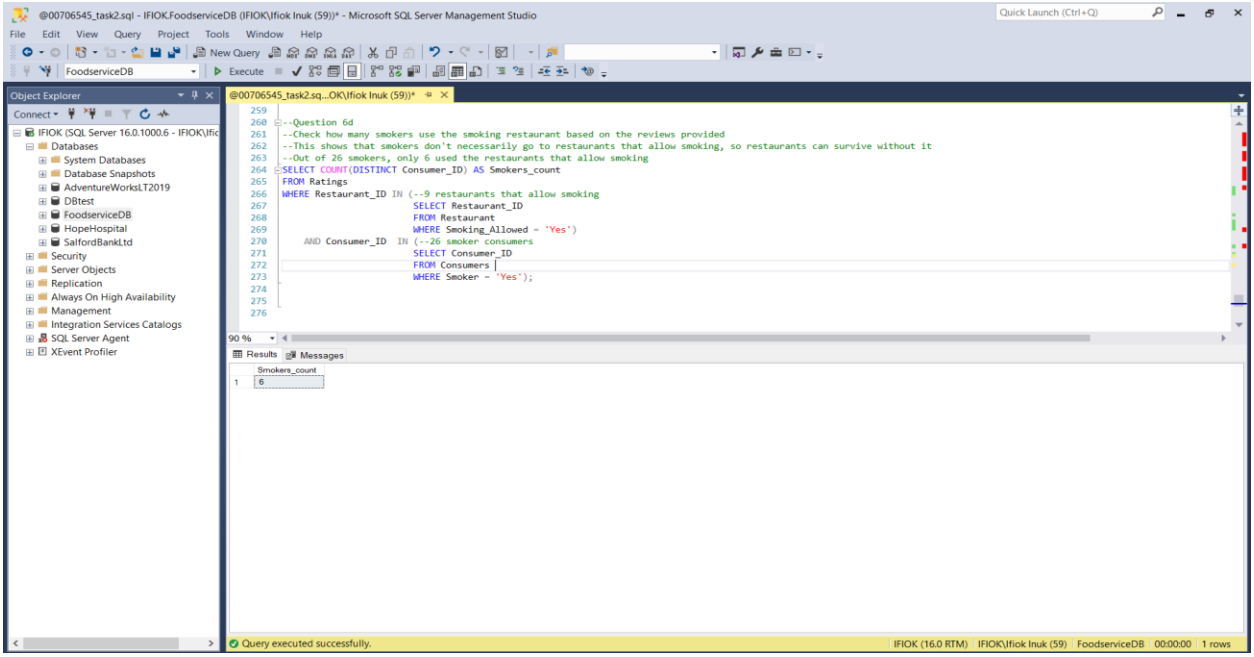

Figure 80: Number of smokers that use the smoking-allowed restaurants

Conclusion

The FoodServiceDB database was developed to analyze the food service data in Mexico. The database has four tables that contain data on the restaurants, consumers, consumer's ratings and cuisines in the appropriate format. The program for this database was written using a method that allows reusability for updated versions of the imported csv files.

The database ensures that queries can be executed to gain valuable insights into restaurants' ratings and the impact of the cuisines on the consumers. These insights will impact the business decisions that the company will make and help improve its profitability.