TASK 1: HOSPITAL(DATABASE) MANAGEMENT SYSTEM

My main goal as a database development consultant for the Hospital Management System is to create and put into place a reliable database system that is suited to a hospital's particular requirements. To improve patient care and streamline hospital operations, this system will manage patient records, doctor information, appointment scheduling, medical histories, and other critical data points.

FEATURES AND COMPONENTS OF DATABASES:

To enable effective data management and manipulation, our solution will include a variety of database objects, such as stored procedures, user-defined functions, views, and triggers. These items will have painstaking design to guarantee security, usability, and data integrity.

METHOD OF STRUCTURED DEVELOPMENT:

The Hospital Management System database will be developed using an organized methodology that includes the following stages:
1. Requirement Analysis: Comprehending the requirements and features that the hospital demands.
2. Database Design: Constructing an effective database schema to arrange healthcare data.
3. Implementation: Constructing stored procedures, tables, views, and other database objects.
4. Testing: Extensive testing to guarantee that the system operates as anticipated in a range of conditions.
5. Optimization: Improving the database's scalability and performance by fine tweaking.

NORMALIZATION: THIRD NORMAL FORM (3NF) EXPLANATION

In order to lessen data redundancy and enhance data integrity, normalization is a database architecture approach that groups data attributes into distinct tables. Normalization is to ensure that the data is stored correctly and to remove redundant data. Normal forms are a collection of guidelines that are applied during the normalization process. One such form that expands upon the first and second normal forms is the Third Normal Form (3NF).

The Hospital Management System database does 3NF in the following ways:

1. First Normal Form (1NF), A primary key exists in every table:
DepartmentID in Departments
DoctorID in Doctors
ScheduleID in Doctors Availability
PatientID in Patients
AppointmentID in Appointments
MedicalRecordID in Medical Records
PrescriptionID in Prescription

MRD_ID in MedicalRecords_Diagnosis
MRA_ID in MedicalRecords_Allergy
All values in the column are atomic, or indivisible neither arrays nor repeated groupings are present.

2. Normal Second Form (2NF)
A table must be in 1NF in order to be in 2NF, Every non-key attribute must to be totally dependent on the primary key in order to function.

Departments: `DepartmentName` and `DepartmentID` rely on `DepartmentID` for full functionality.

Doctors: `FirstName`, `MiddleName`, `LastName`, `Specialty`, `Email`, `PhoneNumber`, and `DepartmentID` are among the attributes that uniquely identify each row; they are completely functionally dependent on `DoctorID`.

Doctors Availability: Every row is uniquely identified by its `ScheduleID`, and all other table elements—`DoctorID`, `ScheduleDate`, `StartTime`, and `EndTime`—are completely functionally dependent on it.

Patients: As the primary key, `PatientID` is completely functionally dependent on attributes such as `FirstName`, `MiddleName`, `LastName`, `DateOfBirth`, `Address`, `PhoneNumber`, `EmailAddress`, `InsuranceNumber`, `Username`, `Password`, `RegistrationDate`, and `LeaveDate`.

Appointments:`AppointmentID` is the main key, and all other attributes, including `PatientID`, `DoctorID`, `DepartmentID`, `AppointmentDate`, `AppointmentTime`, `Review`, and `Status`, rely on it completely for their functionality.

3. Normal Form (3NF): A table must be in 2NF in order to be in 3NF.
Non-key attributes shouldn't be transitively dependent on the main key.

Medical Records: - The primary key is `MedicalRecordID`, and attributes with direct dependencies on it include `PatientID`, `DoctorID`, and `AppointmentDate`.

Prescription: The primary key is `PrescriptionID`, and attributes with direct dependencies on it include `MedicalRecordID`, `MedicineName`, `MedicinePrescribedDate`, `Dosage`, `Frequency`, `StartDate`, and `EndDate.

MedicalRecords_Diagnosis: There are no transitive dependencies between `MRD_ID` and `Diagnosis`; instead, they are directly dependent on one another.

MedicalRecords_Allergy: There is no transitive dependency between `MRA_ID` and `Allergy`; instead, they are directly dependant on one another.

T-SQL

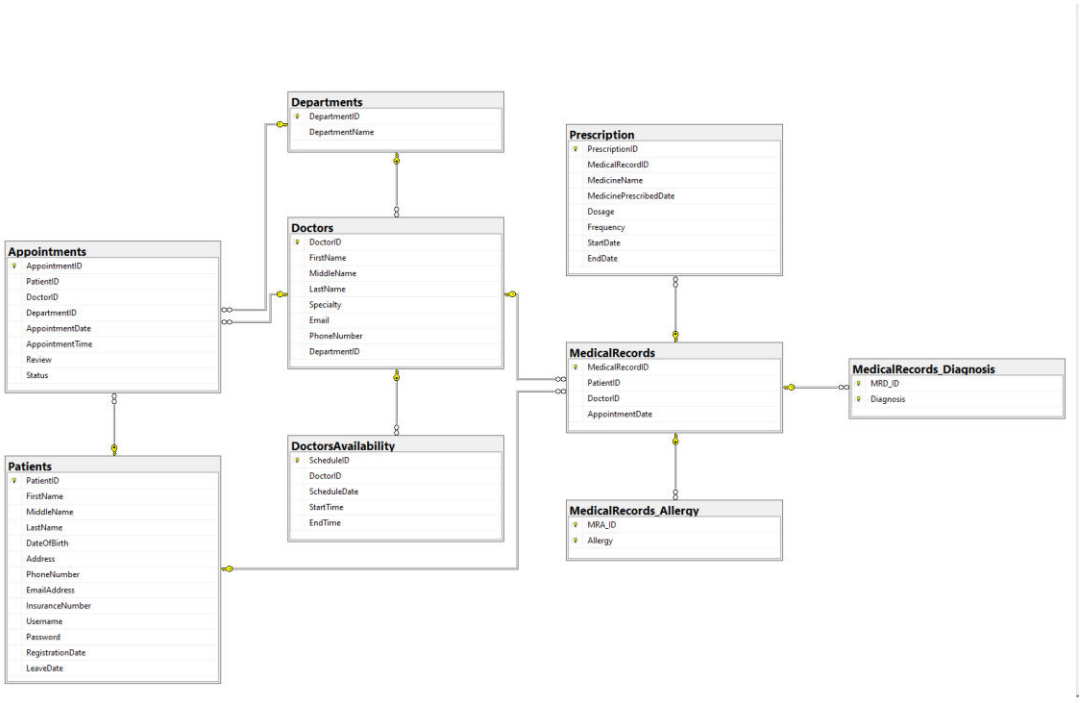
T-SQL, a Microsoft programming extension, enhances SQL Server's capabilities, including row processing, declared variables, transaction control, and exception handling. It is used by all applications interacting with SQL Server, including queries, tables, constraints, views, stored procedures, columns, and data types. T-SQL IDs, assigned at object creation time, are unique and assigned to objects, ensuring their uniqueness.

T-SQL function types consist of:

Statements in T-SQL are the common language and for every program that communicates with SQL Server. T-SQL queries can use the choose statement, restrict rows, choose columns, label output columns, and change search conditions.

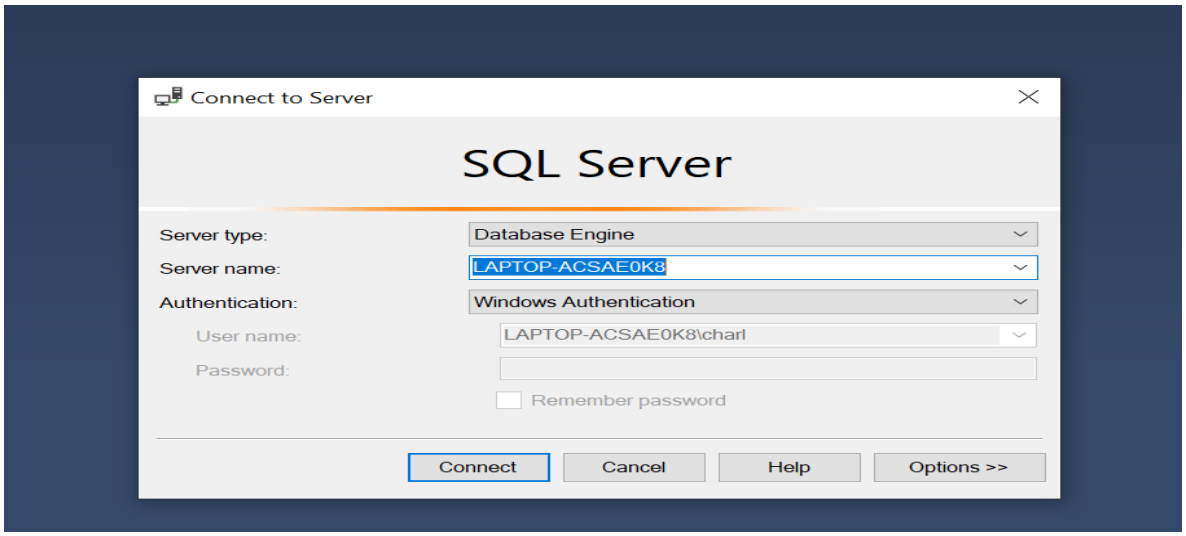
Data stored in Microsoft SQL Server databases can be managed and altered using the database programming language T-SQL (Transact-SQL).

ENTITY RELATION DIAGRAM FOR HOSPITAL MANAGEMENT SYSTEM

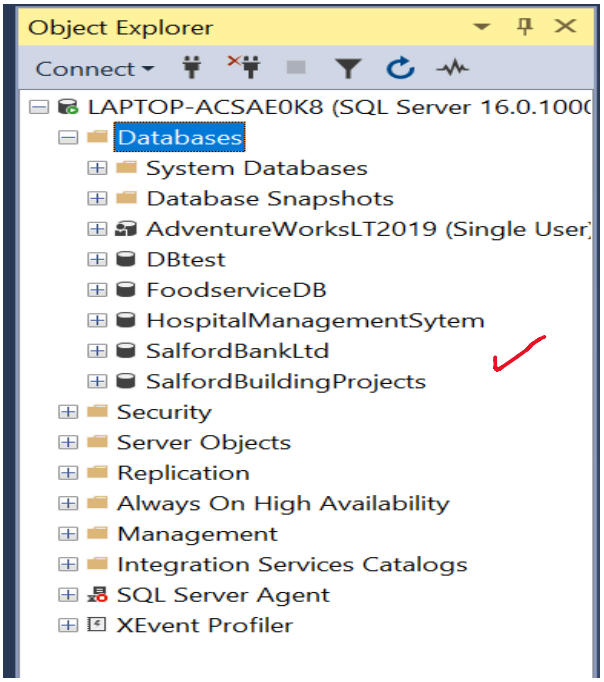


DATABASE CREATION:

Connecting the SQL Server to the local system's server name is the first step. I have linked my SQL server to my server, which is called "LAPTOP-ACSAE0K8." The creation of the database comes next after the server is connected. I gave the database the name ‘HOSPITALMANAGEMENTSYSTEM’ when I established it.



Connect the newly constructed database to run the following query after it was created. After refreshing the Object Explorer, the created database should show up in the left panel.



PART-1: DESIGN AND NORMALISE THE PROPOSED DATABASE INTO 3NF.

To create and normalize the proposed database into 3NF, we need to take the following steps:

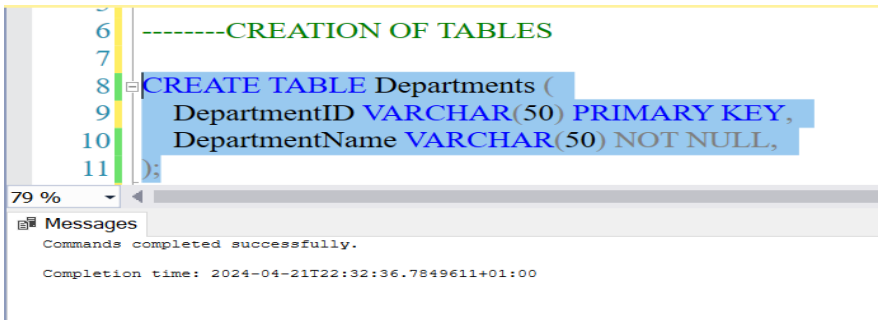
- Identify the entities and the links between them.
- Create a table for each object.
- Indicate the primary keys for every table.
- Define foreign keys to establish links between tables.
- Normalize the tables using 3NF.
- Implement limitations to ensure the accuracy of the data.

CREATING THE TABLES AND DEFININD DATATYPES

DEPARTMENTS TABLE:

The Departments table is a 3NF design, with a DepartmentID as the primary key and a DepartmentName as the department name. It is free of repeating groups, functional dependencies, and transitive dependencies. The main key (DepartmentID) is the only dependent attribute, and every property is directly dependent on it. This design reduces data redundancy and ensures data integrity. The data types used for each field are suitable for the data they store, ensuring efficiency and accuracy in data representation and storage.

T-SQL CODE



INSERTING VALUES TO THE DEPARTMENT TABLE

Medical specialties inside a hospital can be identified and categorized with the use of the INSERT INTO Departments statement, which adds initial department information to the Departments table. It inserts two values for each department, along with distinct department names and IDs. Standard names facilitate database maintenance for users and developers, while department IDs and names are alphanumeric codes that are used to rapidly reference each department.

T-SQL IMPLEMENTATION

```
--------POPULATING TABLES
-- Insert statements for Departments table
INSERT INTO Departments (DepartmentID, DepartmentName) VALUES
('DR1', 'Cardiology'),
('DR2', 'Neurology'),
('DR3', 'Orthopedics'),
('DR4', 'Pediatrics'),
('DR5', 'Oncology'),
('DR6', 'Gynecology'),
('DR7', 'Gastroenterology');
```

79 %

Messages

```
(7 rows affected)
Completion time: 2024-04-21T23:22:37.5597851+01:00
```

DOCTORS TABLE:

The Third Normal Form (3NF) for the Doctors table follows normalization guidelines to minimize redundancy and guarantee data integrity. Every column has atomic, indivisible values, and every record has a distinct primary key. All non-key properties are operationally dependent on the primary key, which is a composite key with 'DoctorID'. Moreover, there are no transitive connections between non-key characteristics and the primary key in the table. VARCHAR (50), NVARCHAR (100), NVARCHAR (50) are the data types that are utilized. This guarantees accuracy and efficiency when storing data.

T-SQL CODE

```
CREATE TABLE Doctors (
    DoctorID VARCHAR(50) PRIMARY KEY,
    FirstName NVARCHAR(100) NOT NULL,
    MiddleName NVARCHAR(100),
    LastName NVARCHAR(100) NOT NULL,
    Specialty VARCHAR(50) NOT NULL,
    Email VARCHAR(100) DEFAULT NULL,
    PhoneNumber  VARCHAR(50) NOT NULL,
    DepartmentID VARCHAR(50) NOT NULL,
    FOREIGN KEY (DepartmentID) REFERENCES Departments (DepartmentID)
);
```

79 %

Messages

```
Commands completed successfully.
Completion time: 2024-04-21T22:34:44.9245196+01:00
```

INSERTING VALUES TO THE DOCTORS TABLE:

Several rows of doctor data, each representing a distinct doctor's details, are inserted into the Doctors table by this statement. Based on the supplied values, the INSERT INTO statement puts data into the appropriate columns of the Doctors table. A foreign key that links to the Departments table is used to store the data in the Doctors table. The statement for handling doctor data is clear-cut and effective.

T-SQL IMPLEMENTATION

```
-- Insert statements for Doctors table
INSERT INTO Doctors (DoctorID, FirstName, MiddleName, LastName, Specialty, Email, PhoneNumber, DepartmentID) VALUES
('D10','Adim','James','Johnson','Orthopedic Surgeon','adimjohnson@yahoo.com','345-678-9045','DR3'),
('D11','Emily','NULL','Okem','Pediatrician','emilyoke@yandex.com','456-789-0176','DR4'),
('D12','David','Charles','Wilson', 'Oncologist','davidwilson@ru.com','567-890-1234','DR5'),
('D13','Oge','Maria','Martinez','Gynecologist','ogemartinez@gmail.com','678-901-2354','DR6'),
('D14','Martha','NULL','Taylor','Gastroenterologist','mmataylor@outlook.com','789-012-3478','DR7'),
('D8','Nkechi','Edaward','Kalu','Cardiologist','nkelu@gmail.com  ','123-456-7999','DR1'),
('D9','Alice','NULL','Mba','Neurologist','alicemba@gmail.com','234-567-8934','DR2');

79 %
Messages
(7 rows affected)
Completion time: 2024-04-21T23:23:31.4300388+01:00
```

DOCTORS AVAILABILITY TABLE:

The Third Normal Form (3NF) guarantees that all characteristics in a database are only functionally reliant on the primary key. ScheduleID, a distinct identity for every availability schedule, is the main key in the Doctors Availability table. The ScheduleID serves as the functional basis for the schedule Date and start/end timings, and the DATE and TIME datatypes are appropriate for storing precise times.

T-SQL CODE

```
CREATE TABLE DoctorsAvailability (
ScheduleID VARCHAR(50) PRIMARY KEY,
DoctorID VARCHAR(50) NOT NULL,
ScheduleDate DATE NOT NULL,
StartTime TIME NOT NULL,
EndTime TIME NOT NULL,
FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

Messages
Commands completed successfully.
Completion time: 2024-04-21T23:08:23.0396549+01:00
```

INSERTING VALUES TO THE DOCTORS Availability:

Values with fields for ScheduleID, DoctorID, ScheduleDate, StartTime, and EndTime is inserted into the DoctorsAvailability table with the SQL query. ScheduleDate indicates the available date, while schedule ID is the distinct identification for every availability schedule. The SQL statement uses distinct ScheduleIDs for simple identification and maintenance, as well as ScheduleDate, StartTime, and EndTime for effective appointment scheduling.

T-SQL IMPLEMENTATION

```
-- Insert statements for DoctorsAvailability table
INSERT INTO DoctorsAvailability (ScheduleID, DoctorID, ScheduleDate, StartTime, EndTime) VALUES
('S8', 'D8', '2024-04-21', '08:00:00', '20:00:00'),
('S9', 'D9', '2024-04-22', '08:00:00', '20:00:00'),
('S10', 'D10', '2024-04-23', '08:00:00', '20:00:00'),
('S11', 'D11', '2024-04-24', '08:00:00', '20:00:00'),
('S12', 'D12', '2024-04-25', '08:00:00', '20:00:00'),
('S13', 'D13', '2024-04-26', '08:00:00', '20:00:00'),
('S14', 'D14', '2024-04-27', '08:00:00', '20:00:00');

Messages
(7 rows affected)
Completion time: 2024-04-21T23:55:51.5118337+01:00
```

PATIENTS TABLE:

The Third Normal Form (3NF) ensures that all non-prime characteristics in a database are functionally dependent on the primary key. The table of patients has three data types: INT, NVARCHAR (50), and NVARCHAR (50). The INT data type ensures unique identification, while the NVARCHAR data type holds the patient's first, middle, and last name. The patient's ID determines the data types and password. The Patient's table is in the Third Normal Form (3NF) for efficient data management and security.

T-SQL CODE

```
CREATE TABLE Patients (
    PatientID INT IDENTITY(1,1) PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    MiddleName NVARCHAR(50),
    LastName NVARCHAR(50) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Address NVARCHAR(200) NOT NULL,
    PhoneNumber VARCHAR(15) NOT NULL,
    EmailAddress NVARCHAR(100) NOT NULL,
    InsuranceNumber VARCHAR(50) NOT NULL,
    Username VARCHAR(50) NOT NULL,
    Password VARCHAR(50) NOT NULL,
    RegistrationDate DATE NOT NULL,
    LeaveDate DATE NULL
);
```

```
Messages
Commands completed successfully.
Completion time: 2024-04-21T22:37:30.4570334+01:00
```

INSERTING VALUES TO THE PATIENTS:

The SQL INSERT query inserts records into the Patients table such as first name, middle name, last name, birthdate, address, phone number, email address, insurance number, login username, password, registration date, and departure date. Data for specified columns can be found using the VALUES keyword. Any set of values included in parenthesis represents a single entry. For real-world situations, users and passwords ought to be more complicated and safer. Every entry has the Leave Date set to NULL, meaning that no patient has departed yet.

T-SQL IMPLEMENTATION

```
-- Insert statements for Patients table
INSERT INTO Patients (FirstName, MiddleName, LastName, DateOfBirth, Address, PhoneNumber, EmailAddress, InsuranceNumber, Username, Password, RegistrationD
('Ifeanyi', 'Ben', 'Clark', '1975-01-01', '123 Main St', '123-456-7890', 'ifyclark@gmail.com', '123456789', 'ify123', 'password', '2022-01-01', NULL),
('Onyi', NULL, 'Ore', '1970-05-05', '764 Oak St', '234-567-8901', 'onyimoore@yaoo.com', '234567890', 'ony456', 'password123', '2022-02-01', NULL),
('Sophia', 'Rosa', 'Nkem', '1971-10-10', '876 Elm St', '345-678-9012', 'sophiankem@hotmail.com', '345678901', 'sia789', 'password456', '2022-03-01', NULL),
('Ethan', NULL, 'Emeh', '1965-03-15', '202 Pine St', '456-789-0123', 'ethanemeh@yandex.com', '456789012', 'etha01', 'password789', '2022-04-01', NULL),
('Helen', 'Grace', 'Chidi', '1978-06-20', '203 Cedar St', '567-890-1234', 'helen@gmail.com', '567890123', 'helbella202', 'passwordabc', '2022-05-01', NULL),
('Andrew', NULL, 'Ope', '1976-09-25', '303 Maple St', '678-901-2345', 'andrew@outlook.com', '678901234', 'andy303', 'passworddef', '2022-06-01', NULL),
('Benjamin', 'Nnamdi', 'Okoye', '1970-12-30', '404 Walnut St', '789-012-3456', 'aben@korumail.com', '789012345', 'okoye404', 'passwordegf', '2022-07-01', NULL),
('Ada','Gregg','Lark','1970-01-01','13 Min St','123-446-7690','adalark@gmail.com','123t54789','ada123','passweed','2021-01-01',NULL),
('Oni','NULL', 'Ere','1965-05-05','74 Tak St','224-547-8901','onimre@yahoo.com', '234563850','on1436', 'paword123','2012-02-01',NULL),
('Sapia ','Osa','Ikem','1973-10-10','76 Elem St','345-648-9032','saphiaikem@hotmail.com','345279901','sai789','passord426','2019-03-01',NULL),
('Ehian','NULL', 'Umeh','1968-03-15','02 Pinet St','456-729-1123','ehianumeh@yandex.com','457u89212','ethi01','password229','2018-04-01',NULL),
('Heen ','Gace','Chida','1972-06-20','20 Cesri St','567-290-1134', 'heen@gmail.com','5672rr113','heballa202','passwordbc','2016-01-01',NULL),
('Andy','NULL','Wope','1977-09-25','33 Male St','678-901-2345','andy@outlook.com','6749ed234','andy303','passworddef','2021-06-01',NULL),
('Bemin', 'Nadi','Ukoye','1972-12-30','40 Ulnut St','789-012-3456','ben@korumail.com','789kk2345','ukoye404','pas33ordegf','2004-07-01',NULL);
```

```
Messages
(14 rows affected)
Completion time: 2024-04-22T00:00:32.3830370+01:00
```

APPOINTMENTS TABLE:

In a 3NF table, every column depends on the primary key, ensuring data is unique and not recurring. The Appointments table has atomic values, eliminating any attributes not reliant on the main key. The primary key is used to function non-key attributes like PatientID, DoctorID, DepartmentID, AppointmentDate, AppointmentTime, Review, and Status. The types of data for scheduled meetings include the ID of Appointment (VARCHAR(50), a special number assigned to every appointment), Patient Identification Number (INT), Doctor ID (VARCHAR(50), Department ID (VARCHAR(50), Date of Appointment), Time of Appointment (Time), Examine (NVARCHAR(MAX), Patient Review). This ensures that each non-key attribute functions based on the primary key, ensuring a consistent and efficient data structure.

T-SQL CODE

```
55  CREATE TABLE Appointments (
56      AppointmentID VARCHAR(50) PRIMARY KEY,
57      PatientID INT,
58      DoctorID VARCHAR(50),
59      DepartmentID VARCHAR(50),
60      AppointmentDate DATE NOT NULL,
61      AppointmentTime TIME NOT NULL,
62      Review NVARCHAR(MAX) NOT NULL,
63      Status VARCHAR(20) CHECK (Status IN ('Pending', 'Cancelled', 'Completed')),
64      FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
65      FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID),
66      FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
67  );
```

```
9 %
Messages
Commands completed successfully.

Completion time: 2024-04-21T22:39:41.0507194+01:00
```

INSERTING VALUES TO THE APPOINTMENTS TABLE:

A new record with the appointment's date, time, review, and status is inserted into the 'Appointments' table via the INSERT INTO statement. Ensuring correct and current information, the INSERT INTO statement inserts these new entries into the 'Appointments' table.

T-SQL IMPLEMENTATION

```
159 INSERT INTO Appointments (AppointmentID, PatientID, DoctorID, DepartmentID, AppointmentDate, AppointmentTime, Review, Status) VALUES
160 ('AP1', 12, 'D8', 'DR1', '2024-04-27', '08:00:00.0000000', 'Good service', 'Completed'),
161 ('AP2', 13, 'D10', 'DR3', '2024-04-24', '19:00:00.0000000', 'Excellent', 'Completed'),
162 ('AP3', 14, 'D14', 'DR7', '2024-04-25', '18:00:00.0000000', 'Satisfied', 'Completed'),
163 ('AP4', 15, 'D9', 'DR2', '2024-05-25', '16:00:00.0000000', 'Very good', 'Pending'),
164 ('AP5', 16, 'D12', 'DR5', '2024-04-23', '12:00:00.0000000', 'Excellent', 'Pending'),
165 ('AP6', 17, 'D14', 'DR7', '2024-04-23', '13:00:00.0000000', 'Satisfied', 'Completed'),
166 ('AP7', 18, 'D9', 'DR2', '2024-05-01', '09:00:00.0000000', 'Satisfied', 'Cancelled'),
167 ('AP8', 19, 'D10', 'DR3', '2024-05-16', '10:00:00.0000000', 'Very good', 'Cancelled'),
168 ('AP9', 20, 'D11', 'DR4', '2024-04-23', '11:00:00.0000000', 'Not good', 'Completed'),
169 ('AP10', 21, 'D12', 'DR5', '2024-04-22', '12:00:00.0000000', 'Excellent', 'Completed'),
170 ('AP11', 22, 'D13', 'DR6', '2024-04-22', '13:00:00.0000000', 'Very good', 'Cancelled'),
171 ('AP12', 23, 'D14', 'DR7', '2024-04-22', '14:00:00.0000000', 'Satisfied', 'Completed'),
172 ('AP13', 24, 'D8', 'DR1', '2024-04-22', '16:00:00.0000000', 'Very good', 'Completed'),
173 ('AP14', 25, 'D12', 'DR5', '2024-04-22', '20:00:00.0000000', 'Very good', 'Completed');
```

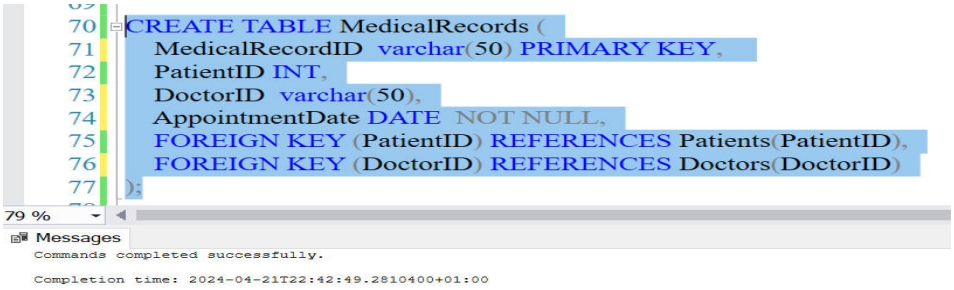
```
9 %
Messages
(14 rows affected)

Completion time: 2024-04-22T00:20:49.8738592+01:00
```

MEDICAL RECORDS TABLE:

The Medical Records Table is a 3NF database design that ensures the removal of unnecessary data and logical ordering of data dependencies. To be in 3NF, a table must meet the requirements of the 2nd Normal Form (2NF) and confirm that non-prime attributes don't depend transitively on the primary key. The table has a primary key (MedicalRecordID) that uniquely identifies each record, satisfying the requirements of 2NF. MedicalRecordID is the table's main key, acting as a special identification code for every medical record. PatientID is an INT data type, representing the patient's ID linked to the medical record. DoctorID is a foreign key that points to the Doctors table. The date of appointment is a date-based data type, ensuring that every medical record always has the appointment date.

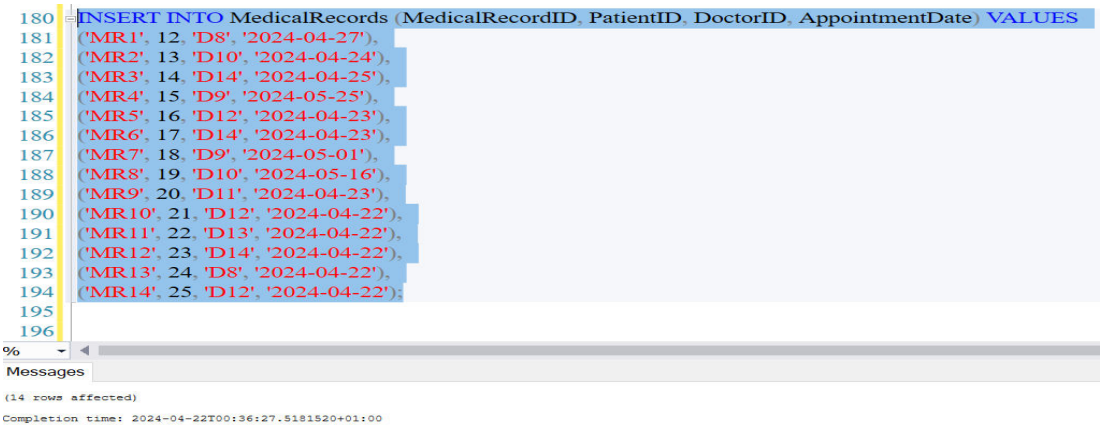
T-SQL CODE



INSERTING VALUES TO THE MEDICAL RECORDS TABLE:

The INSERT statement in the MedicalRecords database ensures the unique identification of medical records, facilitating the retrieval of specific values. The foreign key 'MR1' is used to reference the 'PatientID', a foreign key from the 'Patients' table, and the 'DoctorID', a foreign key from the 'Doctors' table. The 'Date of Appointment' field displays the appointment date, enabling tracking of patient's healthcare visits for specific conditions. This database design ensures data integrity and maintains unique identification for each record, facilitating efficient management and patient care tracking.

T-SQL IMPLEMENTATION



PRESCRIPTION TABLE:

A prescription ID, a foreign key named MedicalRecordID, a Medicine name, the prescribed date, dosage information, the frequency of drug administration, a start date, and an end date are all included in the 3NF Prescription Table Analysis. Variable character strings, or VARCHAR(50), are the data types defined in the table ,MedicalRecordID is a foreign key that references MedicalRecords, whereas prescription ID is the primary key. Name, dosage, frequency, start date, and finish date of the medication are all atomic as well.

T-SQL CODE

```
CREATE TABLE Prescription (
    PrescriptionID VARCHAR(50) PRIMARY KEY,
    MedicalRecordID VARCHAR(50),
    MedicineName VARCHAR(100) NOT NULL,
    MedicinePrescribedDate DATE NOT NULL,
    Dosage VARCHAR(50) NOT NULL,
    Frequency VARCHAR(50) NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    FOREIGN KEY (MedicalRecordID) REFERENCES MedicalRecords(MedicalRecordID)
);
```

```
Messages
Commands completed successfully.

Completion time: 2024-04-21T22:44:14.5344445+01:00
```

INSERTING VALUES TO PRESCRIPTION:

The medical record system, MedicalRecords, is used to track prescriptions for patients. Each prescription is assigned a unique PrescriptionID, which serves as a foreign key to the database. The VALUES clause inserts multiple rows with comparable data, ensuring that each prescription is linked to a specific medical record. This method ensures that every prescription is accurately recorded, retaining relevant information about the prescribed medication. The database ensures that every prescription is linked to a specific medical record.

T-SQL IMPLEMENTATION

```
INSERT INTO Prescription (PrescriptionID, MedicalRecordID, MedicineName, MedicinePrescribedDate, Dosage, Frequency, StartDate, EndDate)
VALUES
('PRA', 'MR1', 'Statins', '2024-04-14', '10mg', 'Once daily', '2024-05-14', '2024-05-31'),
('PRB', 'MR2', 'Antidepressants', '2024-05-15', '20mg', 'Twice daily', '2024-05-15', '2024-05-28'),
('PRC', 'MR3', 'Corticosteroids', '2024-05-16', '15mg', 'Three times daily', '2024-05-16', '2024-05-30'),
('PRD', 'MR4', 'Vaccines', '2024-04-20', '30mg', 'Once daily', '2024-05-17', '2024-06-30'),
('PRE', 'MR5', 'Chemotherapy agents', '2024-06-18', '25mg', 'Twice daily', '2024-06-18', '2024-08-31'),
('PRF', 'MR6', 'Antifungals', '2024-03-21', '40mg', 'Once daily', '2024-07-19', '2024-07-30'),
('PRG', 'MR7', 'Laxatives', '2024-03-22', '35mg', 'Twice daily', '2024-05-07', '2024-08-31'),
('PRH', 'MR8', 'Statins', '2024-04-20', '10mg', 'Once daily', '2024-04-18', '2024-04-30'),
('PRI', 'MR9', 'Antidepressants', '2024-04-20', '20mg', 'Twice daily', '2024-04-18', '2024-05-28'),
('PRJ', 'MR10', 'Corticosteroids', '2024-04-20', '15mg', 'Three times daily', '2024-05-18', '2024-05-30'),
('PRK', 'MR11', 'Vaccines', '2024-04-20', '30mg', 'Once daily', '2024-04-18', '2024-06-30'),
('PRL', 'MR12', 'Antidepressants', '2024-05-25', '20mg', 'Twice daily', '2024-05-25', '2024-05-28'),
('PRM', 'MR13', 'Chemotherapy agents', '2024-03-13', '25mg', 'Twice daily', '2024-07-13', '2024-08-31'),
('PRN', 'MR14', 'Antifungals', '2024-03-29', '40mg', 'Once daily', '2024-07-29', '2024-08-30');
```

```
Messages
(14 rows affected)
Completion time: 2024-04-22T00:39:42.1945120+01:00
```

MEDICALRECORDS_DIAGNOSIS TABLE:

The MedicalRecords_Diagnosis table 3NF satisfies the following requirements: every table possesses a primary key, ensuring unique identity for each row, and no transitive dependencies on the main key. The VARCHAR datatype is selected for its ability to manage variable-length strings and short diagnosis names. The primary key is the combination of Diagnosis and MRD_ID, ensuring individuality and clear identity for each diagnosis report. The foreign key MRD_ID references the MedicalRecordID in the table, linking every diagnosis to an authentic medical record.

T-SQL CODE

```
91
92  CREATE TABLE MedicalRecords_Diagnosis (
93      MRD_ID VARCHAR(50) NOT NULL,
94      Diagnosis varchar(50) NOT NULL,
95      FOREIGN KEY (MRD_ID) REFERENCES MedicalRecords(MedicalRecordID),
96      PRIMARY KEY (MRD_ID, Diagnosis)
97  );
98
```

```
Messages
Commands completed successfully.

Completion time: 2024-04-21T23:00:21.3095404+01:00
```

INSERTING VALUES TO MEDICALRECORDS_DIAGNOSIS TABLE:

For storing diagnosis data pertaining to medical records, the MedicalRecords_Diagnosis table is an essential resource. It has two primary fields: Diagnosis, which stores the related diagnosis, and MRD_ID, which is a foreign key connecting to the medical record. When new records are inserted into the table, the INSERT INTO statement makes sure that every row has a distinct MRD_ID and related Diagnosis. When adding data to the database, data validation is essential to make sure the diagnosis value is accurate and pertinent to the medical records, and that the MRD_ID is present as a foreign key.

T-SQL IMPLEMENTATION

```
218
219  -- Insert statements for MedicalRecords_Diagnosis table
220  INSERT INTO MedicalRecords_Diagnosis (MRD_ID, Diagnosis) VALUES
221  ('MR1', 'Hypertension'),
222  ('MR10', 'Tendinitis'),
223  ('MR11', 'Endometriosis'),
224  ('MR12', 'Neuropathy'),
225  ('MR13', 'Cancer'),
226  ('MR14', 'Endometriosis'),
227  ('MR2', 'Neuropathy'),
228  ('MR3', 'Tendinitis'),
229  ('MR4', 'Asthma'),
230  ('MR5', 'Cancer'),
231  ('MR6', 'Endometriosis'),
232  ('MR7', 'Pancreatitis'),
233  ('MR8', 'Cancer'),
234  ('MR9', 'Neuropathy');
235
```

```
Messages
(14 rows affected)
Completion time: 2024-04-22T00:40:42.6928444+01:00
```

MEDICALRECORDS_ALLERGY TABLE:

The primary key, MRA_ID, is the primary key that uniquely identifies every record in the table, ensuring that every allergy is stored in varchar (100) datatype. This ensures that each non-prime attribute, such as allergy, relies on the primary key for functionality. Transitively, non-key attributes should not rely on the main key, as the principal key directly affects allergies, and there is no indirect connection between MRA_ID and allergies.

T-SQL CODE

```
100 CREATE TABLE MedicalRecords_Allergy (
101     MRA_ID  varchar(50) NOT NULL,
102     Allergy varchar(100) NOT NULL,
103     FOREIGN KEY (MRA_ID) REFERENCES MedicalRecords (MedicalRecordID),
104     PRIMARY KEY (MRA_ID, Allergy)
105 )
```

```
Messages
Commands completed successfully.
Completion time: 2024-04-21T23:05:38.7265394+01:00
```

INSERTING VALUES TO MEDICALRECORDS_ALLERGY TABLE:

The HospitalManagementSystem database uses a primary key as a foreign key to uniquely identify each entry in medical records. The MRA_ID column stores the allergy name or type. Data integrity and logical storage are ensured when records are inserted into the MedicalRecords_Allergy table using the INSERT INTO statement.

T-SQL IMPLEMENTATION

```
257 -- Inserting  data into MedicalRecords_Allergy table
258 INSERT INTO MedicalRecords_Allergy (MRA_ID, Allergy) VALUES
259 ('MR1', 'yes'),
260 ('MR2', 'yes'),
261 ('MR3', 'yes'),
262 ('MR4', 'no'),
263 ('MR5', 'no'),
264 ('MR6', 'yes'),
265 ('MR7', 'no'),
266 ('MR8', 'yes'),
267 ('MR9', 'no'),
268 ('MR10', 'no'),
269 ('MR11', 'yes'),
270 ('MR12', 'yes'),
271 ('MR13', 'no'),
272 ('MR14', 'yes');
273
274
275
276
277
```

```
Messages
(14 rows affected)
Completion time: 2024-04-20T00:35:22.1744950+01:00
```

PART 2

QUESTION 2

To make sure that the AppointmentDate in the Appointments table is not in the past, the HospitalManagementSystem database is putting in place a constraint. Using the ALTER TABLE statement in T-SQL, this constraint is applied to the table, and its AppointmentDate field is checked to make sure it is not earlier than the current system date. Due to the constraint's ability to prohibit users from inadvertently setting appointments for past dates, data integrity is preserved, and the user experience is enhanced. Furthermore, it lessens pointless processing of appointments from the past, improving system performance. By putting this constraint in place, you can make sure that the scheduling in the Appointments table is accurate, current, and takes future appointments into account.

T-SQL CODE

```
256 --Question 2
257 --To add a constrain to check that the appoinment date is not in the past
258 ALTER TABLE dbo.Appointments
259 ADD CONSTRAINT Check_AppointmentDateNotInPast
260 CHECK (AppointmentDate >= CAST(GETDATE() AS DATE));
261
262
```

```
Messages
Commands completed successfully.
Completion time: 2024-04-22T00:43:53.9865414+01:00
```

Therefore, using a later date that is in the past.

```
253 --To add a constrain to check that the appoinment date is not in the past
254 ALTER TABLE dbo.Appointments
255 ADD CONSTRAINT Check_AppointmentDateNotInPast
256 CHECK (AppointmentDate >= CAST(GETDATE() AS DATE));
257
258
259 SELECT* from Doctors
```

```
Messages
Msg 2714, Level 16, State 5, Line 254
There is already an object named 'Check_AppointmentDateNotInPast' in the database.
Msg 1750, Level 16, State 1, Line 254
Could not create constraint or index. See previous errors.

Completion time: 2024-04-23T21:32:44.4703354+01:00
```

QUESTION 3

A list of people with cancer diagnoses in their medical records who are older than 40 years old is obtained using the SQL query. The SELECT DISTINCT clause in the query is used to extract unique entries from the result set. PatientID, FirstName, LastName, DateOfBirth, and Diagnosis are the columns that have been chosen. The main table for data retrieval is the Patients table. Based on the PatientID, an inner join is conducted between the MedicalRecords table and the Patients table. This link links every medical record to its corresponding diagnosis. By computing a date forty years prior to the present date, the WHERE clause screens out patients who are older than forty. Diagnoses that contain the word "CANCER" alone, regardless of case, are filtered by the UPPER clause.

T-SQL CODE

```
273  --Question 3
274  --List all the patients with older than 40 and have Cancer in diagnosis
275  SELECT DISTINCT
276      p.PatientID,
277      p.FirstName,
278      p.LastName,
279      p.DateOfBirth,
280      m.Diagnosis
281  FROM
282      Patients p
283  JOIN
284      MedicalRecords mr ON p.PatientID = mr.PatientID
285  JOIN
286      MedicalRecords_Diagnosis m ON mr.MedicalRecordID = m.MRD_ID
287  WHERE
288      p.DateOfBirth < DATEADD(YEAR, -40, GETDATE())
289      AND UPPER(m.Diagnosis) LIKE '%CANCER%';
290
```

Results | Messages

	PatientID	FirstName	LastName	DateOfBirth	Diagnosis
1	16	Helen	Chidi	1978-06-20	Cancer
2	19	Ada	Lark	1970-01-01	Cancer
3	24	Andy	Wope	1977-09-25	Cancer

QUESTION 4A.

A tool to access patient data and specifics of medications provided to patients based on a specified medication name is the `SearchMedicineByName` stored procedure. The process's goal is to increase efficiency by lowering network traffic. The prescribed medicine names and their prescribed dates are displayed in the query by means of a SQL statement that selects columns from the `Patients`, `MedicalRecords`, and `Prescription` tables.

Next, a WHERE clause is used to filter the results, ensuring that only records with the medicine name matching or containing the supplied @MedicineName are included.

Next, the results are sorted in descending order by MedicinePrescribedDate, with the most recent prescriptions appearing first. The saved process provides a convenient means to look up medications by name across patient prescriptions, offering crucial details that help medical personnel comprehend the drug history of their patients.

T-SQL CODE

```
--Question 4A Search Procedure to check a particular medicine Name
CREATE PROCEDURE SearchMedicineByName
    @MedicineName VARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        p.FirstName,
        p.LastName,
        pmd.MedicineName,
        pmd.MedicinePrescribedDate
    FROM
        Patients p
    JOIN
        MedicalRecords mr ON p.PatientID = mr.PatientID
    JOIN
        Prescription pmd ON mr.MedicalRecordID = pmd.MedicalRecordID
    WHERE
        UPPER(pmd.MedicineName) LIKE '%' + UPPER(@MedicineName) + '%'
    ORDER BY
        pmd.MedicinePrescribedDate DESC;

END;
```

```
Messages
Commands completed successfully.
Completion time: 2024-04-22T01:06:52.8425741+01:00
```

TO CALL THE PROCEDURE AND SHOW RESULT

To retrieve medications based on a given medication name, use the database-based `SearchMedicineByName` stored procedure. The name of the medication being looked for is represented by the parameter @MedicineName = 'Statins', which is used to carry out the procedure. Using the wildcard {%} before and after the {@MedicineName} argument value, the `LIKE` operator makes sure that any medicine name that contains the term 'Statins' is found. The outcome is a collection of rows from the Prescription table where the term 'Statins' is either present in the MedicineName or matches it. Using the wildcard in this stored method, it is effective to search for medications by name.

T-SQL CODE

```
318 EXEC SearchMedicineByName @MedicineName = 'Statins';
319
```

72 %

Results | Messages

	FirstName	LastName	MedicineName	MedicinePrescribedDate
1	Ada	Lark	Statins	2024-04-20
2	Ifeanyi	Clark	Statins	2024-04-14

QUESTION 4B

With an appointment date set to today, the T-SQL method `GetPatientsDiagnosisAndAllergiesToday()` obtains the diagnosis and allergies for any patient whose medical record contains that information. Three statements make up the function structure: SELECT, JOIN, and LEFT JOIN. Using PatientID, the JOIN statement links the `MedicalRecords` and `Patients` tables. Based on MedicalRecordID, the JOIN statement joins the MedicalRecords_Diagnosis table with the MedicalRecords_Diagnosis table. Based on MedicalRecordID, the JOIN statement links the MedicalRecords_Diagnosis and MedicalRecords_Allergy tables. The results are limited to patients whose medical records have a AppointmentDate set to today's date, thanks to the `WHERE` clause. To reduce the amount of data returned and concentrate on the essential basic information needed for the current task. For patients who are booked for an appointment today, this structure offers a simple and straightforward method to get the diagnosis and allergies. Patients with scheduled appointments are retrieved using the dbo.GetPatientsDiagnosisAndAllergiesToday() function. This function identifies patients, fetches their diagnoses and allergies, and presents a unified view for each patient using STRING_AGG for clarity.

T-SQL CODE

```
321 -------4B Return a full list of diagnosis and allergies or a specific patient who has an appointmnet today(
322 ----since i have more than one patient for the date
323 CREATE FUNCTION GetPatientsDiagnosisAndAllergiesToday()
324 RETURNS TABLE
325 AS
326 RETURN
327 (
328     SELECT
329         p.PatientID,
330         p.FirstName,
331         p.LastName,
332         md.Diagnosis,
333         ma.Allergy
334     FROM
335         Patients p
336     LEFT JOIN
337         MedicalRecords mr ON p.PatientID = mr.PatientID
338     LEFT JOIN
339         MedicalRecords_Diagnosis md ON mr.MedicalRecordID = md.MRD_ID
340     LEFT JOIN
341         MedicalRecords_Allergy ma ON mr.MedicalRecordID = ma.MRA_ID
342     WHERE
343         mr.AppointmentDate = CAST(GETDATE() AS DATE)
344 );
```

%

Messages

Commands completed successfully.

TO CALL THE PROCEDURE AND SHOW RESULT

```
SELECT * FROM dbo.GetPatientsDiagnosisAndAllergiesToday()-------since i have more than one patient for same day
```

Results | Messages

	PatientID	FirstName	LastName	Diagnosis	Allergy
1	21	Sapia	Ikem	Tendinitis	no
2	22	Ehian	Umeh	Endometriosis	yes
3	23	Heen	Chida	Neuropathy	yes
4	24	Andy	Wope	Cancer	no
5	25	Bemin	Ukoye	Endometriosis	yes

QUESTION 4C

The `UpdateDoctorDetails` stored procedure provides an organized and regulated way to update a doctor's information in the `Doctors` table. The process allows changing a doctor's profile with several input parameters. Based on the supplied DoctorID, the procedure is intended to update the doctor's information in the Doctors table. The `EXEC UpdateDoctorDetails` statement, which demonstrates how to run a stored procedure with sample values to update a doctor's details, is used to conduct the procedure. After the operation is completed, the post-execution statement is used to retrieve and display all the records from the Doctors table, displaying the doctor's updated details.

T- SQL CODE

QUESTION 4D.

```
------4C Stored Procedure to Update an existing Doctors details
CREATE PROCEDURE UpdateDoctorDetails
    @DoctorID VARCHAR(50), @FirstName NVARCHAR(50), @MiddleName NVARCHAR(50) = NULL, @LastName NVARCHAR(50),
    @Specialty NVARCHAR(50), @Email NVARCHAR(100), @PhoneNumber NVARCHAR(15), @DepartmentID VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE Doctors
    SET
        FirstName = @FirstName, MiddleName = @MiddleName, LastName = @LastName, Specialty = @Specialty, Email = @Email,
        PhoneNumber = @PhoneNumber, DepartmentID = @DepartmentID
    WHERE
        DoctorID = @DoctorID;
    IF @@ROWCOUNT = 0
    BEGIN
        RAISERROR('Doctor with ID %s not found.', 16, 1, @DoctorID);
        RETURN;
    END
    SELECT 'Doctor details updated successfully.' AS Message;
END;
```

Messages

```
Commands completed successfully.
Completion time: 2024-04-22T01:28:45.6786723+01:00
```

```
EXEC UpdateDoctorDetails
    @DoctorID = 'D8', @FirstName = 'Nkechimma', @MiddleName = 'Edaward', @LastName = 'Kalum',
    @Specialty = 'Cardiologist', @Email = 'nkelu_updated@gmail.com', @PhoneNumber = '123-456-7999',
    @DepartmentID = 'DR1';
```

Results | Messages

	Message
1	Doctor details updated successfully.

```
SELECT*FROM Doctors
```

Results | Messages

	DoctorID	FirstName	MiddleName	LastName	Specialty	Email	PhoneNumber	DepartmentID
1	D10	Adim	James	Johnson	Orthopedic Surgeon	adimjohnson@yahoo.com	345-678-9045	DR3
2	D11	Emily	NULL	Okem	Pediatrician	emilyoke@yandex.com	456-789-0176	DR4
3	D12	David	Charles	Wilson	Oncologist	davidwilson@ru.com	567-890-1234	DR5
4	D13	Oge	Maria	Martinez	Gynecologist	ogemartinez@gmail.com	678-901-2354	DR6
5	D14	Martha	NULL	Taylor	Gastroenterologist	mmataylor@outlook.com	789-012-3478	DR7
6	D8	Nkechimma	Edaward	Kalum	Cardiologist	nkelu_updated@gmail.com	123-456-7999	DR1
7	D9	Alice	NULL	Mba	Neurologist	alicemba@gmail.com	234-567-8934	DR2

A structured way to remove finished appointments from a clone table (Clone_Appointment_tb) while keeping the original data in the 'Appointments' table is to use the DeleteCompletedAppointments stored procedure. This method guarantees that information is kept for future use in a different table. To ensure that the completed appointments have been successfully deleted, the process entails making a clone table, running the stored procedure, and double-checking the clone table. The original data is kept in the 'Appointments' table and is managed in a safe and organized manner by the DeleteCompletedAppointments stored procedure. By using this technique, you may protect your data even further and make sure that crucial documents are not erased.

T-SQL CODE

```
388 CREATE PROCEDURE DeleteCompletedAppointments
389 AS
390 BEGIN
391     DELETE FROM Clone_Appointment_tb
392     WHERE Status = 'Completed';
393 END;
394
395
```

72 %

Messages

```
Commands completed successfully.

Completion time: 2024-04-22T01:38:44.8383669+01:00
```

CLONE APPOINTMENT TABLE CREATED

```
380 --4D Stored Procedure to Delete the appointment who status is already completed.
381 -- To create a clone Appointment Table in order to retian my data in the Appoinment Table
382 SELECT *
383 INTO Clone_Appointment_tb
384 FROM Appointments;
385 Select * from Clone_Appointment_tb
```

72 %

Results | Messages

	AppointmentID	PatientID	DoctorID	DepartmentID	AppointmentDate	AppointmentTime	Review	Status
1	AP1	12	D8	DR1	2024-04-27	08:00:00.0000000	Good service	Completed
2	AP10	21	D12	DR5	2024-04-22	12:00:00.0000000	Excellent	Completed
3	AP11	22	D13	DR6	2024-04-22	13:00:00.0000000	Very good	Cancelled
4	AP12	23	D14	DR7	2024-04-22	14:00:00.0000000	Satisfied	Completed
5	AP13	24	D8	DR1	2024-04-22	16:00:00.0000000	Very good	Completed
6	AP14	25	D12	DR5	2024-04-22	20:00:00.0000000	Very good	Completed
7	AP2	13	D10	DR3	2024-04-24	19:00:00.0000000	Excellent	Completed
8	AP3	14	D14	DR7	2024-04-25	18:00:00.0000000	Satisfied	Completed
9	AP4	15	D9	DR2	2024-05-25	16:00:00.0000000	Very good	Pending
10	AP5	16	D12	DR5	2024-04-23	12:00:00.0000000	Excellent	Pending
11	AP6	17	D14	DR7	2024-04-23	13:00:00.0000000	Satisfied	Completed
12	AP7	18	D9	DR2	2024-05-01	09:00:00.0000000	Satisfied	Cancelled
13	AP8	19	D10	DR3	2024-05-16	10:00:00.0000000	Very good	Cancelled
14	AP9	20	D11	DR4	2024-04-23	11:00:00.0000000	Not good	Completed

```
394 --To crosscheck the clone table to confirm the completed Appoinment has been deleted
395 select * from Clone_Appointment_tb;
396
```

65 %

Results | Messages

	AppointmentID	PatientID	DoctorID	DepartmentID	AppointmentDate	AppointmentTime	Review	Status
1	AP11	22	D13	DR6	2024-04-22	13:00:00.0000000	Very good	Cancelled
2	AP4	15	D9	DR2	2024-05-25	16:00:00.0000000	Very good	Pending
3	AP5	16	D12	DR5	2024-04-23	12:00:00.0000000	Excellent	Pending
4	AP7	18	D9	DR2	2024-05-01	09:00:00.0000000	Satisfied	Cancelled
5	AP8	19	D10	DR3	2024-05-16	10:00:00.0000000	Very good	Cancelled

QUESTION 5

A complete tool that makes it easy to obtain and analyse appointment details, doctor information, and department details in one view is the AllAppointmentsDetails view. Using their Doctors and Departments IDs, users can join departments, tables, and doctors. The view offers a comprehensive overview of all appointments, highlighting information about the appointment, the doctor, the department, and any related reviews or comments. Users can obtain insights into the hospital's appointment system by combining tables based on the IDs. By retrieving and displaying every entry from the `AllAppointmentsDetails` view, the JOIN statement gives users access to information about the hospital's appointment system.

T-SQL CODE

```
--Question 5 Create a View the appointment date and time,
---showing all previous and current appointments for all doctors,
---and including details of the department (the doctor is associated with),
---doctor's specialty and any associate review/feedback given for a doctor
CREATE VIEW AllAppointmentsDetails AS
SELECT
    A.AppointmentID,
    A.AppointmentDate,
    A.AppointmentTime,
    A.Review,
    A.Status,
    D.DoctorID,
    D.FirstName AS DoctorFirstName,
    D.LastName AS DoctorLastName,
    D.Specialty,
    Dep.DepartmentName
FROM
    Appointments A
JOIN
    Doctors D ON A.DoctorID = D.DoctorID
JOIN
    Departments Dep ON D.DepartmentID = Dep.DepartmentID;
```

72 %

Messages

```
Commands completed successfully.
Completion time: 2024-04-22T02:03:39.2329569+01:00
```

ALLAPPOINTMENTSDETAILS TABLE CREATED

```
SELECT * FROM AllAppointmentsDetails;
```

72 %

Results | Messages

	AppointmentID	AppointmentDate	AppointmentTime	Review	Status	DoctorID	DoctorFirstName	DoctorLastName	Specialty	DepartmentName
1	AP1	2024-04-27	08:00:00.0000000	Good service	Completed	D8	Nkechimma	Kalum	Cardiologist	Cardiology
2	AP10	2024-04-22	12:00:00.0000000	Excellent	Completed	D12	David	Wilson	Oncologist	Oncology
3	AP11	2024-04-22	13:00:00.0000000	Very good	Cancelled	D13	Oge	Martinez	Gynecologist	Gynecology
4	AP12	2024-04-22	14:00:00.0000000	Satisfied	Completed	D14	Martha	Taylor	Gastroenterologist	Gastroenterology
5	AP13	2024-04-22	16:00:00.0000000	Very good	Completed	D8	Nkechimma	Kalum	Cardiologist	Cardiology
6	AP14	2024-04-22	20:00:00.0000000	Very good	Completed	D12	David	Wilson	Oncologist	Oncology
7	AP2	2024-04-24	19:00:00.0000000	Excellent	Completed	D10	Adim	Johnson	Orthopedic Surgeon	Orthopedics
8	AP3	2024-04-25	18:00:00.0000000	Satisfied	Completed	D14	Martha	Taylor	Gastroenterologist	Gastroenterology
9	AP4	2024-05-25	16:00:00.0000000	Very good	Pending	D9	Alice	Mba	Neurologist	Neurology
10	AP5	2024-04-23	12:00:00.0000000	Excellent	Pending	D12	David	Wilson	Oncologist	Oncology
11	AP6	2024-04-23	13:00:00.0000000	Satisfied	Completed	D14	Martha	Taylor	Gastroenterologist	Gastroenterology
12	AP7	2024-05-01	09:00:00.0000000	Satisfied	Cancelled	D9	Alice	Mba	Neurologist	Neurology
13	AP8	2024-05-16	10:00:00.0000000	Very good	Cancelled	D10	Adim	Johnson	Orthopedic Surgeon	Orthopedics
14	AP9	2024-04-23	11:00:00.0000000	Not good	Completed	D11	Emily	Okem	Pediatrician	Pediatrics

QUESTION 6

The 'UpdateAppointmentStatus' trigger ensures that an appointment is cancelled and automatically updates its status. This trigger ensures that the slot is rebooked, allowing for rescheduling. The 'UPDATE' operation on the 'Appointments' database uses the trigger, minimizing manual intervention and preventing data errors. This automation ensures that cancelled spaces are quickly reopened for reservations.

T-SQL CODE

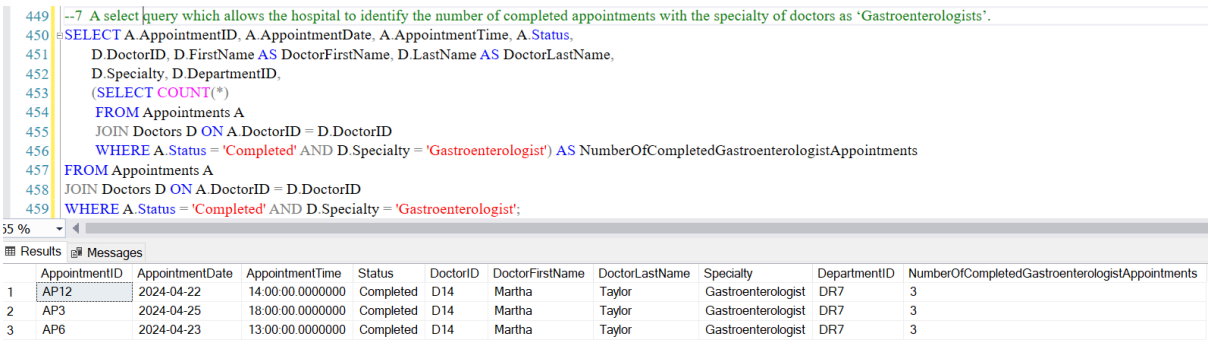
```
--Question 6 A trigger to Change the Cancelled Appointment To Available
  CREATE TRIGGER UpdateAppointmentStatus
ON Appointments
AFTER UPDATE
AS
BEGIN
  -- Check if the Status column has been updated to 'Cancelled'
  IF UPDATE(Status)
  BEGIN
    UPDATE Appointments
    SET Status = 'Available'
    FROM inserted i
    WHERE Appointments.AppointmentID = i.AppointmentID
    AND i.Status = 'Cancelled';
  END
END;
```

```
Messages
Commands completed successfully.
Completion time: 2024-04-22T02:05:37.7240298+01:00
```

QUESTION 7

Using a subquery, the SQL query obtains comprehensive data regarding the number of completed appointments for gastroenterologists, including the overall number of completed appointments for each gastroenterologist. A.AppointmentID, A.AppointmentDate, A.AppointmentTime, A.Status, D.DoctorID, D.FirstName, D.LastName, D.Specialty, D.DepartmentID, and the total number of completed appointments are among the columns included in the query. Embedded in the main query, the subquery determines the total number of completed appointments for physicians with the specialty of "Gastroenterology." To monitoring and evaluating appointment data particularly for the gastroenterology department, this combined data is helpful. Utilizing the 'DoctorID' column, the primary query links the 'Appointments' and 'Doctors' tables to obtain appointments and the associated physician information.

T-SQL CODE



```
--7  A select query which allows the hospital to identify the number of completed appointments with the specialty of doctors as 'Gastroenterologists'.
SELECT A.AppointmentID, A.AppointmentDate, A.AppointmentTime, A.Status,
    D.DoctorID, D.FirstName AS DoctorFirstName, D.LastName AS DoctorLastName,
    D.Specialty, D.DepartmentID,
    (SELECT COUNT(*)
     FROM Appointments A
     JOIN Doctors D ON A.DoctorID = D.DoctorID
     WHERE A.Status = 'Completed' AND D.Specialty = 'Gastroenterologist') AS NumberOfCompletedGastroenterologistAppointments
FROM Appointments A
JOIN Doctors D ON A.DoctorID = D.DoctorID
WHERE A.Status = 'Completed' AND D.Specialty = 'Gastroenterologist';
```

Results Messages

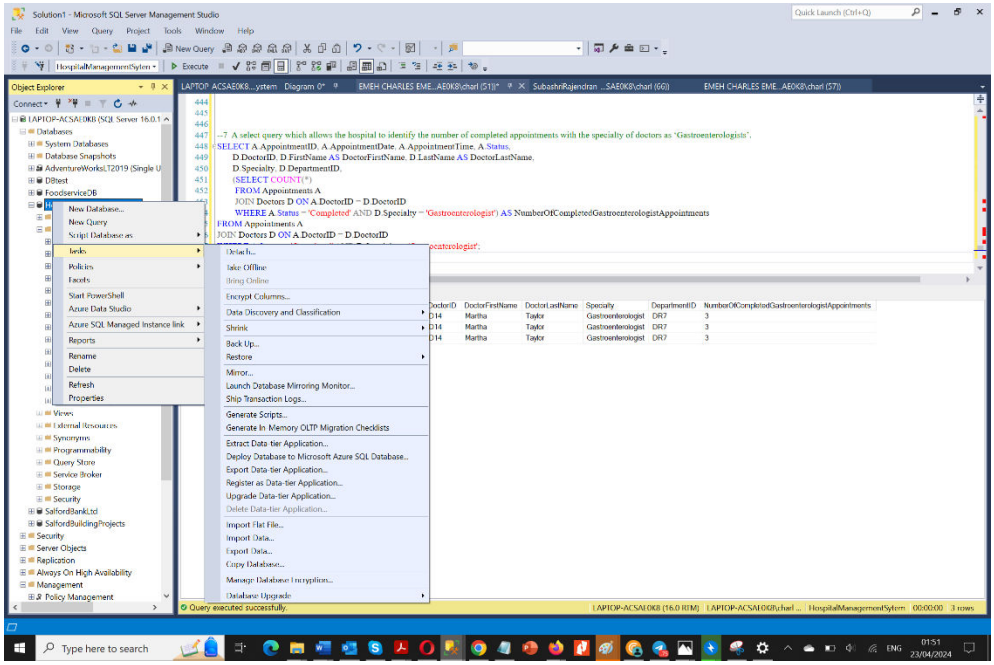
	AppointmentID	AppointmentDate	AppointmentTime	Status	DoctorID	DoctorFirstName	DoctorLastName	Specialty	DepartmentID	NumberOfCompletedGastroenterologistAppointments
1	AP12	2024-04-22	14:00:00.0000000	Completed	D14	Martha	Taylor	Gastroenterologist	DR7	3
2	AP3	2024-04-25	18:00:00.0000000	Completed	D14	Martha	Taylor	Gastroenterologist	DR7	3
3	AP6	2024-04-23	13:00:00.0000000	Completed	D14	Martha	Taylor	Gastroenterologist	DR7	3

BACKUP PROCEDURE

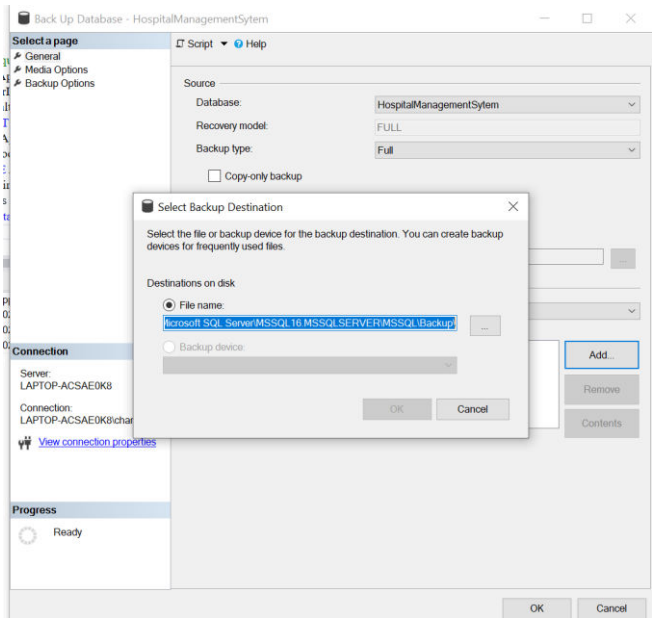
In Hospital management systems, database backup and recovery are crucial to preventing data loss in the event of a system failure or corrupted data. It is important to create a recovery strategy and make sure that daily and weekly full and incremental backups are performed. Automated tools should be used for regular backups, and the procedure should be tested to guarantee dependability and speedy restoration.

The following steps are implemented to create the backup for the data file:

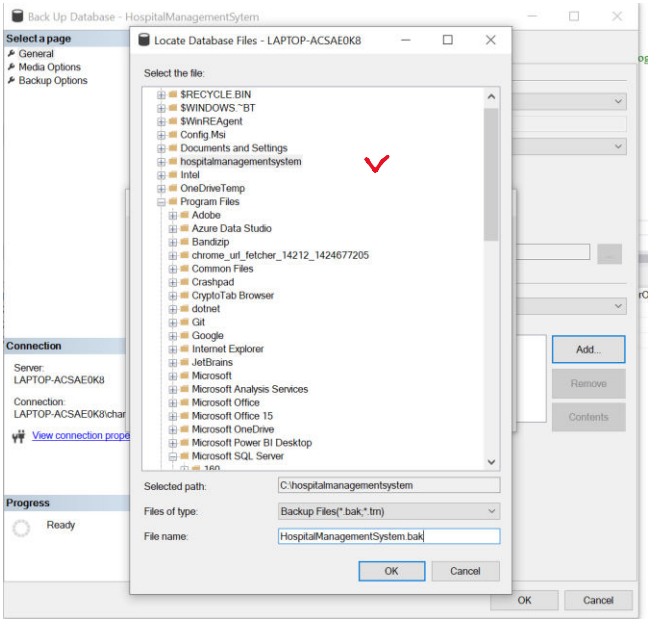
Step 1: Right click on the created database in the Object Explorer.
Hospital Management System → Tasks →Back up



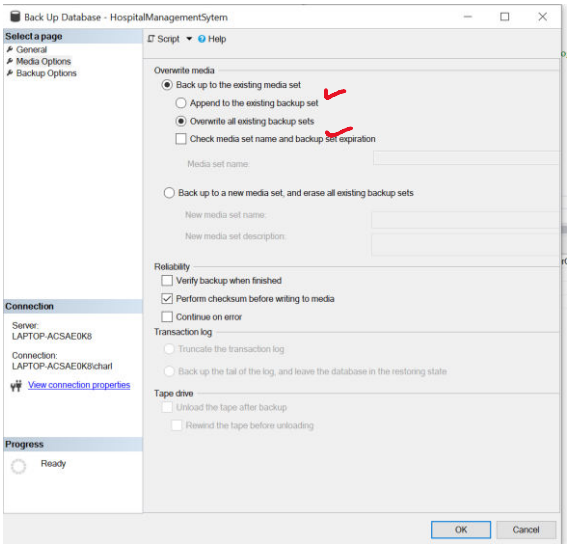
Step 2: Click on Remove and the click Add to select the Back-up file which has been created before.



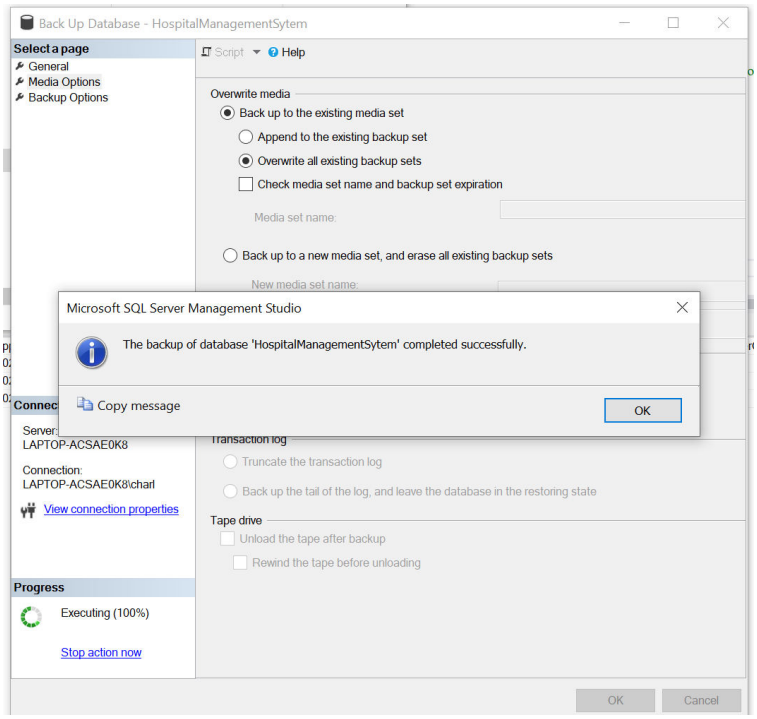
Step 3:
Once the File is selected, give the name as ‘HospitalManagementSystem.bak’ to save the file in the given name.



Step 4: After that when the relevant options are selected, backup the existing media and overwrite all existing backup set expiration and the dialog box will pop up saying, the backup was successful.



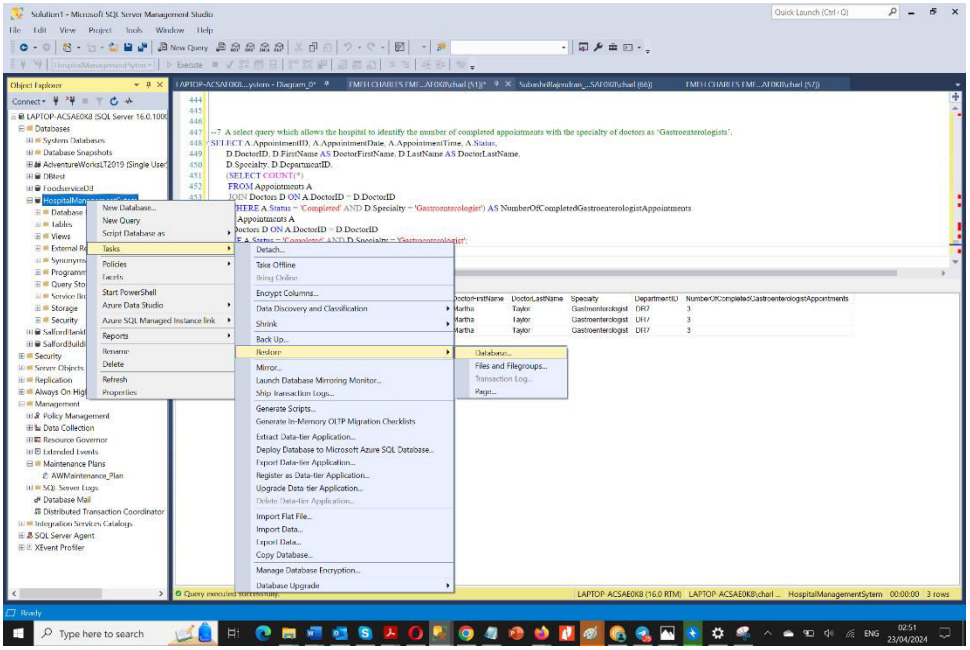
Step 5: Database successfully backed up.



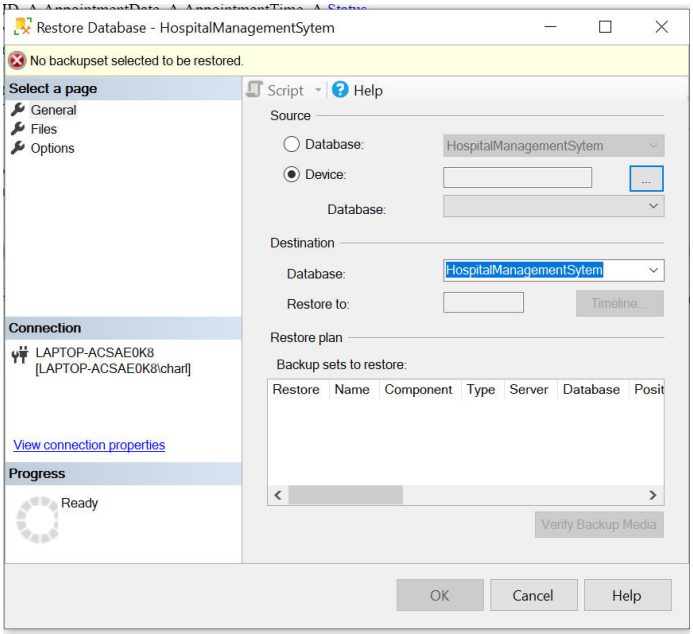
DATABASE RECOVERY

To ensure data consistency and integrity, database administration is essential. Identification of failure categories, confirmation of backups, database restoration, application of transaction logs, updating of applications, stakeholder communication, documentation of the recovery process, database testing and monitoring, and implementation of preventive measures are all part of the recovery process. The procedure guards against downtime and data loss by guaranteeing data availability and integrity.

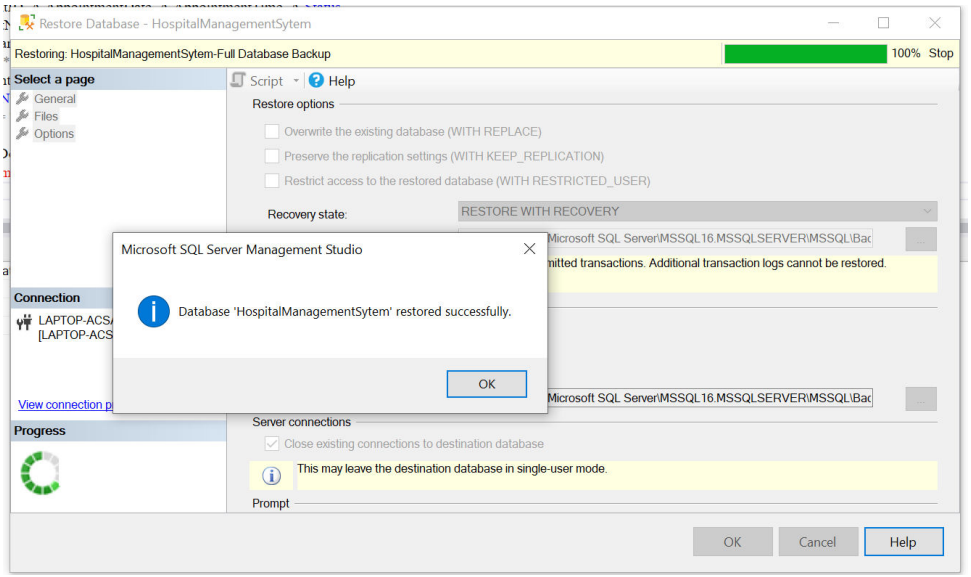
STEP 1:



STEP 2:


Restore Database - HospitalManagementSytem
No backupset selected to be restored.
Select a page
General
Files
Options
Script
Help
Source
Database:
HospitalManagementSytem
Device:
Database:
Destination
Database:
HospitalManagementSytem
Restore to:
Timeline...
Restore plan
Backup sets to restore:
Restore
Name
Component
Type
Server
Database
Posit
Verify Backup Media
Connection
LAPTOP-ACSAE0K8
[LAPTOP-ACSAE0K8\charl]
View connection properties
Progress
Ready
OK
Cancel
Help

STEP 3:


Restore Database - HospitalManagementSytem
Restoring: HospitalManagementSytem-Full Database Backup
100%
Stop
Select a page
General
Files
Options
Script
Help
Restore options
Overwrite the existing database (WITH REPLACE)
Preserve the replication settings (WITH KEEP_REPLICATION)
Restrict access to the restored database (WITH RESTRICTED_USER)
Recovery state:
RESTORE WITH RECOVERY
Microsoft SQL Server Management Studio
Database 'HospitalManagementSytem' restored successfully.
OK
Connection
Progress
Server connections
Close existing connections to destination database
This may leave the destination database in single-user mode.
Prompt
OK
Cancel
Help

DATABASE SECURITY

Database security is crucial for protecting sensitive data and ensuring only authorized users have access. To enhance database security, assign roles and permissions, create login users, and grant specific permissions on stored procedures and views. The login user authenticates users and restricts access, while granting select permissions on views and execute permissions on stored procedures. Regularly assess and modify user roles and rights to maintain database security and adapt to changing business requirements. Secure login users with strong passwords, appropriate allocation of roles and permissions, and specific permissions on stored procedures included.

T-SQL CODE

```
--Database Security
--To create a database Login Users name Database_Admin,Doctors_Mgt,AppointmentMgt_Officer
Create Login Database_Admin with Password = 'EMEH8604$';
Create Login Doctors_Mgt with Password = 'STAN1992S';
Create Login AppointmentMgt_Officer with password ='SELECTION1998$';


-- Create users in the database
Create User Database_Admin for Login Database_Admin;
Create User Doctors_Mgt for Login Doctors_Mgt;
Create User AppointmentMgt_Officer for Login AppointmentMgt_Officer;

--Grant All privileges to Database_Admin
GRANT CONTROL ON SCHEMA::dbo TO Database_Admin;

--Grant privileges to AppointmentMgt_Officer
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Appointments TO AppointmentMgt_Officer ;
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Clone_Appointment_tb TO AppointmentMgt_Officer;
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Patients TO AppointmentMgt_Officer;

--Grant Privileges to Doctors_Mgt
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Medicalrecords TO Doctors_Mgt;
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Doctors TO Doctors_Mgt;
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER ON dbo.Departments TO Doctors_Mgt;



-- Grant Stored Procedure Permissions using no Schema access
GRANT EXECUTE ON DeleteCompletedAppointments TO Database_Admin;
GRANT EXECUTE ON UpdateDoctorDetails TO Doctors_Mgt;
GRANT EXECUTE ON SearchMedicineByName TO AppointmentMgt_Officer;
```

```
%
Messages
Commands completed successfully.

Completion time: 2024-04-23T23:34:02.7426971+01:00
```

CONCLUSION

We have developed an extensive database system that meets the requirements of the Hospital management for maintaining member information, and update services. A well-designed schema with the appropriate tables, columns, and constraints, along with a group of stored procedures, user-defined functions, views, and database operation triggers, are the main elements of our solution. The structured process we used to design and implement the database system included: comprehending the requirements, designing the schema, creating database objects, writing queries, and evaluating the system.

TASK 2: FOOD SERVICE COMPANY DATABASE SYSTEM

INTRODUCTION:

A database system was developed for a food service company, storing data on restaurants, consumers, ratings, and cuisines. The system includes tables for Restaurants, Consumers, Ratings, and Restaurant_Cuisine, each containing detailed information about each restaurant. The system helps the company evaluate and improve services, categorizes, and filters restaurants based on offerings, and analyses real-world datasets for better decision-making for the company.

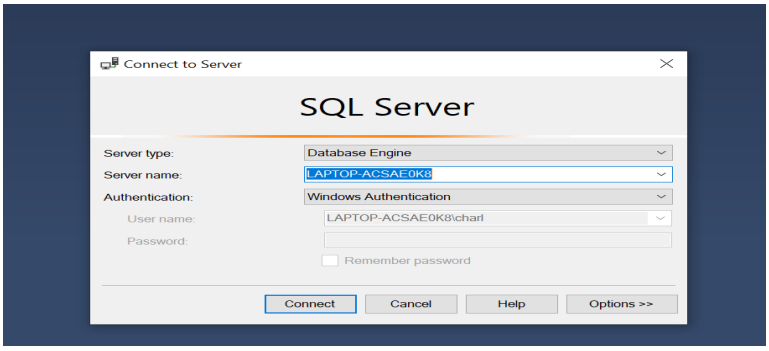
PART 1: DATABASE CREATION AND IMPORTING THE DATA:

Creating a database, importing the Four tables from the supplied CSV files, and setting up the necessary primary and foreign key constraints are all steps in the first phase of the task at hand. The three tables in the database will be called consumers, restaurants, ratings, restaurant_cuisines, the database will be called FoodserviceDB. To make it simpler to refer to the data, we must also make sure that the column names match those in the CSV files.

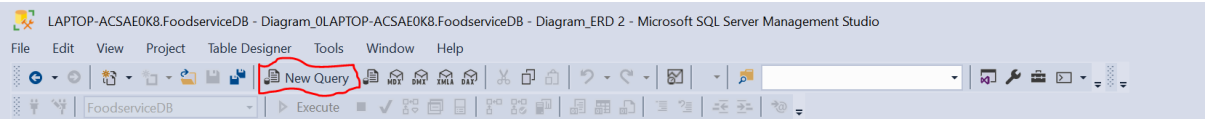
CREATING THE DATABASE:

To create the database and tables, we will be using SQL Server Management Studio. The following steps outline the process:

Step 1: Open SQL Server Management Studio and connect to the SQL Server instance.

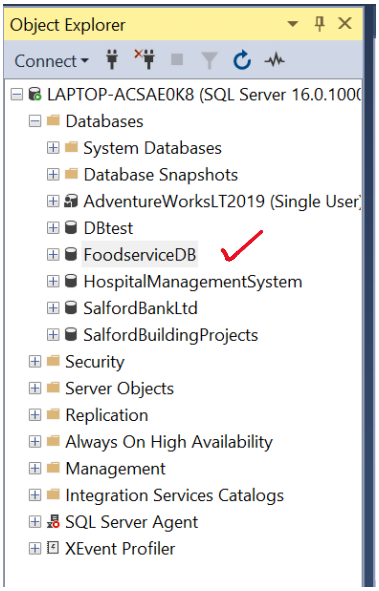


Step 2: Click on ‘New Query’ below the toolbox.

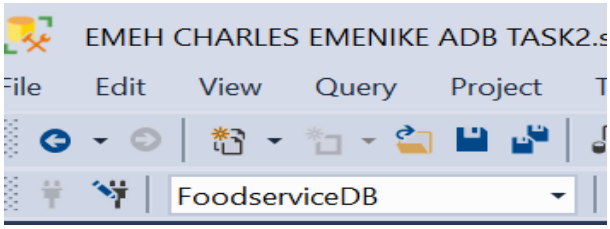


Step 3: Type the query and click on the execute button to execute it. Create the database with ‘FoodserviceDB. as the database name.

Step 4: Once the database is created, refresh the left panel to check if the created. database is appearing.



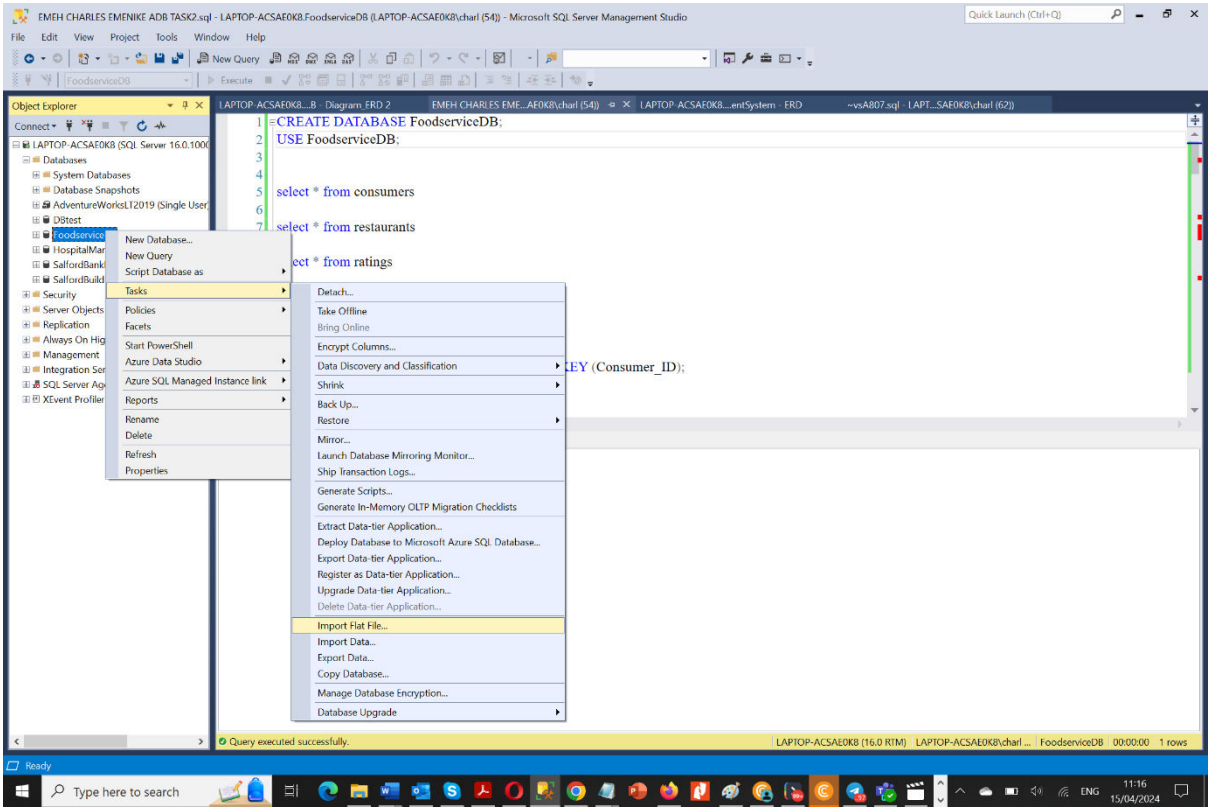
Step 5: Then change the database in which the queries are to be executed. Change it to 'FoodserviceDB



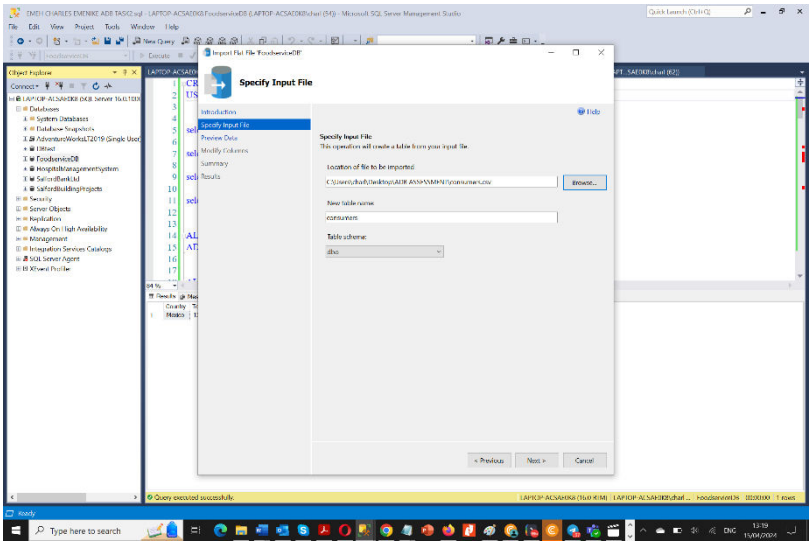
IMPORTING THE DATA:

The next step is to import the data from the local system after the tables has been created. To do this, we can use SQL Server Management Studio's Import Flat File wizard. The procedure is outlined in the following steps:

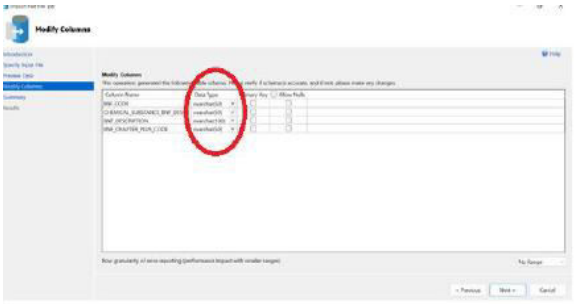
Step 1: Right Click in the created database ‘FoodserviceDB and select the Tasks option from the next appearing box.



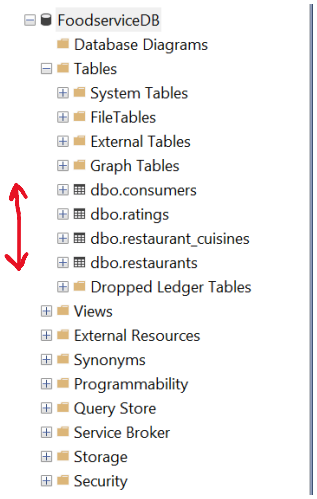
Step 2: Once the Import Flat File is opened, click on the Browse button to select the data from the local system. In the New table name, leave the name as ‘restaurant, consumers, ratings, restaurant_cuisine’ as directed in the assessment brief to let the code be rerunnable when submitted. After selecting the ‘consumers’ data from the local system, click on Next button at the bottom.



Step 3: After that the data in the chosen file will appear in the ‘Preview Data’, click next and then choose the Data type in this ‘Modify Columns’ and click Next.

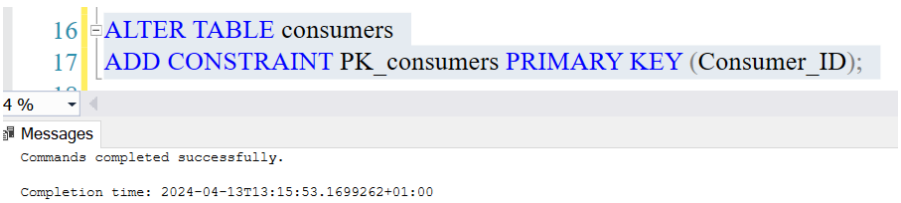


Step 4: Next the Results page will appear stating that the inserted data is successfully imported. Once all these steps are done, Click close. All these steps were followed to import the restaurant, consumers, ratings, restaurant_cuisine files respectively. The created tables and the imported tables will be in the object explorer panel below the created database FoodserviceDB.



ASSIGNING KEYS TO THE TABLE USING ‘ALTER’ STATEMENTS

Database design involves the `ALTER TABLE` statement, which allows users to apply new constraints to an existing table. The `ADD CONSTRAINT` statement adds a new constraint to the `consumers` table, defining the action to be performed. The primary key constraint, 'Consumer_ID', ensures referential integrity, data integrity, and uniqueness. This prevents duplicate entries and improves data consistency by forbidding the insertion of NULL values in the 'Consumer_ID' column. The primary constraint ensures data integrity and uniqueness.



ALSO, FOR RESTAURANT TABLE

```
18
19  ALTER TABLE restaurants
20  ADD CONSTRAINT PK_restaurants PRIMARY KEY (Restaurant_ID);
```

```
%
Messages
Commands completed successfully.

Completion time: 2024-04-13T13:21:51.1395193+01:00
```

A foreign key constraint, such as Fk_Consumer, ensures referential integrity between the `ratings` and `Consumers` tables by ensuring that each value in the `Consumer_ID column matches a valid `Consumer_ID in the `Consumers` table. This improves data consistency, integrity, and error prevention by preventing orphaned records and limiting data errors. However, it is crucial to ensure that the values in the `ratings` column match the `Consumers` table's values to prevent issues. Foreign key constraints may affect the speed of insert, update, and delete activities.

```
21  ALTER TABLE ratings
22  ADD CONSTRAINT Fk_Consumer
23  FOREIGN KEY (Consumer_ID) REFERENCES Consumers(Consumer_ID);
```

```
%
Messages
Commands completed successfully.

Completion time: 2024-04-13T14:18:51.6351355+01:00
```

```
26  ALTER TABLE ratings
27  ADD CONSTRAINT Fk_Restaurant
28  FOREIGN KEY (Restaurant_ID) REFERENCES restaurants(Restaurant_ID);
```

```
%
Messages
Commands completed successfully.

Completion time: 2024-04-13T14:23:43.1099929+01:00
```

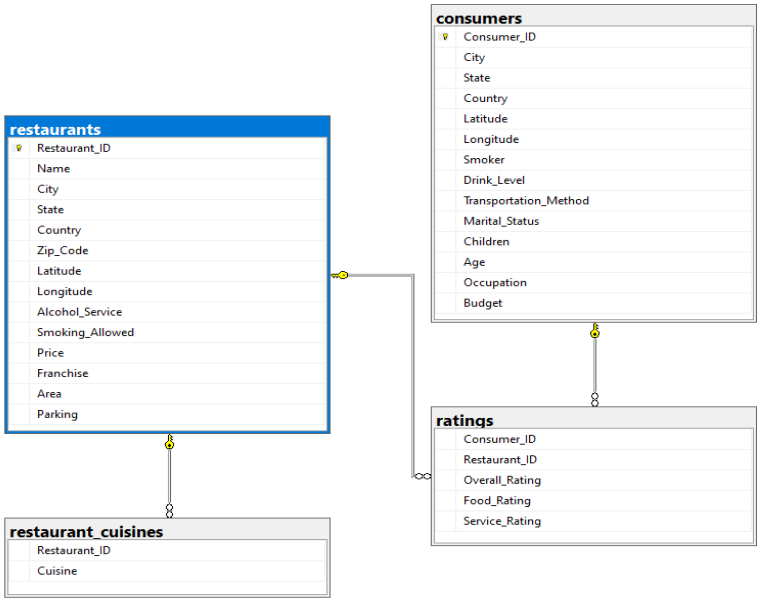
```
30  ALTER TABLE restaurant_cuisines
31  ADD CONSTRAINT Fk_restaurant_cuisines
32  FOREIGN KEY (Restaurant_ID) REFERENCES restaurants(Restaurant_ID);
```

```
%
Messages
Commands completed successfully.

Completion time: 2024-04-13T14:31:59.8534084+01:00
```

DATABASE DIAGRAM:

A database diagram displays a database's organizational structure and linkages. It helps to illustrate the tables, columns, relationships, and restrictions that are present in the database by providing a visual depiction of the database schema. A database diagram can be made using a variety of notations, including the Entity-Relationship (ER) notation, the Unified Modelling Language (UML), and Data Flow Diagrams (DFD). It is a useful tool for understanding the database structure and its relationships quickly and easily. This is the four data's schema.



RETRIEVING THE DATA: AFTER IMPORTING THE DATA, THE NEXT STEP IS TO RETRIEVE THE DATA

The integrity of foreign keys is crucial in data management, as they ensure that column matches in the `ratings` and `Consumers` tables match valid values. This prevents orphaned records and errors, while also ensuring that the values in the `ratings` column match the `Consumers` table's values.

FOR CONSUMERS TABLE CODE, SELECT*FROM CONSUMERS.

```
17  select * from consumers
```

Results | Messages

Consumer_ID	City	State	Country	Latitude	Longitude	Smoker	Drink_Level	Transportation_Method	Marital_Status	Children	Age	Occupation	Budget
U1001	San Luis Potosi	San Luis Potosi	Mexico	22.1399974822998	-100.978805541992	0	Abstemious	On Foot	Single	Independent	23	Student	Medium
U1002	San Luis Potosi	San Luis Potosi	Mexico	22.1500873565674	-100.983322143555	0	Abstemious	Public	Single	Independent	22	Student	Low
U1003	San Luis Potosi	San Luis Potosi	Mexico	22.1198463439941	-100.94652557373	0	Social Drinker	Public	Single	Independent	23	Student	Low
U1004	Cuernavaca	Morelos	Mexico	18.867000579834	-99.1829986572266	0	Abstemious	Public	Single	Independent	72	Employed	Medium
U1005	San Luis Potosi	San Luis Potosi	Mexico	22.1834774017334	-100.959892272949	0	Abstemious	Public	Single	Independent	20	Student	Medium
U1006	San Luis Potosi	San Luis Potosi	Mexico	22.1499996185303	-100.983001708984	1	Social Drinker	Car	Single	Independent	23	Student	Medium
U1007	San Luis Potosi	San Luis Potosi	Mexico	22.1184635162354	-100.938255310059	0	Casual Drinker	Public	Single	Independent	23	Student	Low
U1008	San Luis Potosi	San Luis Potosi	Mexico	22.122989654541	-100.923812866211	0	Social Drinker	Public	Single	Independent	23	Student	Low
U1009	San Luis Potosi	San Luis Potosi	Mexico	22.1594276428223	-100.990447998047	0	Abstemious	On Foot	Single	Kids	21	Student	Medium
U1010	San Luis Potosi	San Luis Potosi	Mexico	22.1908893585205	-100.998672485352	0	Social Drinker	Car	Married	Kids	25	Student	Medium
U1011	Ciudad Victoria	Tamaulipas	Mexico	23.7249717712402	-99.1528549194336	0	Abstemious	Public	Single	Independent	23	Student	Medium
U1012	Cuernavaca	Morelos	Mexico	18.8133487701416	-99.2436981201172	0	Casual Drinker	Public	Single	Independent	24	Student	Medium
U1013	San Luis Potosi	San Luis Potosi	Mexico	22.1746234893799	-100.993873596191	0	Abstemious	Public	Single	Independent	30	Employed	Medium
U1014	Ciudad Victoria	Tamaulipas	Mexico	23.7516078948975	-99.1701049804688	0	Abstemious	Public	Single	Independent	22	Student	Medium
U1015	San Luis Potosi	San Luis Potosi	Mexico	22.1267604827881	-100.905212402344	1	Social Drinker	Public	Single	Independent	23	Student	Medium
U1016	San Luis Potosi	San Luis Potosi	Mexico	22.1562461853027	-100.977401733398	0	Casual Drinker	On Foot	Single	Independent	21	Student	Medium
U1017	Cuernavaca	Morelos	Mexico	18.952615737915	-99.2016143798828	0	Casual Drinker	Public	Single	NULL	21	Employed	Medium
U1018	San Luis Potosi	San Luis Potosi	Mexico	22.1909484863281	-100.917900085449	1	Casual Drinker	Public	Single	Independent	23	Student	Low
U1019	San Luis Potosi	San Luis Potosi	Mexico	22.1533851623535	-100.975296020508	0	Casual Drinker	Public	Single	Independent	23	Student	Medium
U1020	Cuernavaca	Morelos	Mexico	18.8781890869141	-99.2229690551758	0	Abstemious	Public	Single	Independent	30	Employed	Medium

Query executed successfully. | LAPTOP-ACSAE0K8 (16.0 RTM) | LAPTOP-ACSAE0K8\charl ... | FoodserviceDB | 00:00:00 | 138 rows

FOR RESTAURANTS TABLE CODE, SELECT*FROM RESTAURANTS

```
17  select * from restaurants
```

Results | Messages

Restaurant_ID	Name	City	State	Country	Zip_Code	Latitude	Longitude	Alcohol_Service	Smoking_Allowed	Price	Franchise	Area	Parking
132560	Puesto de Gorditas	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7523040771484	-99.1669158935547	None	Yes	Low	0	Open	Public
132561	Cafe Ambar	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7268180847168	-99.1265029907227	None	No	Low	0	Closed	None
132564	Church's	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7309246063232	-99.1451873779297	None	No	Low	0	Closed	None
132572	Cafe Chaires	San Luis Potosi	San Luis Potosi	Mexico	NULL	22.1416473388672	-100.992713928223	None	No	Low	0	Closed	Yes
132583	McDonalds Centro	Cuernavaca	Morelos	Mexico	62000	18.922290802002	-99.2343292236328	None	No	Low	1	Closed	None
132584	Gorditas Doña Tota	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7523651123047	-99.1652908325195	None	No	Medium	1	Closed	Yes
132594	Tacos De Barbacoa Enfrente Del Tec	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7521667480469	-99.1657104492188	None	No	Low	0	Open	Public
132608	Hamburguesas La Perica	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7588043212891	-99.1651306152344	None	Yes	Low	1	Open	Public
132609	Pollo Frito Buenos Aires	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7602691650391	-99.1658630371094	None	No	Low	1	Closed	Yes
132613	Carnitas Mata	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529029846191	-99.1650772094727	None	Yes	Medium	1	Closed	Yes
132626	La Perica Hamburguesa	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7375831604004	-99.1351318359375	None	No	Medium	1	Closed	Yes
132630	Palomo Tec	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529296875	-99.1644744873047	None	No	Low	0	Closed	None
132654	Carnitas Mata Calle 16 de Septiembre	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.735523223877	-99.1295852661133	None	No	Low	0	Closed	None
132660	Carnitas Mata Calle Emilio Portes Gil	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7529430389404	-99.164680480957	None	No	Low	0	Closed	None
132663	Tacos Abi	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7525100708008	-99.1669540405273	None	No	Low	0	Closed	None
132665	Tacos Correcaminos	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7367973327637	-99.1342391967773	None	No	Low	0	Closed	None
132667	Little Pizza Emilio Portes Gil	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7526969909668	-99.1633605957031	None	No	Low	1	Closed	None
132668	Tacos El Guero	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7382125854492	-99.1519546508789	None	No	Low	0	Closed	None
132706	Gorditas Dona Tota	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.7292156219482	-99.1323547363281	None	No	Medium	1	Closed	Public
132715	Tacos De La Estacion	Ciudad Victoria	Tamaulipas	Mexico	NULL	23.732421875	-99.1586608886719	None	No	Low	0	Open	None

Query executed successfully. | LAPTOP-ACSAE0K8 (16.0 RTM) | LAPTOP-ACSAE0K8\charl ... | FoodserviceDB | 00:00:00 | 130 rows

FOR RATINGS TABLE CODE, SELECT*FROM RATINGS

```
17  select * from ratings
```

Results | Messages

Consumer_ID	Restaurant_ID	Overall_Rating	Food_Rating	Service_Rating
U1077	135085	2	2	2
U1077	135038	2	2	1
U1077	132825	2	2	2
U1077	135060	1	2	2
U1068	135104	1	1	2
U1068	132740	0	0	0
U1068	132663	1	1	1
U1068	132732	0	0	0
U1068	132630	1	1	1
U1067	132584	2	2	2
U1067	132733	1	1	1
U1067	132732	1	2	2
U1067	132630	1	0	1
U1067	135104	0	0	0
U1067	132560	1	0	0
U1103	132584	1	2	1
U1103	132732	0	0	2
U1103	132630	1	2	0
U1103	132613	2	2	2
U1103	132667	1	2	2

Query executed successfully.

FOR RESTAURANT_CUISINES TABLE CODE, SELECT*FROM RESTAURANT_CUISINES

```
17  select * from restaurant_cuisines
```

Results | Messages

	Restaurant_ID	Cuisine
	132560	Regional
	132572	Cafeteria
	132583	American
	132584	Mexican
	132594	Mexican
	132608	Mexican
	132609	Fast Food
	132613	Mexican
	132626	Italian
0	132630	Mexican
1	132663	Mexican
2	132665	Mexican
3	132667	Armenian
4	132668	Mexican
5	132706	Mexican
6	132715	Mexican
7	132717	Fast Food
8	132723	Mexican
9	132732	Mexican
0	132733	Pizzeria

Query executed successfully.

PART 2

QUESTION 1: This SQL query uses predefined conditions and filters to gather data from the restaurants` table. `WHERE Price = 'Medium' to filter restaurants based on their price, `AND Area = 'Open' to select restaurants with an open area, `AND Restaurant_ID IN (SELECT Restaurant_ID FROM restaurant_cuisines WHERE cuisine = 'Mexican') subquery to filter restaurants based on the type of cuisine they serve, and a `FROM restaurants` clause to specify the table to retrieve the data from are all included in the query. The `Restaurant_ID IN (...) condition guarantees that the result contains only restaurants whose `Restaurant_IDs match those returned by the subquery.

```
35  ----PART 2
36  ----QUESTION ONE
37  SELECT *
38  FROM restaurants
39  WHERE Price = 'Medium'
40  AND Area = 'Open'
41  AND Restaurant_ID IN (SELECT Restaurant_ID FROM restaurant_cuisines WHERE cuisine = 'Mexican');
42
```

Results | Messages

	Restaurant_ID	Name	City	State	Country	Zip_Code	Latitude	Longitude	Alcohol_Service	Smoking_Allowed	Price	Franchise	Area	Parking
1	135018	El Oceano Dorado	Cuernavaca	Morelos	Mexico	NULL	18.8598022460938	-99.2221603393555	Full Bar	Yes	Medium	0	Open	Yes
2	135106	El Rincón De San Francisco	San Luis Potosi	San Luis Potosi	Mexico	78000	22.1497097015381	-100.976089477539	Wine & Beer	Bar Only	Medium	0	Open	None

QUESTION 2: The SQL query SELECT cuisine, COUNT(*) as Total_restaurants, rc, ratings r ON rc.Restaurant_ID = r.Restaurant_ID, WHERE Overall_Rating = 1 AND (cuisine = 'Mexican' OR cuisine = 'Italian') , Following 'restaurant_cuisines' table and 'ratings' table, the query fetches data from 'restaurant_cuisines' and 'ratings' tables. It counts the number of restaurants (Total_restaurants) for each cuisine with an overall rating of 1 and the cuisine being either 'Mexican' or 'Italian'. The result displays two rows: one for 'Mexican' cuisine with the total number of restaurants with an overall rating of 1 and another for 'Italian' cuisine with the total number of restaurants with an overall rating of 1. The example output shows that there are 81 Mexican restaurants and 11 Italian restaurants with an overall rating of 1 in the database. The query is executed by fetching data from the 'restaurant_cuisines' and 'ratings' tables, filtering the ratings to include only 'Mexican' or 'Italian' cuisines, and grouping the result by the 'cuisine' column is carried out by obtaining information from the "restaurant_cuisines" and "ratings" tables, limiting the ratings to just "Mexican" or "Italian" cuisines, and classifying the outcome according to the "cuisine" column and comparing them based on the total restaurant respectively.

```
45  ------QUESTION 2.
46  SELECT cuisine, COUNT(*) as Total_restaurants
47  FROM restaurant_cuisines rc
48  JOIN ratings r
49  ON rc.Restaurant_ID = r.Restaurant_ID
50  WHERE Overall_Rating = 1
51  AND (cuisine = 'Mexican' OR cuisine = 'Italian')
52  GROUP BY cuisine;
```

Results | Messages

	cuisine	Total_restaurants
	Italian	11
	Mexican	87

QUESTION 3: The average age of customers who have given a service rating of zero is determined using the SQL query. The consumers table's data is retrieved using a sequence of steps in the query. The ratings table and consumers table are linked using the Consumer_ID column, and the joined data is filtered to contain only entries where the Service_Rating is 0. This requirement makes sure that when determining the average age, only customers who have given a service rating of 0 are taken into account.
For instance, the query will join the tables based on the Consumer_ID column, filter the combined data to include only customers with a Service_Rating of 0, and get the average age if the sample data comes from the ratings and consumers tables of these consumers, and round the average age to the nearest whole number.

```
54  ------- QUESTION 3.
55  SELECT ROUND(AVG(c.Age),0) AS Average_Age
56  FROM consumers c
57  JOIN ratings r
58  ON c.Consumer_ID = r.Consumer_ID
59  WHERE Service_Rating = 0;
60
```

Results | Messages

	Average_Age
	26

QUESTION 4: This SQL query compiles a list of restaurants, their youngest age, and customer ratings for their food based on data retrieved from several sources. The query joins the ratings table with itself using the Restaurant_ID and links the ratings of the same restaurant using a self-join. The customers who have rated the restaurants are then linked by the query, which uses the Consumer_ID to join the ratings table with the consumer's table. The name of the restaurant is then obtained by joining the restaurants table and the ratings table using the Restaurant_ID.
The query computes the minimum age and shows the youngest age among customers who have evaluated each restaurant. It groups the results by Restaurant_ID, Restaurant_Name, and Food_Rating. The sequence in which the results are arranged first by Youngest Age, which goes from youngest to oldest, and then by Food Rating, which goes from highest to lowest.
The query determines the youngest age group of customers who have left ratings for each restaurant and shows the Food_Rating that those customers have left. The restaurants are listed with the youngest age group at the top and the food rating at the bottom. The query gives the name of each restaurant along with its ID by incorporating the restaurant name from the 'Restaurants' table, which enhances the information in the return.

```
62  ------QUESTION 4.
63  SELECT r.Restaurant_ID, rest.Name AS Restaurant_Name, MIN(c.Age) AS Youngest_Age, ra.Food_Rating
64  FROM ratings r
65  JOIN ratings ra ON r.Restaurant_ID = ra.Restaurant_ID
66  JOIN consumers c ON ra.Consumer_ID = c.Consumer_ID
67  JOIN Restaurants rest ON r.Restaurant_ID = rest.Restaurant_ID
68  GROUP BY r.Restaurant_ID, rest.Name, ra.Food_Rating
69  ORDER BY Youngest_Age ASC, ra.Food_Rating DESC;
```

Results | Messages

	Restaurant_ID	Restaurant_Name	Youngest_Age	Food_Rating
	135013	Giovannis	18	2
	135019	Restaurant Bar Coty Y Pablo	18	2
	134999	Kiku Cuernavaca	18	1
	132773	El Cotorreo	18	1
	134999	Kiku Cuernavaca	19	2
	132767	Restaurant Familiar El Chino	19	2
	135021	Subway	19	2
	135039	Restaurant De Mariscos De Picon	20	2
	135025	El Rincon De San Francisco	20	2
0	135051	Restaurante Versalles	20	2
1	134975	Rincon Del Bife	20	2
2	135001	Vips	20	2
3	134996	Sanborns Casa Piedra	20	2
4	135062	Restaurante El Cielo Potosino	20	2
5	135060	Restaurante Marisco Sam	20	2
6	132583	McDonalds Centro	20	2
7	135052	La Cantina Restaurante	20	2
8	132766	Mikasa	20	2
9	134986	Restaurant Las Mañanitas	20	2
0	132768	Mariscos Tia Licha	20	2

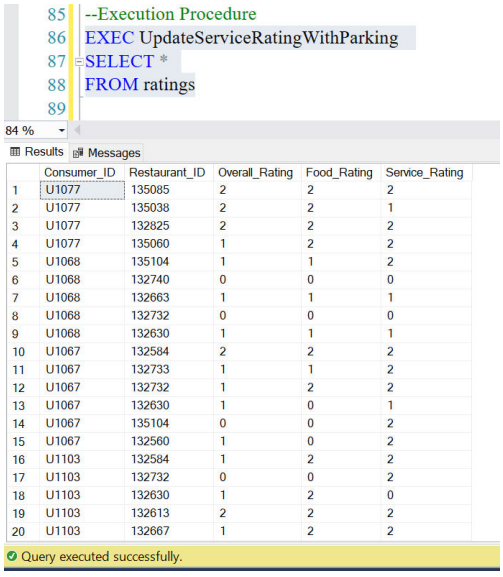
Query executed successfully.

QUESTION 5: A SQL Server utility called `UpdateServiceRatingWithParking` modifies the Service_Rating for eateries that provide parking in the `ratings table. For restaurants that have parking available, as specified in the restaurants table, either 'yes' or 'public', it sets the Service_Rating to 2. Creating a new stored procedure, running the SQL statement, updating ratings, setting Service_Rating to 2, defining a filter condition based on the `Restaurant_ID column, and choosing Restaurant_ID values from the `restaurants` table where the `Parking` column has values of 'yes' or 'public' are the various components of the procedure. The `END` clause indicates the conclusion of the stored procedure's body, and the `);` clause ends the `IN clause and the `UPDATE` statement.

```
72  --5. STORED PROCEDURE
73  CREATE PROCEDURE UpdateServiceRatingWithParking
74  AS
75  BEGIN
76      UPDATE ratings
77      SET Service_Rating = 2
78      WHERE Restaurant_ID IN (
79          SELECT Restaurant_ID
80          FROM restaurants
81          WHERE Parking IN ('yes', 'public')
82      );
83  END
```

```
essages
mmands completed successfully.
mpletion time: 2024-04-13T23:18:59.4707536+01:00
```

To update the `ServiceRating` field in the `ratings` table depending on the restaurant's parking availability, the `EXEC UpdateServiceRatingWithParking` statement runs a stored procedure. If parking is available, the stored procedure may raise the ratings. The `SELECT * FROM ratings` query displays all columns and rows after retrieving all records from the `ratings` table. These SQL statements and queries have different purposes when they are executed, and the specifics of each one impact the ratings` table based on how it is implemented.



```
85  --Execution Procedure
86  EXEC UpdateServiceRatingWithParking
87  SELECT *
88  FROM ratings
89
```

84 %

Results | Messages

	Consumer_ID	Restaurant_ID	Overall_Rating	Food_Rating	Service_Rating
1	U1077	135085	2	2	2
2	U1077	135038	2	2	1
3	U1077	132825	2	2	2
4	U1077	135060	1	2	2
5	U1068	135104	1	1	2
6	U1068	132740	0	0	0
7	U1068	132663	1	1	1
8	U1068	132732	0	0	0
9	U1068	132630	1	1	1
10	U1067	132584	2	2	2
11	U1067	132733	1	1	2
12	U1067	132732	1	2	2
13	U1067	132630	1	0	1
14	U1067	135104	0	0	2
15	U1067	132560	1	0	2
16	U1103	132584	1	2	2
17	U1103	132732	0	0	2
18	U1103	132630	1	2	0
19	U1103	132613	2	2	2
20	U1103	132667	1	2	2

Query executed successfully.

QUESTION 6.1: NESTED QUERIES-EXISTS

The `restaurants` table's `Name` and `City` columns are chosen by the main query, which aliases it as `r. The `ratings` table's constant value 1 is chosen by the subquery, which then filters records where the `Restaurant_ID` matches the `Restaurant_ID` from the `restaurants` table (aliased as `r) in the main query. The `EXISTS` clause verifies that the records that the subquery returned exist. The restaurants with at least one rating in the ratings table are identified by their Name and City.

```
91  --6.1 EXISTS
92  SELECT Name, City
93  FROM restaurants r
94  WHERE EXISTS (SELECT 1 FROM ratings WHERE r.Restaurant_ID = ratings.Restaurant_ID);
95
```

84 %

Results | Messages

	Name	City
1	Puesto de Gorditas	Ciudad Victoria
2	Cafe Ambar	Ciudad Victoria
3	Church's	Ciudad Victoria
4	Cafe Chaires	San Luis Potosi
5	McDonalds Centro	Cuernavaca
6	Gorditas Doña Tota	Ciudad Victoria
7	Tacos De Barbacoa Enfrente Del Tec	Ciudad Victoria
8	Hamburguesas La Perica	Ciudad Victoria
9	Pollo Frito Buenos Aires	Ciudad Victoria
10	Carnitas Mata	Ciudad Victoria
11	La Perica Hamburguesa	Ciudad Victoria
12	Palomo Tec	Ciudad Victoria
13	Carnitas Mata Calle 16 de Septiembre	Ciudad Victoria
14	Carnitas Mata Calle Emilio Portes Gil	Ciudad Victoria
15	Tacos Abi	Ciudad Victoria
16	Tacos Correcaminos	Ciudad Victoria
17	Little Pizza Emilio Portes Gil	Ciudad Victoria
18	Tacos El Guero	Ciudad Victoria
19	Gorditas Dona Tota	Ciudad Victoria
20	Tacos De La Estacion	Ciudad Victoria

Query executed successfully. LAPTOP-ACSAE0

QUESTION 6.2: NESTED QUERIES-IN

To filter the 'Consumers' table according to different cities in the 'Restaurants' table, the SQL query employs a subquery. The 'Restaurants' table is queried by the subquery to obtain a list of unique cities, making sure that each city appears only once. Using the 'City' column as a filter, the primary query pulls all columns from the 'Consumers' table. Based on the list of cities the subquery returned, the 'IN' keyword filters the 'City' column in the 'Consumers' table. This facilitates the identification and retrieval of customer information for the cities in which the business operates restaurants, enabling improved analysis and comprehension of customer behaviour.

```
97  --6.2 IN
98  SELECT *
99  FROM Consumers
100 WHERE City IN (SELECT DISTINCT City FROM restaurants);
101
```

94 %

Results | Messages

	Consumer_ID	City	State	Country	Latitude	Longitude	Smoker	Drink_Level	Transportation_Method	Marital_Status	Children	Age	Occupation	Budget
1	U1001	San Luis Potosi	San Luis Potosi	Mexico	22.1399974822998	-100.978805541992	0	Abstemious	On Foot	Single	Independent	23	Student	Medium
2	U1002	San Luis Potosi	San Luis Potosi	Mexico	22.1500873565674	-100.983322143555	0	Abstemious	Public	Single	Independent	22	Student	Low
3	U1003	San Luis Potosi	San Luis Potosi	Mexico	22.1198463439941	-100.94652557373	0	Social Drinker	Public	Single	Independent	23	Student	Low
4	U1004	Cuernavaca	Morelos	Mexico	18.867000579834	-99.1829986572266	0	Abstemious	Public	Single	Independent	72	Employed	Medium
5	U1005	San Luis Potosi	San Luis Potosi	Mexico	22.1834774017334	-100.959892272949	0	Abstemious	Public	Single	Independent	20	Student	Medium
6	U1006	San Luis Potosi	San Luis Potosi	Mexico	22.1499996185303	-100.983001708984	1	Social Drinker	Car	Single	Independent	23	Student	Medium
7	U1007	San Luis Potosi	San Luis Potosi	Mexico	22.1184635162354	-100.938255310059	0	Casual Drinker	Public	Single	Independent	23	Student	Low
8	U1008	San Luis Potosi	San Luis Potosi	Mexico	22.122989654541	-100.923812866211	0	Social Drinker	Public	Single	Independent	23	Student	Low
9	U1009	San Luis Potosi	San Luis Potosi	Mexico	22.1594276428223	-100.990447998047	0	Abstemious	On Foot	Single	Kids	21	Student	Medium
10	U1010	San Luis Potosi	San Luis Potosi	Mexico	22.1908893585205	-100.998672485352	0	Social Drinker	Car	Married	Kids	25	Student	Medium

SYSTEM FUNCTIONS
QUESTION 6.4: USE OF GROUP BY, HAVING AND ORDER BY CLAUSES

The 'Country' column is retrieved from the'restaurants' table using the SQL query SELECT Country, COUNT(*) AS Total_Restaurants. The result column is aliased as 'Total_Restaurants' and counts the number of rows for each group. The'restaurants' table is where the data is taken from. The 'HAVING COUNT(*) > 5' filter restricts the results to those countries where the count of restaurants is greater than five. The 'GROUP BY Country' clause groups the results based on the 'Country' column. The countries with the most restaurants are at the top of the 'ORDER BY Total_Restaurants DESC' order, which arranges the result set in descending order. The overall goal is to show the number of restaurants in each country that has more than five.

```
102  --6.3 SYSTEM FUNCTION GROUP BY,HAVING
103  SELECT Restaurant_ID, COUNT(*) AS Total_Ratings
104  FROM ratings
105  GROUP BY Restaurant_ID
106  HAVING COUNT(*) > 10;
```

4 %

Results | Messages

	Restaurant_ID	Total_Ratings
1	132921	17
2	135028	15
3	132572	15
4	135045	13
5	135051	14
6	135108	11
7	135085	36
8	135025	15
9	135042	20
10	135062	21
11	135057	15
12	132856	14
13	135026	11
14	132862	18
15	135069	12
16	135060	22
17	132830	12
18	135046	11
19	135066	12
20	135038	24

Query executed successfully.