

ADVANCED DATABASE TASK 1

INTRODUCTION

The effective management of library has to do with a lot of tasks which include but are not limited to fines, loans, item, repayment. Etc. Actually this can be handled using database management system (DBMS) to improve the level of accuracy and data consistency. To do this it is important to streamline the overall processes of managing the data of a library which include books, DVD, as well as journal etc. As a database developer, I am tasked with the designing of a new database to store the information of members, their loan history, item collected and overdue payment. Etc. To do this task, the system should be designed in such a way that it is able to track the status of the item in order to know if the item is available, lost or removed. The system should also be designed to handle logins as well as member information even after a member decides to leave. In this task, the major goal is to design database that is quite efficient and one which can be relied on to meet the needs of the library. Ensuring data integrity to avoid system downtimes will help to prevent losing critical data. Being able to identify the entities and their relationship and to define the data attributes with their respective data types for each entity will enable me to analyse library's requirements and breaking down the information into discrete data elements. When designing this database, it is necessary to consider a database schema where all the necessary information of the members is being stored. To achieve this goal, I will need to consider the specific requirements of the library and create a database design that is flexible, scalable, and easy to maintain.

Design and normalization of the proposed database into 3NF

To determine if the proposed database is in 3NF, it is important to note that each of the table that was designed has a primary key, and non-key attributes that are dependent only on the primary key. For example, the Member table has attributes that are dependent on the MemberId, which is the primary key. This is also applicable to the Item table which has attributes that are dependent on the ItemId, which is the primary key. Establishing a many-to-many relationship between members and items, you will see that the Loan table has foreign keys that reference the Member and Item tables. The Overdue table also has foreign keys that reference the Member and Item tables, respectively. Finally, the Repayment table has a foreign key that references the Overdue table to track the repayment of overdue amounts associated with specific overdue. Also, there are no transitive dependencies. From the above illustration, the code used for this task is in 3NF.

DATABASE DESIGN DECISION.

The code used in this task creates a database named "LibraryManagementSystem" and defines five tables which include Member, Item, Loan, Overdue, and Repayment. I shall be explaining this one after the other.

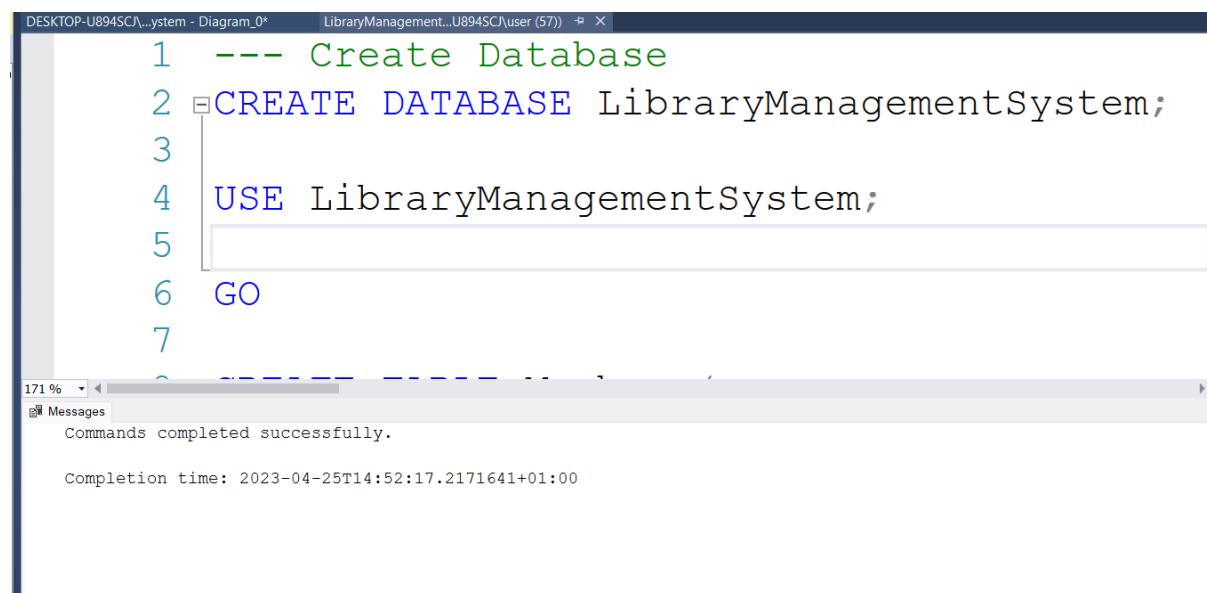
1. Member Table: I shall be designing the "Member table" to store information about the library's members. Each member will be linked and associated by a unique 'MemberId' and login details which include the username and password. The members name which includes the "Username", "Password", to store login details, "FirstName", "MiddleName", and "LastName" columns store the member's name, while the "Address1" and "Address2" columns store their address. The Dateofbirth column will stores the member's date of birth, while the EmailAddress and ContactNumber columns will also store their contact information. The MembershipEndDate column stores the date on which the member's membership expires.
2. Item Table: Designing the item table will store information about the items in the library. Each item is associated by a special ItemId. In this context, the "ItemTitle" column will store the title of the item, while the "ItemType" column stores the type of item like book, CD, DVD, journal. The Author column will help to store the name of the author of the item listed above while the "YearOfPublication" column stores the year in which the item was published by the author. The Status column stores the status of the item to know if the item is available, lost or removed. while the ISBN column stores the International Standard Book Number of the item. The ItemAddedDate column stores the date on which the item was added to the library's collection.
3. Loan Table: The information about the loans collected and paid by the members shall be stored on the loan table. Similarly, like other tables, each loan is identified by a unique LoanId. The MemberId column will reference the Member table to associate the loan with a specific member. The ItemId column references the Item table to associate the loan with a specific item. The ItemBorrowedDate column stores the date on which the item was borrowed, The ItemDueDate will store the information on the due date it was returned. The ItemReturnedDate column stores the date on which the item was returned. The OverdueRate column stores the rate at which overdue fees are charged to members who fails to meet the deadline of the return date.
4. Overdue Table: Each overdue item is identified by a special OverduelId. The MemberId column will reference the Member table to associate the overdue item with a specific member. The ItemId column will also reference the Item table to associate the overdue item with a specific item. The date and time in which the item became overdue will be stored on the "OverdueDateTime" column. while the OverdueTotalAmount column stores the total amount of overdue fees associated with the item.

5. Repayment Table: A table shall be designed to enable members make payment for their overdue item. Each repayment made is identified by a unique "RepaymentId". The OverdueId column references the Overdue table to associate the repayment with a specific overdue item. The RepaymentDateTime column stores the date and time on which the repayment was made by the member. while the RepaymentAmount column stores the amount of the repayment. The PaymentMethod column stores the method of payment that was used. This could be cash or card.

IMPLEMENTATION OF THE DESIGN USING T-SQL STATEMENT

The execution of any database design has to do with creating a new database in Microsoft SQL Server Management Studio and defining the tables and relationships that was obtained based on the design. The database design used in this task includes five tables: Member, Item, Loan, Overdue, and Repayment along with their respective fields and primary keys. The steps involve in creating this database are listed below showing that the command was successfully completed along with the screenshot.

1. Create The Database using the CREATE DATABASE Statement below.



The screenshot shows a Microsoft SQL Server Management Studio window titled "DESKTOP-U894SC\user - Diagram_0*". The query pane contains the following T-SQL code:

```
1 --- Create Database
2 CREATE DATABASE LibraryManagementSystem;
3
4 USE LibraryManagementSystem;
5
6 GO
7
```

The status bar at the bottom indicates "Commands completed successfully." and "Completion time: 2023-04-25T14:52:17.2171641+01:00".

The statement "CREATE DATABASE Library Management System" was used to create a database with the name "LibraryManagementSystem". Upon creation, this database can now be used to store and manage data related to a library management system, such as information about books, borrowers, loans, etc.

2. Use the Database

The screenshot shows a SQL Server Management Studio window. The title bar indicates the connection is to 'LibraryManagement...U894SC\user (54)' and the database is 'DESKTOP-U894SC\...\system - Diagram_0'. The main pane displays the following SQL script:

```
1 --- Create Database
2 CREATE DATABASE LibraryManage
3
4 USE LibraryManagementSystem;
5
6 GO
```

Below the script, the message 'Commands completed successfully.' is displayed, along with the completion time: 'Completion time: 2023-04-17T13:07:08.8813208+01:00'.

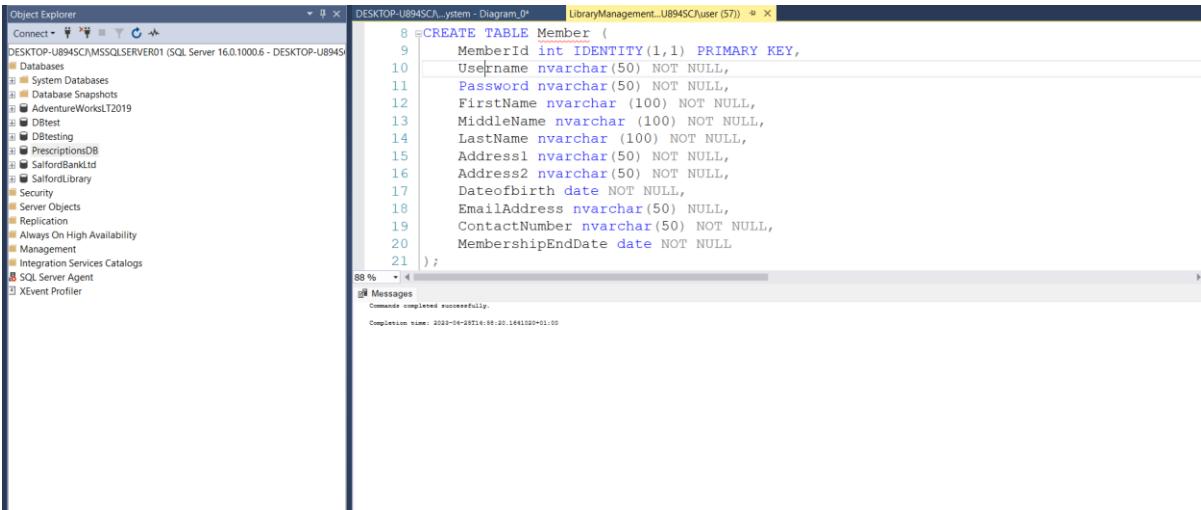
To select a particular database to work with in a SQL server, I shall use The SQL statement "USE". In this scenario, the statement "USE LibraryManagementSystem" is used to select the specific database "LibraryManagementSystem" that was created earlier using the "CREATE DATABASE" statement. Once this statement has been executed, any subsequent SQL statements will be executed within the context of the "LibraryManagementSystem" database,

The screenshot shows a SQL Server Management Studio window with the 'Object Explorer' pane on the left. The 'Databases' node is expanded, showing various databases including 'AdventureWorksLT2019', 'DBtest', 'DBtesting', 'LibraryManagementSystem', 'PrescriptionsDB', 'SalfordBankLtd', and 'SalfordLibrary'. The main pane displays the same SQL script as the previous screenshot:

```
1 --- Create Database
2 CREATE DATABASE LibraryManage
3
4 USE LibraryManagementSystem;
5
6 GO
```

Below the script, the message 'Commands completed successfully.' is displayed, along with the completion time: 'Completion time: 2023-04-25T14:52:17.2171641+01:00'.

3. Create The Member Table



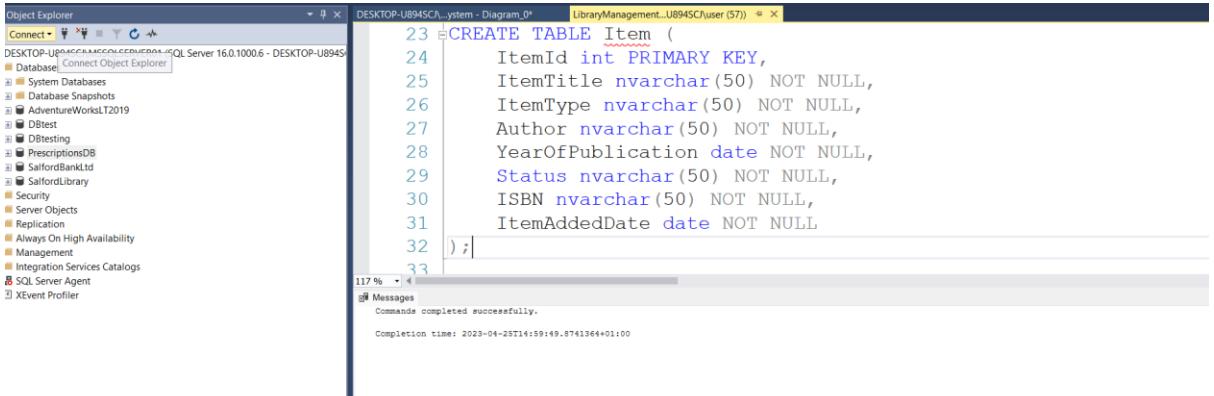
The screenshot shows the Object Explorer on the left and the Script pane on the right. The Object Explorer lists several databases, including 'AdventureworksLT2019', 'DBtest', 'DBtesting', 'PrescriptionDB', 'SalfordBankLtd', and 'SalfordLibrary'. The Script pane contains the following SQL code:

```
8 CREATE TABLE Member (
9     MemberId int IDENTITY(1,1) PRIMARY KEY,
10    Username nvarchar(50) NOT NULL,
11    Password nvarchar(50) NOT NULL,
12    FirstName nvarchar(100) NOT NULL,
13    MiddleName nvarchar(100) NOT NULL,
14    LastName nvarchar(100) NOT NULL,
15    Address1 nvarchar(50) NOT NULL,
16    Address2 nvarchar(50) NOT NULL,
17    DateOfBirth date NOT NULL,
18    EmailAddress nvarchar(50) NULL,
19    ContactNumber nvarchar(50) NOT NULL,
20    MembershipEndDate date NOT NULL
21 ) ;
```

The status bar at the bottom indicates: 'Commands completed successfully.' and 'Completion time: 2023-04-28T14:59:10.1641020+01:00'.

After the above command has been successfully executed, I shall be creating the "member table". In this case, the statement "CREATE TABLE Member" is used to create a table named "Member" with several columns, which are MemberId, FirstName, MiddleName, LastName, Address1, Address2, Dateofbirth, EmailAddress, ContactNumber, and MembershipEndDate. The primary key constraint is applied to the MemberId column, which will ensure that each row in the table has a special MemberId value. The "NOT NULL" constraint is applied to several columns, which means that these columns cannot contain null values. The NULL constraint is applied only to the EmailAddress column, which means that this column can contain null values.

4. Create the Item Table



The screenshot shows the Object Explorer on the left and the Script pane on the right. The Object Explorer lists the same databases as the previous screenshot. The Script pane contains the following SQL code:

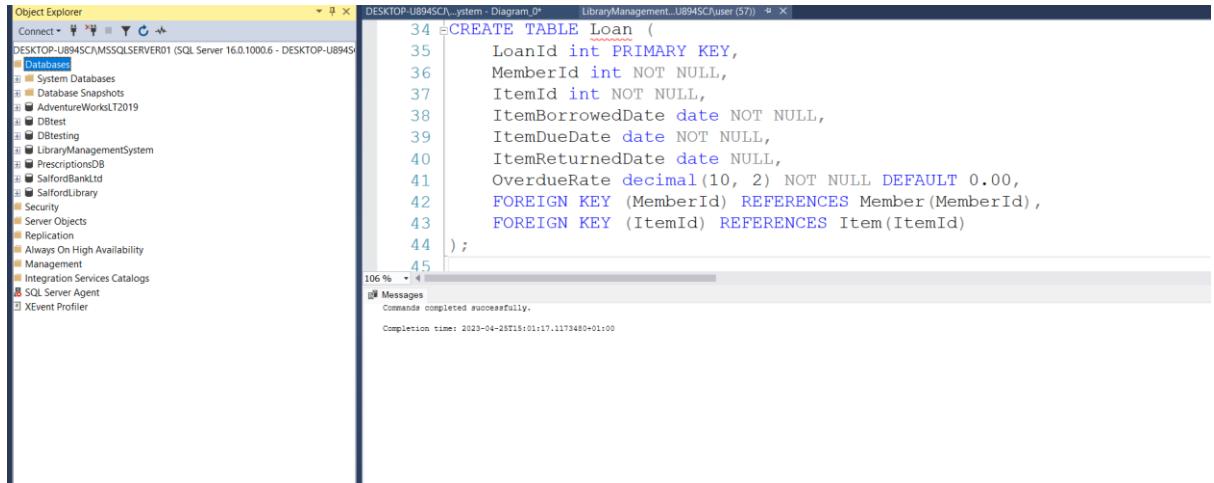
```
23 CREATE TABLE Item (
24     ItemId int PRIMARY KEY,
25     ItemTitle nvarchar(50) NOT NULL,
26     ItemType nvarchar(50) NOT NULL,
27     Author nvarchar(50) NOT NULL,
28     YearOfPublication date NOT NULL,
29     Status nvarchar(50) NOT NULL,
30     ISBN nvarchar(50) NOT NULL,
31     ItemAddedDate date NOT NULL
32 ) ;
```

The status bar at the bottom indicates: 'Commands completed successfully.' and 'Completion time: 2023-04-28T14:59:49.8741364+01:00'.

The Item table shall be created using the SQL statement above. In this case, the statement "CREATE TABLE Item" is used to create a table named "Item" which are made up of different columns, including ItemId, ItemTitle, ItemType, Author, YearOfPublication, Status, ISBN, and ItemAddedDate. The primary key constraint is

applied to the ItemId column as seen above. The “NOT NULL” constraint is applied to all columns. This simply means that these columns cannot contain null values.

5. Create the Loan Table



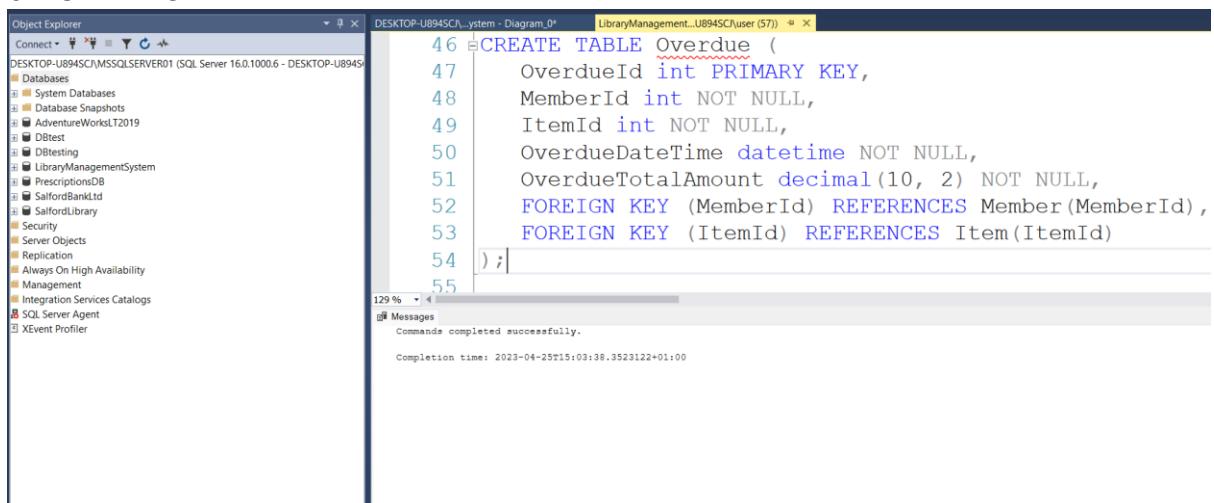
The screenshot shows the Object Explorer on the left with various databases listed. The central pane displays the T-SQL code for creating the 'Loan' table:

```
34  CREATE TABLE Loan (
35      LoanId int PRIMARY KEY,
36      MemberId int NOT NULL,
37      ItemId int NOT NULL,
38      ItemBorrowedDate date NOT NULL,
39      ItemDueDate date NOT NULL,
40      ItemReturnedDate date NULL,
41      OverdueRate decimal(10, 2) NOT NULL DEFAULT 0.00,
42      FOREIGN KEY (MemberId) REFERENCES Member(MemberId),
43      FOREIGN KEY (ItemId) REFERENCES Item(ItemId)
44  );
45
```

The status bar at the bottom indicates "Commands completed successfully." and "Completion time: 2023-04-23T15:01:17.1173480+01:00".

A table shall be created for the “loan” using the SQL the statement "CREATE TABLE Loan" which consists of several columns, including LoanId, MemberId, ItemId, ItemBorrowedDate, ItemReturnedDate, ItemDueDate, and OverdueRate. The primary key constraint is applied to the LoanId column, which ensures that each row in the table has a unique LoanId value. The “Itemreturneddate” column is NULL VALUES. While the rest column is “NOT NULL” which ensures that these columns cannot contain null values.

6. Create Overdue Table



The screenshot shows the Object Explorer on the left with various databases listed. The central pane displays the T-SQL code for creating the 'Overdue' table:

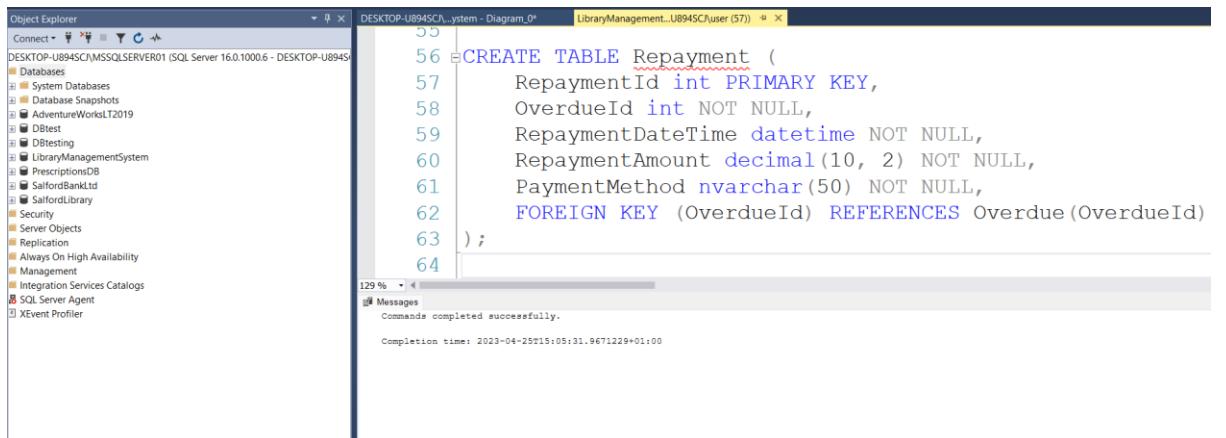
```
46  CREATE TABLE Overdue (
47      OverdueId int PRIMARY KEY,
48      MemberId int NOT NULL,
49      ItemId int NOT NULL,
50      OverdueDateTime datetime NOT NULL,
51      OverdueTotalAmount decimal(10, 2) NOT NULL,
52      FOREIGN KEY (MemberId) REFERENCES Member(MemberId),
53      FOREIGN KEY (ItemId) REFERENCES Item(ItemId)
54  );
55
```

The status bar at the bottom indicates "Commands completed successfully." and "Completion time: 2023-04-25T15:03:38.3523122+01:00".

After executing the loan table command, the overdue table shall be created. The statement "CREATE TABLE Overdue" is used to create a table named "Overdue"

with several columns, including OverdueId, MemberId, ItemId, OverdueDateTime, and OverdueTotalAmount. The primary key constraint is applied to the OverdueId column, which ensures that each row in the table has a perculiar OverdueId value. The NOT NULL constraint is applied to several columns, which ensures that these columns cannot contain null values.

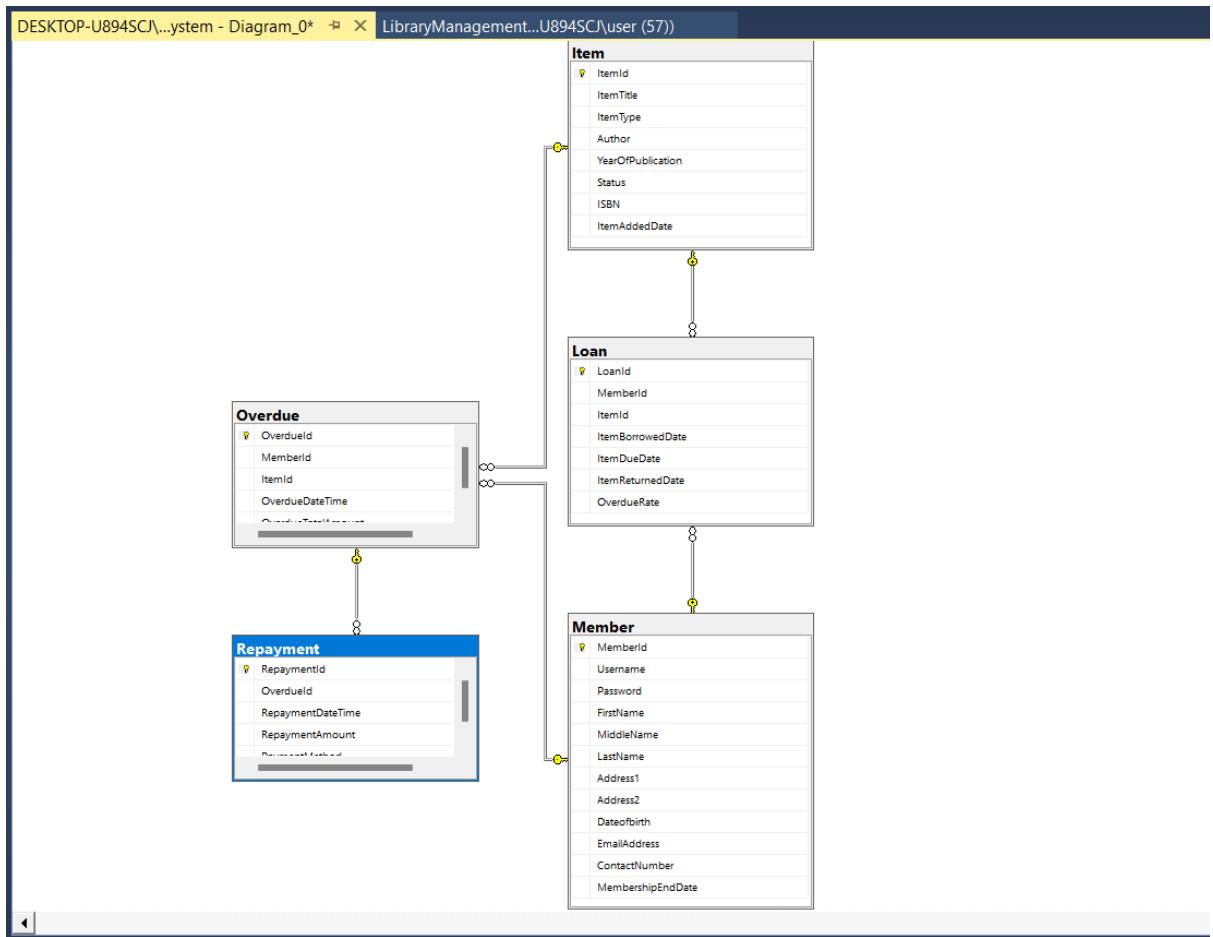
7. Create Repayment Table



The screenshot shows the Object Explorer on the left with various databases listed. The central pane displays a T-SQL script for creating the 'Repayment' table. The script includes columns for RepaymentId (primary key), OverdueId (NOT NULL), RepaymentDateTime (NOT NULL), RepaymentAmount (NOT NULL), and PaymentMethod (NOT NULL). A FOREIGN KEY constraint named 'Overdue' is defined, linking the OverdueId column in the Repayment table to the OverdueId column in the Overdue table. The execution message at the bottom indicates the command completed successfully.

```
CREATE TABLE Repayment (
    RepaymentId int PRIMARY KEY,
    OverdueId int NOT NULL,
    RepaymentDateTime datetime NOT NULL,
    RepaymentAmount decimal(10, 2) NOT NULL,
    PaymentMethod nvarchar(50) NOT NULL,
    FOREIGN KEY (OverdueId) REFERENCES Overdue(OverdueId)
);
```

The repayment table shall be created using The SQL statement "CREATE TABLE Repayment" This table consist of several columns, including RepaymentId, OverdueId, RepaymentDateTime, RepaymentAmount, and PaymentMethod. The primary key constraint is applied to the RepaymentId column, which ensures that each row in the table has a unique RepaymentId value for members. The NOT NULL constraint is applied to several columns, which ensures that these columns cannot contain null values.



Each of the table shown above is defined with appropriate data types of right constraints, and the appropriate foreign key relationships to ensure data integrity. By implementing this design, the library can effectively manage its members, items, and loans, while also tracking any overdue items and associated fees.

COLUMN(S) OF PRIMARY KEYS AND FOREIGN KEYS

In this task, the primary keys and foreign keys used for this database design are listed below.

- Primary keys Columns:
 - Member table: MemberId
 - Item table: ItemId
 - Loan table: LoanId
 - Overdue table: OverduelId
 - Repayment table: RepaymentId
- Foreign keys Columns:
 - Loan table: MemberId (references Member table MemberId)

- Loan table: ItemId (references Item table ItemId)
- Overdue table: MemberId (references Member table MemberId)
- Overdue table: ItemId (references Item table ItemId)
- Repayment table: Overdueld (references Overdue table Overdueld)

DATA TYPE USED FOR EACH COLUMN

This shall be summarised in a tabular form.

1. MEMBER TABLE:

Columns	DATA TYPES	JUSTIFICATIONS
MemberId (PK)	Int (IDENTITY (1,1))	The columns can store whole number of values that are unique for each member.
UserName	Nvarchar (50)	It allows storing Unicode characters and has a variable length.
Password	Nvarchar (50)	It allows storing Unicode characters and has a variable length.
FirstName	Nvarchar (100)	The data stored in this column does not exceed a certain limit
MiddleName	Nvarchar (100)	This is set to ensure that the data stored in this column does not exceed a certain limit.
LastName	Nvarchar (100)	maximum length of 100 characters. This is set to ensure

		that the data stored in this column does not exceed a certain limit.
Address1	Nvarchar (50)	with a maximum length of 50 characters" is set to ensure that the data in this column does not exceed certain limit.
Address2	Nvarchar (50)	a maximum length of 50 characters" is set to ensure that the data in this column does not exceed certain limit.
Dateofbirth	Date	store date values in the format of YYYY-MM-DD. Using a date data type for the Dateofbirth column allows for efficient storage and manipulation of date values.
ContactNumber	Nvarchar (50)	The maximum length of 50 characters is set to ensure that the data stored in this column does not exceed a certain limit.
EmailAddress	Nvarchar (50)	The maximum length of 50 characters is set to ensure that the data stored in this column does not exceed a certain limit.
MembershipEndDate	Date	A date data type used to store the end date of the member's membership.

2. ITEM TABLE

COLUMNS	DATA TYPES	JUSTIFICATIONS
ItemId (PK)	Int	Allows for storing whole numbers that uniquely define items in the library system. INT is used as the data type because it's a whole number.
Item Title	Nvarchar (50)	To store title of an item. Its set 50 to allow for longer item titles.
Item Type	Nvarchar (50)	It's set 50 to allow for longer character type. It

		allows for storing text data representing
Author	Nvarchar (50)	A string data type to store the author of an item. It will allow for sorting text data representing the name of the authors
Year of Publication	Date	Allows for storing the year of publication of each item in the library system. It represents a calendar date (year, month and day)
Status	Nvarchar (50)	Using this data type for the status column allows for storing a short description of the current status of each item in the library system
ISBN	Nvarchar (50)	The International Standard Book Number (ISBN) is a unique identifier for books and related materials. It contains 13 digits. The ISBN can contain both numbers and letters, making it suitable for storing as a string data type in SQL.
Item Added Date	Date	Date data type ensures that the ItemAddedDate column only stores valid date values, and any invalid date values will result in an error when inserted into the table.

3. LOAN TABLE

COLUMNS	DATA TYPE	JUSTIFICATION
---------	-----------	---------------

LoanId (PK)	Int	INT is used as the data type because it's a whole number.
MemberId	Int (Not Null)	data type is appropriate because the values will be whole numbers.
ItemId	Int (Not Null)	The int data type is appropriate because the values will be whole numbers.
Itemborroweddate	Date (Not Null)	The date data type is appropriate because it stores only the date without the time.
ItemDueDate	Date (Not Null)	The date data type is appropriate because it stores only the date without the time.
ItemReturnedDate.	Date (Null)	The date data type is appropriate because it stores only the date without the time, and the NULL value is allowed because the item may not have been returned yet.
Overdue rate	Decimal 10, 2) (NOT NULL DEFAULT 0.00)	it allows for storing decimal values, such as interest rates, with high precision and accuracy. The DEFAULT constraint ensures that the value is set to 0.00 by default if not explicitly specified.

4. OVERDUE TABLE

Columns	DATA TYPE	JUSTIFICATION
---------	-----------	---------------

Overdueld (Pk)	Int	data type is appropriate because the values will be whole numbers, and the PRIMARY KEY constraint ensures that each record has a unique value.
MemberId	Int (Not Null)	the values will be whole numbers.
ItemId	Int (Not Null)	The int data type is appropriate because the values will be whole numbers.
OverdueDateTime	DateTime (Not Null)	The datetime data type is appropriate because it allows for storing both date and time information.
OverdueTotalAmount	decimal (10, 2) (NOT NULL):	The decimal data type is appropriate because it allows for storing decimal values, such as money, with high precision and accuracy.

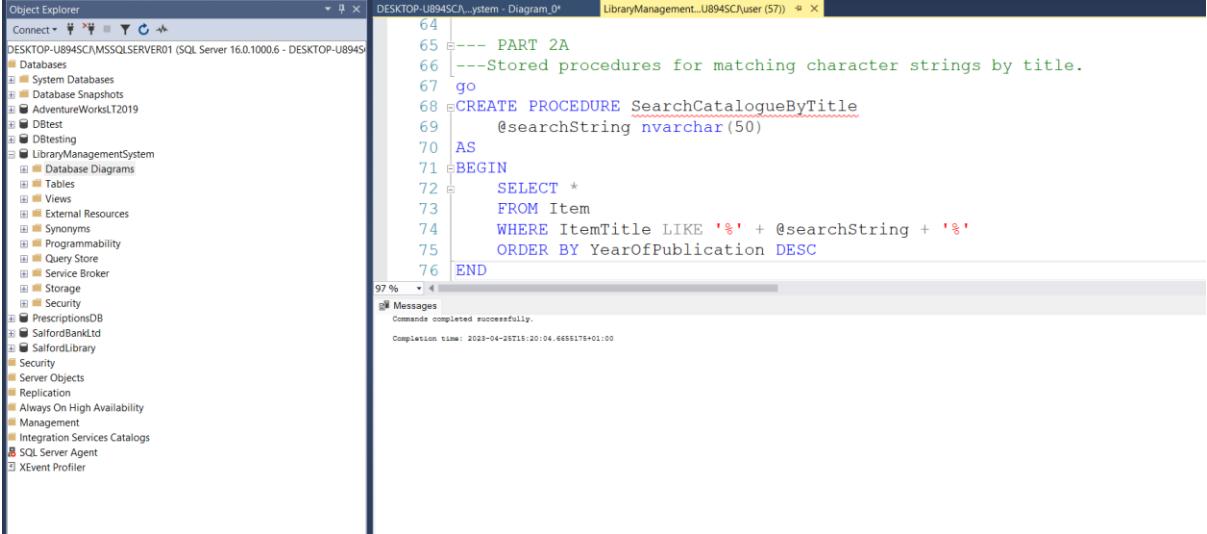
5. REPAYMENT TABLE

COLUMNS	DATA TYPE	JUSTIFICATION
RepaymentId (pk)	Int	The int data type is appropriate because the values will be whole numbers, and the PRIMARY KEY constraint ensures that each record has a unique value.

Overdueld	Int (Not Null)	The int data type is appropriate because the values will be whole numbers.
RepaymentDateTime	DateTime (Not Null)	The datetime data type is appropriate because it allows for storing both date and time information.
RepaymentAmount	Decimal (10, 2) (NOT NULL)	The decimal data type is appropriate because it allows for storing decimal values, such as money, with high precision and accuracy.
PaymentMethod	Nvarchar (50) (NOT NULL)	The method used for making the repayment (e.g., cash, credit card, etc.). The nvarchar data type is appropriate because it allows for storing string values of variable length up to 50 characters.

PART 2A

This stored procedure for matching character strings by title usually takes a search string as input. It will return all items from the Item table whose 'ItemTitle' column contains the search string. The 'ORDER BY' clause will then sort the results obtained by the 'YearOfPublication' column in descending order, which means the most recent publications among the item will appear first.



```

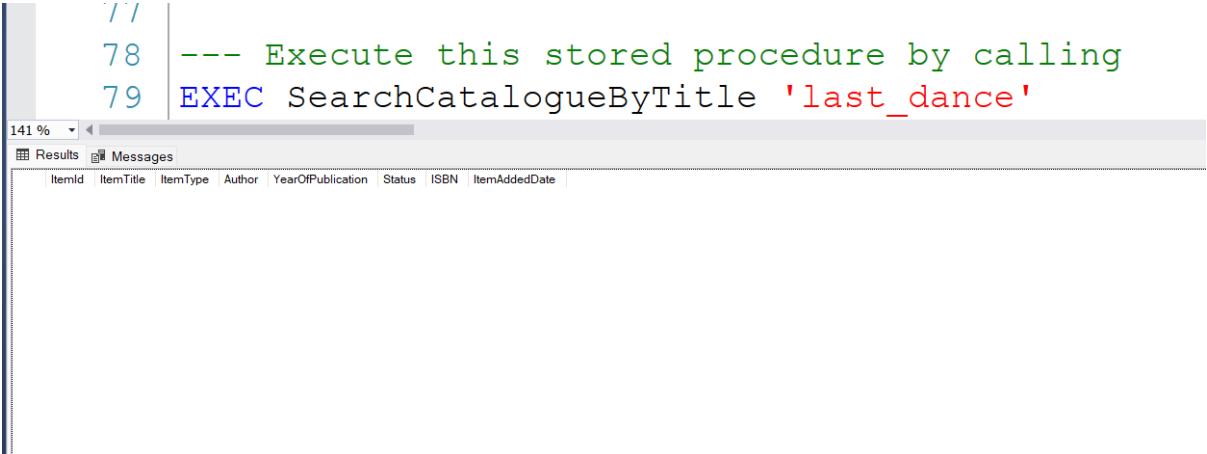
64
65 --- PART 2A
66 --- Stored procedures for matching character strings by title.
67 go
68 CREATE PROCEDURE SearchCatalogueByTitle
69   @searchString nvarchar(50)
70 AS
71 BEGIN
72   SELECT *
73   FROM Item
74   WHERE ItemTitle LIKE '%' + @searchString + '%'
75   ORDER BY YearOfPublication DESC
76 END
    
```

Messages

Commands completed successfully.

Completion time: 2023-04-26T15:20:04.6655175+01:00

Executing the below code will return all items that have "Last Dance" in the title, sorted with the most recent publications first.



```

77
78 --- Execute this stored procedure by calling
79 EXEC SearchCatalogueByTitle 'last_dance'
    
```

PART 2B

Here, the "ItemReturnedDate is NULL". Therefore, this stored procedure will retrieve all loan records from the 'Loan table' that are not yet returned as well as the corresponding item and member information from the 'Item' and 'Member' tables using JOINs. A filter to return only those loans whose due date is less than five days

from the current date is applied which is achieved using the 'DATEDIFF' function to calculate the difference in days between the current date '(GETDATE ())' and the due date '(ItemDueDate)'.

```

81 81 ---PART 2B
82 82 ---Stored procedure to return a full list of all items currently on loan which
83 83 CREATE PROCEDURE GetItemsDueSoon
84 84 AS
85 85 BEGIN
86 86     SELECT *
87 87         FROM Loan
88 88             JOIN Item ON Loan.ItemId = Item.ItemId
89 89             JOIN Member ON Loan.MemberId = Member.MemberId
90 90             WHERE ItemReturnedDate IS NULL
91 91                 AND DATEDIFF(day, GETDATE(), ItemReturnedDate) < 5
92 92 END
93 93

```

Messages
Commands completed successfully.
Completion time: 2023-04-28T15:55:54.5530187+01:00

97 % 97 %

Query executed successfully.

This will return a list of all items currently on loan which have a due date of less than five days from the current date.

```

94 94 ---You can execute this stored procedure by executing:
95 95 EXEC GetItemsDueSoon
96 96

```

129 %

Results Messages

LoanId	MemberId	ItemId	ItemBorrowedDate	ItemDueDate	ItemReturnedDate	OverdueRate	ItemId	ItemTitle	ItemType	Author	YearOfPublication	Status	ISBN	ItemAddedDate	MemberId	Username	Pa

PART 2C

The code below is a stored procedure to inserts a new into the "Member" table in the database. The stored procedure takes parameters which include the details of the new member such as member ID, username, password, first name, middle name, last name, address, date of birth, email address, contact number, and membership end date. These parameters are used to insert a new row into the "Member" table using the INSERT statement. This stored procedure can be executed by passing in the required parameters, as this will automatically insert a new member into the database.

The screenshot shows a code editor window in Microsoft SQL Server Management Studio. The code is a T-SQL script for creating a stored procedure:

```
98 ---- PART 2C
99 --- Stored procedures or user-defined functions to Insert a new member into the database
100 GO
101 CREATE PROCEDURE InsertNewMember
102     @Username nvarchar(50),
103     @Password nvarchar(50),
104     @FirstName nvarchar(100),
105     @MiddleName nvarchar(100),
106     @LastName nvarchar(100),
107     @Address1 nvarchar(50),
108     @Address2 nvarchar(50),
109     @DateOfBirth date,
110     @EmailAddress nvarchar(50),
111     @ContactNumber nvarchar(50),
112     @MembershipEndDate date
113 AS
114 BEGIN
115     INSERT INTO Member (Username, Password, FirstName, MiddleName, LastName, Address1, Address2, DateOfBirth, EmailAddress, ContactNumber, MembershipEndDate)
116     VALUES (@Username, @Password, @FirstName, @MiddleName, @LastName, @Address1, @Address2, @DateOfBirth, @EmailAddress, @ContactNumber, @MembershipEndDate)
117 END;
118
```

The status bar at the bottom indicates "Commands completed successfully." and "Completion time: 2023-04-25T15:36:10.9243504+01:00".

For example, this stored procedure takes the appropriate parameters as listed above for the new member's information and inserts a new row into the 'Member' table.

The screenshot shows the execution of the stored procedure:

```
120 --- Execute this stored procedure of the above code by calling
121 EXEC InsertNewMember 'jdoe', 'mypassword', 'John', 'F', 'Donald',
122
```

The status bar at the bottom indicates "(1 row affected)" and "Completion time: 2023-04-25T15:36:10.9243504+01:00".

This will insert a new member with the specified information into the 'Member' table.

PART 2D

The stored procedure to update the details of an existing member will take the members ID and set the values for specific field to the corresponding input parameter values and update all the input parameters of the member by performing an update operation on the member table. This allow for efficient update of members which are subject to change.

DESKTOP-U894SC\user - Diagram_0* LibraryManagement..U894SC\user (57) ✘ X

```
123 ---PART 2D
124 ---Stored procedures to update the details for an existing member
125 CREATE PROCEDURE UpdateMemberDetails
126     @memberId int,
127     @Username nvarchar(50),
128     @Password nvarchar(50),
129     @firstName nvarchar(100),
130     @middleName nvarchar(100),
131     @lastName nvarchar(100),
132     @address1 nvarchar(50),
133     @address2 nvarchar(50),
134     @dateOfBirth date,
135     @emailAddress nvarchar(50),
136     @contactNumber nvarchar(50),
```

88 %

Messages

Commands completed successfully.

Completion time: 2023-04-25T15:33:48.9500279+01:00

DESKTOP-U894SCJ\...ystem - Diagram_0* LibraryManagement...U894SCJ\user (57) ▾ X

```
137     @membershipEndDate date
138 AS
139 BEGIN
140     UPDATE Member
141     SET Username = @Username,
142         Password = @Password,
143         FirstName = @firstName,
144         MiddleName = @middleName,
145         LastName = @lastName,
146         Address1 = @address1,
147         Address2 = @address2,
148         DateOfBirth = @dateOfBirth,
149         EmailAddress = @emailAddress,
150         ContactNumber = @contactNumber,
151         MembershipEndDate = @membershipEndDate
152     WHERE MemberId = @memberId
153 END
```

73 %

Messages

Commands completed successfully.

Completion time: 2023-04-25T15:33:48.9500279+01:00

For example, executing input parameters for the updated member information and updates the corresponding row in the ‘Member’ table based on the member’s ID as shown below.

```

155 |
156 | --- Execute this stored procedure by calling
157 | EXEC UpdateMemberDetails 1234, 'new_username', 'new_password', 'Jude', '
158 |
159 |
106 % 4 Messages
(0 rows affected)
Completion time: 2023-04-25T15:39:45.0424855+01:00

```

QUESTION 3

```

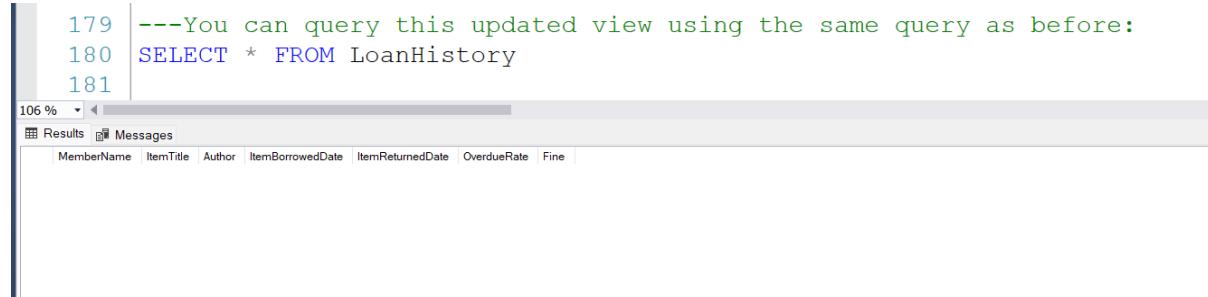
DESKTOP-U894SCJ\...system - Diagram_0* LibraryManagement...U894SCJ\user (57) X
158 |
159 |
160 | --- QUESTION 3
161 | ---A view showing all previous and current loans including details of the item borrowed, borrowed date, due date and any associated
162 | CREATE VIEW LoanHistory AS
163 | SELECT TOP 100 PERCENT
164 |     Member.FirstName + ' ' + Member.LastName AS MemberName,
165 |     Item.ItemTitle,
166 |     Item.Author,
167 |     Loan.ItemBorrowedDate,
168 |     Loan.ItemReturnedDate,
169 |     Loan.OverdueRate,
170 |     CASE
171 |         WHEN Loan.ItemReturnedDate IS NULL AND Loan.ItemReturnedDate < GETDATE() THEN DATEDIFF(day, Loan.ItemDueDate, GETDATE()) *
172 |             ELSE 0
173 |     END AS Fine
174 | FROM Loan
175 | INNER JOIN Member ON Loan.MemberId = Member.MemberId
176 | INNER JOIN Item ON Loan.ItemId = Item.ItemId
177 | ORDER BY MemberName, Loan.ItemBorrowedDate DESC;
178 |
60 % 4 Messages
Commands completed successfully.
Completion time: 2023-04-25T15:43:49.9933332+01:00

```

To create a view that will show the current and past loans, it will include the members full name, item title, author, borrowed date, returned date and any associated fines connected with it. The fine column will sum up the fine amount based on the

overdue rate and the number of days that have exceeded past the due date. This view will join the Loan, Member, and Item tables together to retrieve the necessary information for each loan linked to a specific member.

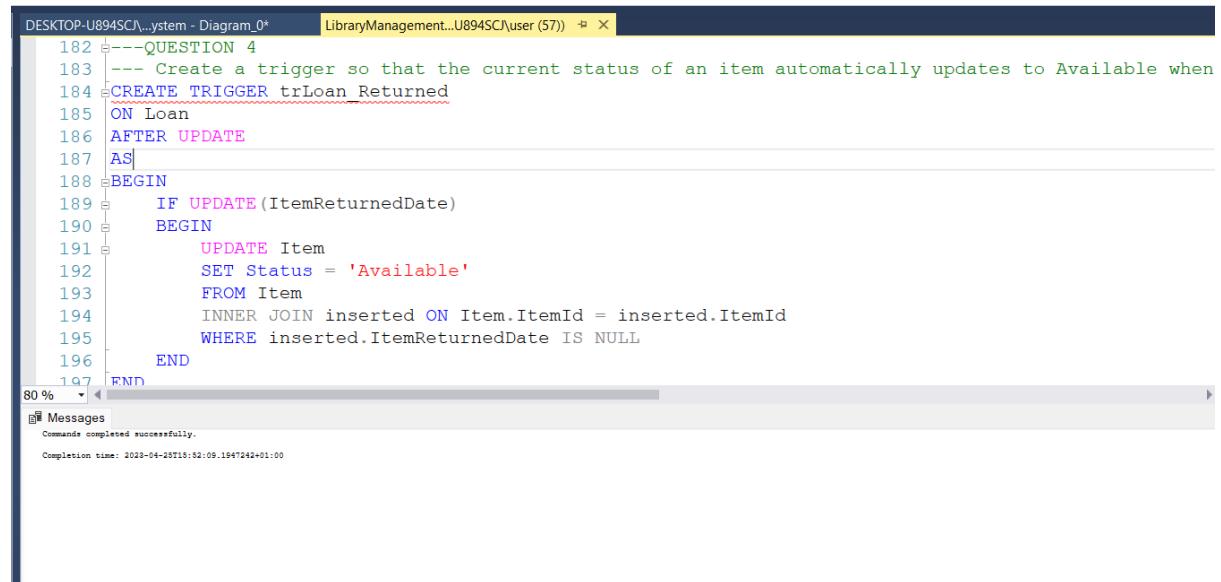
You can query this updated view using the code below.



```
179 ---You can query this updated view using the same query as before:
180 SELECT * FROM LoanHistory
181
```

The screenshot shows a SQL query window with the following details:
- Title bar: DESKTOP-U894SC1\...system - Diagram_0* LibraryManagement...U894SC\user (57)
- Query pane (left): Lines 179, 180, and 181 of the script.
- Results pane (right): A table with columns: MemberName, ItemTitle, Author, ItemBorrowedDate, ItemReturnedDate, OverdueRate, Fine. The table is currently empty.

QUESTION 4



```
182 ---QUESTION 4
183 --- Create a trigger so that the current status of an item automatically updates to Available when
184 CREATE TRIGGER trLoan_Returned
185 ON Loan
186 AFTER UPDATE
187 AS
188 BEGIN
189 IF UPDATE (ItemReturnedDate)
190 BEGIN
191 UPDATE Item
192 SET Status = 'Available'
193 FROM Item
194 INNER JOIN inserted ON Item.ItemId = inserted.ItemId
195 WHERE inserted.ItemReturnedDate IS NULL
196 END
197 END
```

The screenshot shows a SQL query window with the following details:
- Title bar: DESKTOP-U894SC1\...system - Diagram_0* LibraryManagement...U894SC\user (57)
- Query pane (left): Lines 182 through 197 of the script.
- Results pane (right): A message indicating "Commands completed successfully." and a completion time of "2023-04-28T15:52:09.1947242+01:00".

To automatically update an item to show that the status is available when returned, it is necessary to create a trigger. By doing so, the trigger that was created will fire after an update on the loan table to check if the itemreturneddate column has been updated. If it has, the trigger will then set the status table to available for the actual item that was returned using the update statement and join clause as shown above. In a situation, whereby the status has additional states, the trigger will need to be modified to the current states.

QUESTION 5

The screenshot shows a SQL query window in SQL Server Management Studio. The code is as follows:

```
199 --QUESTION 5
200 --- query that will retrieve the total number of loans made on a specific
201 SELECT COUNT(*) AS TotalLoans
202 FROM Loan
203 WHERE ItemBorrowedDate = '2023-04-12';
204
```

The results pane shows a single row with the column 'TotalLoans' containing the value '0'. The status bar at the bottom indicates the query was executed successfully.

To identify the total number of loans made on a particular date, applying the COUNT() function will return the number of rows that meet the conditions. This will give you the sum up of loans being made on a particular date. From the above code, the date used was 2023-04-12 which can be replaced depending on the date you want to view.

QUESTION 6

Insert Data into Member Table

```
226 INSERT INTO Member (Username, Password, FirstName, MiddleName, LastName, Address1, Address2, DateOfBirth, EmailAddress, ContactNumber, MembershipEndDate)
227 VALUES ('Jude_Oseme', 'Password111', 'Jude', 'Isioma', 'Oseme', '245 lloyd St', '245 lloyd St', 'Apt 10', '1992-03-01', 'johndonald@gmail.com', '133-428-6830', '2024-06-20')
228 ('Jackson_Paul', 'Password222', 'Jackson', 'Daniel', 'Paul', '112 Hulme St', '112 Hulme St', 'Suite 6', '1996-03-20', 'jackson.paul@gmail.com', '+234-8356', '2023-06-30')
229 ('Mary_John', 'Password123', 'Mary', 'Hannah', 'John', '225 Morrison st', '225 Morrison st', 'Apt 2', '2000-02-15', 'mary.john@gmail.com', '+44-6336', '2024-08-25')
230 ('Mart_Luth', 'Password348', 'Martins', 'James', 'Luther', '205 Bridge Close', '205 Bridge Close', 'Flat 1', '2000-07-09', 'Martins.Luther@gmail.com', '+888-2567', '2023-09-10')
231 ('thomas_jefferson', 'democracy2', 'Thomas', 'Bob', 'Jefferson', '234 Monticello Ave', '234 Monticello Ave', 'Apt 6', '1993-04-13', 'tjefferson@usa.gov', '+447-1776', '2025-09-10')
232 ('george_Kenneth', 'Manchester1', 'George', 'Karim', 'Kenneth', '1 London Street', '1 London Street', 'Suite 2', '1974-02-22', 'GeorgeKen@usa.gov', '555-1776', '2024-02-22')
233 ('Mercy_David', 'Password002', 'Mercy', 'Halim', 'David', 'John Lester Court', 'John Lester Court', 'Flat 10', '1994-01-01', 'Mercy.David@yahoo.com', '+234-369', '2026-03-07')
234 ('John_Stone', 'Chelsea004', 'John', 'Don', 'Stone', 'Eddie Colman Courts', 'Eddie Colman Courts', 'Apt 6', '1993-02-02', 'Johnstone@gmail.com', '+234-345', '2023-05-20')
235 ('Jim_Ikye', 'Password011', 'Jim', 'Clinton', 'Ikye', '39 Robertson Road', '39 Robertson Road', 'Suite 1', '1989-03-02', 'JimIkye@yahoo.com', '+447-062', '2023-07-15')
236 ('Robert_Clay', 'United05', 'Robert', 'Andrew', 'Clay', '85 Salford drive', '85 Salford drive', 'Flat 3', '1999-09-06', 'RobertClay@gmail.com', '+777-093', '2023-06-07')
237
238 SELECT * From Member
```

Results Messages

MemberId	Username	Password	FirstName	MiddleName	LastName	Address1	Address2	DateOfBirth	EmailAddress	ContactNumber	MembershipEndDate
1	jdoe	mypassword	John	F	Donald	145 Lloyd St		1992-03-01	johndonald@gmail.com	133-428-6830	2024-06-20
2	Jude_Oseme	Password111	Jude	Isioma	Oseme	245 lloyd St		1990-05-05	jude.oseme@gmail.com	+234-8356	2023-06-30
3	Jackson_Paul	Password222	Jackson	Daniel	Paul	112 Hulme St		1996-03-20	jackson.paul@gmail.com	+44-6336	2024-08-25
4	Mary_John	Password123	Mary	Hannah	John	225 Morrison st		2000-02-15	mary.john@gmail.com	+888-2567	2023-09-10
5	Mart_Luth	Password348	Martins	James	Luther	205 Bridge Close		2000-07-09	Martins.Luther@gmail.com	+809-445	2023-04-09
6	thomas_jefferson	democracy2	Thomas	Bob	Jefferson	234 Monticello Ave		1993-04-13	tjefferson@usa.gov	+447-1776	2025-09-10
7	george_Kenneth	Manchester1	George	Karim	Kenneth	1 London Street		1974-02-22	GeorgeKen@usa.gov	555-1776	2024-02-22
8	Mercy_David	Password002	Mercy	Halim	David	John Lester Court		1994-01-01	Mercy.David@yahoo.com	+234-369	2026-03-07
9	John_Stone	Chelsea004	John	Don	Stone	Eddie Colman Courts		1993-02-02	Johnstone@gmail.com	+234-345	2023-05-20
10	Jim_Ikye	Password011	Jim	Clinton	Ikye	39 Robertson Road		1989-03-02	JimIkye@yahoo.com	+447-062	2023-07-15
11	Robert_Clay	United05	Robert	Andrew	Clay	85 Salford drive		1999-09-06	RobertClay@gmail.com	+777-093	2023-06-07

The above screenshot of code is an SQL query that inserts data into the "Member" table of a database. The "INSERT INTO" statement is used to add a new row of data to a table, while the "VALUES" keyword indicates the values to be inserted. In this context, 10 different members with information ranging from username, password, name, address, date of birth, email address, contact number, and membership end date is being inserted into the "Member" table. Each of the value entered is separated with commas and enclosed with a parenthesis. The SELECT statement will extract all the data from the table for viewing or for further analysis.

Insert Data into Item Table

```
240 --- Insert data into the Item table
241 INSERT INTO Item (ItemId, ItemTitle, ItemType, Author, YearOfPublication, Status, ISBN, ItemAddedDate)
242 VALUES
243     (1, 'Last Dance', 'Book', 'Mark Smith', '1975-04-10', 'Available', '3252259870165', '2022-01-10')
244     (2, 'The Great Mystery', 'Book', 'H. Arnold Lugard', '1966-10-11', 'Lost', '6348235798835', '2023-03-09')
245     (3, 'Music for the Soul', 'Book', 'Stella Maris', '1995-04-28', 'Available', '9235443278564', '2023-02-13')
246     (4, 'Africa Colonization', 'Book', 'Jim Peter', '1982-07-09', 'Overdue', '7348862155432', '2023-03-05')
247     (5, 'Digital Industrialization', 'Journal', 'Jones David', '1999-09-07', 'OnLoan', '6231773856249', '2023-01-05')
248     (6, 'Love Birds', 'DVD', 'Marie Sally', '2006-07-12', 'Lost', '6774552371689', '2022-08-11')
249     (7, 'The Physics fundamentals', 'Journal', 'Ramsey Mark', '1983-09-03', 'Lost', '2318799068345', '2023-02-13')
250     (8, 'Data world', 'DVD', 'Luggard Smith', '1970-10-11', 'Available', '2239067354332', '2023-03-05')
251     (9, 'Engineering Mathematics', 'Book', 'Richard Kent', '1999-05-09', 'On Loan', '6330933380567', '2023-01-05')
252     (10, 'The Secrets of Success', 'DVD', 'Alex Ramos', '2005-06-18', 'Removed', '4632891425648', '2023-02-05')
253
254 SELECT * From Item
```

The above SQL command is used to insert data into the "Item" table of a library management system. It shows the different items in the library such as DVD, books, journal. In this case, 10 rows of data were inserted consisting of information such as the item Id, title type, author, year of publication and its current status in the library. The "ItemAddedDate" column is also included, which represents the date on which the item was added to the library's collection. When executed, it will display the items along with the respective information in different columns as shown above.

Insert Data into Loan Table

The screenshot shows a SQL query window in SSMS. The query inserts data into the 'Loan' table with columns: LoanId, MemberId, ItemId, ItemBorrowedDate, ItemDueDate, ItemReturnedDate, and OverdueRate. The data consists of nine rows. A SELECT statement is also present to verify the data.

```
255 --- Insert data into the Loan table
256 INSERT INTO Loan (LoanId, MemberId, ItemId, ItemBorrowedDate, ItemDueDate, ItemReturnedDate, OverdueRate)
257 VALUES
258 (1, 3, 2, '2023-03-01', '2023-03-03', '2023-04-28', 0.10),
259 (3, 4, 3, '2023-02-10', '2023-02-15', '2023-02-27', 0.10),
260 (2, 5, 4, '2023-04-05', '2023-04-10', '2023-04-08', 0.00),
261 (5, 2, 1, '2023-04-10', '2023-05-10', '2023-05-01', 0.00),
262 (4, 3, 6, '2023-04-20', '2023-05-20', '2023-05-26', 0.10),
263 (6, 3, 1, '2023-05-10', '2023-05-25', '2023-05-23', 0.00),
264 (7, 5, 8, '2023-04-18', '2023-04-25', '2023-04-23', 0.00),
265 (8, 6, 2, '2023-03-20', '2023-04-12', '2023-04-20', 0.10),
266 (9, 1, 4, '2023-04-01', '2023-04-05', '2023-04-03', 0.00);
267
268
269 SELECT * From Loan
270
```

Results grid:

LoanId	MemberId	ItemId	ItemBorrowedDate	ItemDueDate	ItemReturnedDate	OverdueRate
1	3	2	2023-03-01	2023-03-03	2023-04-28	0.10
2	5	4	2023-04-05	2023-04-10	2023-04-08	0.00
3	4	3	2023-02-10	2023-02-15	2023-02-27	0.10
4	3	6	2023-04-20	2023-05-20	2023-05-26	0.10
5	2	1	2023-04-10	2023-05-10	2023-05-01	0.00
6	3	1	2023-05-10	2023-05-25	2023-05-23	0.00
7	5	8	2023-04-18	2023-04-25	2023-04-23	0.00
8	6	2	2023-03-20	2023-04-12	2023-04-20	0.10
9	1	4	2023-04-01	2023-04-05	2023-04-03	0.00

Message bar: Query executed successfully.

The SQL query above will insert data into the loan table. The loan table contains information about the loan being made by members. As shown above, it contains all the necessary information that is peculiar to a particular member such as the member ID, item ID, item borrowed date, item due date, item returned date, and overdue rate. SELECT statement retrieves all the data from the table for viewing or further analysis.

Insert Data into Overdue Table

DESKTOP-U894SC\...ystem - Diagram_0* LibraryManagement..U894SC\user (57) ×

```
271 --- Insert data into the Overdue table
272 INSERT INTO Overdue (OverdueId, MemberId, ItemId, OverdueDateTime, OverdueTotalAmount)
273 VALUES
274     (1, 2, 1, '2023-04-16 10:00:22', 1.50),
275     (3, 1, 2, '2023-04-15 09:05:02', 1.20),
276     (6, 4, 2, '2023-04-20 13:15:06', 2.00),
277     (4, 2, 3, '2023-05-22 15:00:10', 3.00),
278     (7, 4, 3, '2023-01-13 08:30:52', 1.00),
279     (9, 2, 7, '2023-02-17 10:27:17', 2.50),
280     (2, 8, 4, '2023-03-09 11:30:43', 4.50),
281     (5, 3, 6, '2023-02-03 09:30:37', 6.20),
282     (8, 5, 4, '2023-02-22 12:05:15', 3.00);
283
284 SELECT * From Overdue
285
```

80 %

Results Messages

	OverdueId	MemberId	ItemId	OverdueDateTime	OverdueTotalAmount
1	1	2	1	2023-04-16 10:00:22.000	1.50
2	2	8	4	2023-03-09 11:30:43.000	4.50
3	3	1	2	2023-04-15 09:05:02.000	1.20
4	4	2	3	2023-05-22 15:00:10.000	3.00
5	5	3	6	2023-02-03 09:30:37.000	6.20
6	6	4	2	2023-04-20 13:15:06.000	2.00
7	7	4	3	2023-01-13 08:30:52.000	1.00
8	8	5	4	2023-02-22 12:05:15.000	3.00
9	9	2	7	2023-02-17 10:27:17.000	2.50

The SQL query above inserts data into the overdue table by storing information's about the overdue items borrowed by the members. The query inserts 9 rows of data into the table, each representing a separate instance of an overdue item. The Select statement will display the resulting details of overdue date of each member.

Insert Data into Repayment Table

The screenshot shows a SQL query window in SSMS. The query is:

```
285  
286  
287 INSERT INTO Repayment (RepaymentId, OverdueId, RepaymentDateTime, RepaymentAmount, PaymentMethod)  
288 VALUES  
289 (1, 3, '2023-03-01 10:15:00', 3.00, 'Credit Card'),  
290 (3, 5, '2023-07-01 11:30:00', 1.00, 'Debit Card'),  
291 (5, 2, '2023-05-15 14:00:00', 5.00, 'Cash'),  
292 (4, 3, '2023-06-05 09:45:00', 8.75, 'credit card'),  
293 (2, 4, '2023-04-23 08:20:33', 4.25, 'Cash'),  
294 (6, 1, '2023-02-12 13:25:42', 2.00, 'Cash'),  
295 (7, 5, '2023-03-07 12:30:20', 3.50, 'Debit Card'),  
296 (8, 7, '2023-04-15 13:05:15', 2.50, 'Cash'),  
297 (9, 6, '2023-06-12 15:10:45', 3.00, 'Cash');  
298  
299 select * From Repayment
```

The results pane shows a table with 9 rows of data:

	RepaymentId	OverdueId	RepaymentDateTime	RepaymentAmount	PaymentMethod
1	1	3	2023-03-01 10:15:00.000	3.00	Credit Card
2	2	4	2023-04-23 08:20:33.000	4.25	Cash
3	3	5	2023-07-01 11:30:00.000	1.00	Debit Card
4	4	3	2023-06-05 09:45:00.000	8.75	credit card
5	5	2	2023-05-15 14:00:00.000	5.00	Cash
6	6	1	2023-02-12 13:25:42.000	2.00	Cash
7	7	5	2023-03-07 12:30:20.000	3.50	Debit Card
8	8	7	2023-04-15 13:05:15.000	2.50	Cash
9	9	6	2023-06-12 15:10:45.000	3.00	Cash

The Repayment table is being updated by this SQL statement. The repayments that members made for their past-due items are detailed in the table. Nine rows are inserted into the table by the statement, each row denoting a single repayment made by a member. The data being entered are the repayment ID, the overdue ID for which the repayment is made, the repayment date and time, the repayment amount, and the payment method.

DATA INTEGRITY AND CONCURRENCY

Data integrity and concurrency is an integral part that must be considered in any database system. This is because Data integrity always ensures accuracy, consistency as well as the validity of the different data stored. Concurrency simply means that different users can have access to the data and modify it if they wish to without any conflicts or difficulties. Data integrity can be achieved by applying constraints such as NOT NULL, FOREIGN KEY. By so doing, these constraints will help to prevent data invalidity from being entered into the database, thereby ensuring the accuracy and consistency of the data. Also implementing a data backup and recovery plan will always ensure that data can be recovered in the event of hardware failure or data corruption which could be caused by any virus. There should be a regular backup stored in a secure location to ensure data availability and prevent data loss. To ensure concurrency, implement locking mechanisms will help to prevent multiple users from modifying the same data at the same time depending on what the system requires, using optimistic or pessimistic locking mechanisms will help to ensure that. Multiple users may read the data simultaneously with optimistic

locking, but only one user may edit the data at once. The data is locked using pessimistic locking as soon as it is accessed, preventing any further access or modification by other users until the lock is released. Implementing transaction management is crucial for preserving the integrity and consistency of data. Transactions group multiple database operations into a single unit of work, ensuring that either all the operations are completed successfully or none of them are. This helps to prevent data inconsistencies and ensures that the database always remains in a consistent state.

DATABASE SECURITY

The following are some advice and guidance on database security that the client should consider:

1. Authentication: The client should ensure to use strong authentication mechanisms to prevent unauthorized access to the database. The use of strong passwords, two-factor authentication, and biometric authentication is highly considered as it will help to enhance database security.
2. Authorization: Implementing authorization policies that limit access to the database to authorized users only. Authorization will ensure that users are assigned specific role with predefined access rights.
3. Encryption: Data are sensitive material and as such, it should be encrypted to prevent unauthorized access or data theft. The client can use techniques such as database encryption, SSL/TLS encryption etc to safeguard the data.
4. Regular Backups: There should be a constant regular backup as these are essential in case of data loss or corruption. The client should implement a backup strategy that includes regular backups of the database as well as the transaction logs.
5. Regular Updates and Patches: The client should regularly update the database management system and apply security patches to prevent vulnerabilities that could be exploited by attackers.
6. Employee Training: Employee training is critical in enhancing database security. The client should provide regular training to employees on database security best practices, including password hygiene, social engineering attacks, and data protection policies.

DATABASE BACKUP

There are scenarios where system failure is likely to occur. Applying Database backup is essential to ensure data are well preserved and available in case of system failures, data corruption, or accidental deletion. To ensure that the library's data is always protected, regular backups should be scheduled, and the backups should be stored in a secure location to prevent data loss due to unforeseen circumstances such as fire or flood.

There are three main types of backups:

1. Full Backup: This is a well detailed and a comprehensive type of backup. It provides a complete snapshot of the database.
2. Differential Backup: This is the fastest type of backup as it takes a very minimal space compared to the full backup. It involves backing up only the data that has changed since the last full backup.
3. Transaction Log Backup: A transaction log backup involves backing up the log of all changes made to the database since the last backup. This type of backup allows for point-in-time recovery and can be used to recover to a specific point in time.

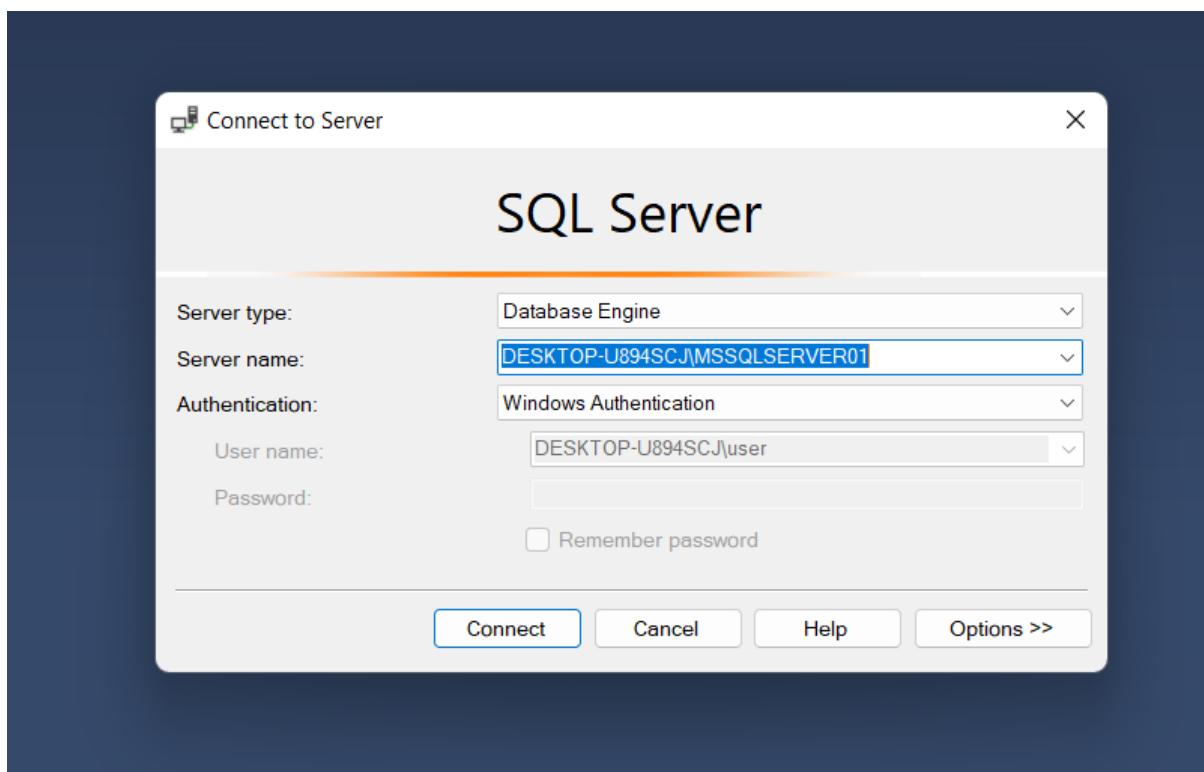
In addition to regular backups, ensure that the restore process are tested to check if it's working properly in case of any disaster. Also, ensure that the backups are being stored in a secure location like a cloud-based storage solution.

ADVANCE DATABASE

TASK 2

INTRODUCTION

As a database consultant working for a pharmaceutical company, I am being tasked with the responsibility to analyse a prescribing data in order to understand the different varieties of medications being prescribed, the institution prescribing the medications, as well as the quantities prescribed. The major reason why most pharmaceutical companies depend on prescribing data is to gain clarity on the market environment, consumer demands as well as client's requirements. A thorough analysis of this data can provide useful information to the companies that will enable them to make a well data driven decision which will have a positive impact in their business strategies. Exploring this data will certainly be a useful tool to companies to enable them identify trends, potential customers and sales strategy effectively. Analysing the data can help in the area of quantities prescribed which can help pharmaceutical companies identify patterns in the use of their drugs. This information will help to ensure that drugs are not being abused but rather used appropriately for safety concern. To tackle this task, connect to the Microsoft SQL server, where the server's name of my device will be connected.



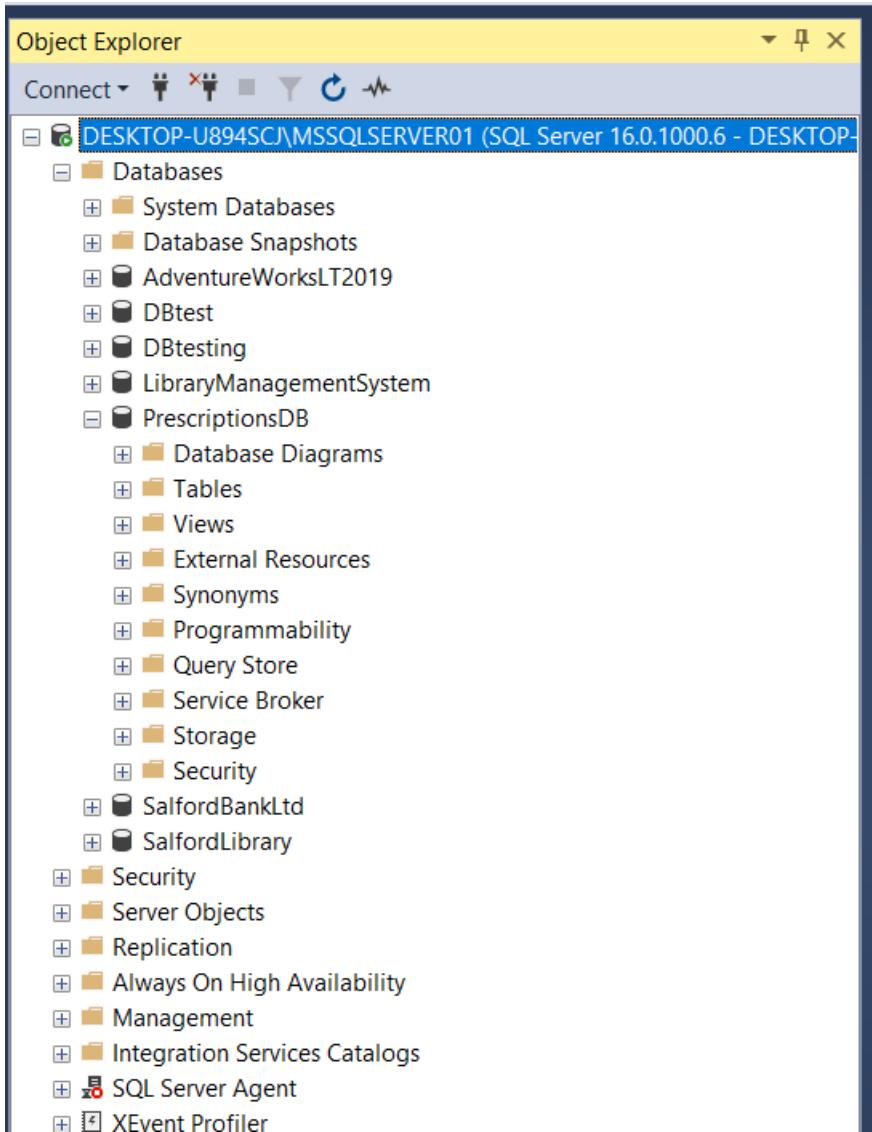
Proceed to create the database for this task. Create the database using the CREATE DATABASE statement. The database created shall be called “PrescriptionsDB”.

The screenshot shows the SSMS interface with the Object Explorer on the left and a query window on the right. The query window contains the following T-SQL code:

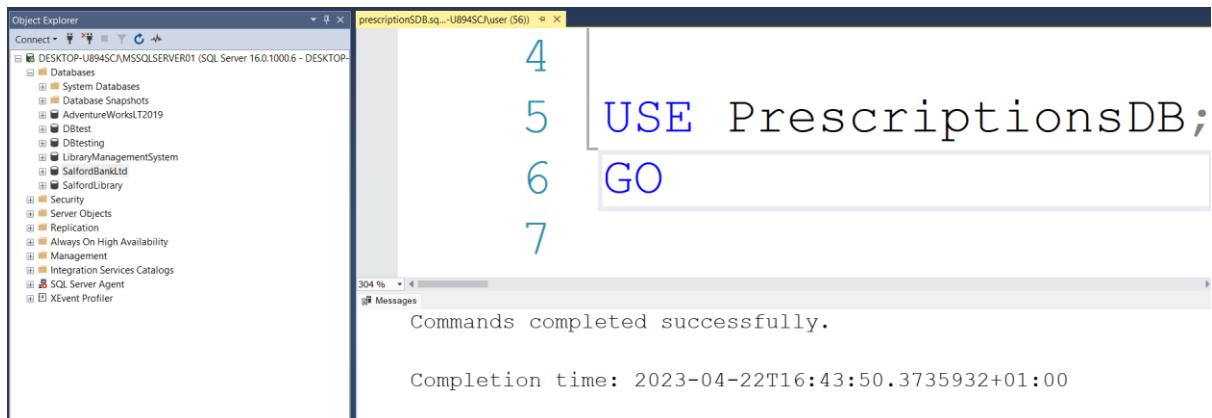
```
1 --- Create the database
2
3 CREATE DATABASE Prescriptions
4
5 USE PrescriptionsDB;
6 GO
```

Below the code, a message indicates "Commands completed successfully." and shows the completion time as "Completion time: 2023-04-22T16:40:02.5997552+01:00".

Check to see if it has been successfully created by refreshing the object explorer and clicking the plus sign next to the databases to expand. The below screenshot shows that it has been successfully created as the database name “PrescriptionsDB” appears.



Use the below command for the newly created PrescriptionsDB database.

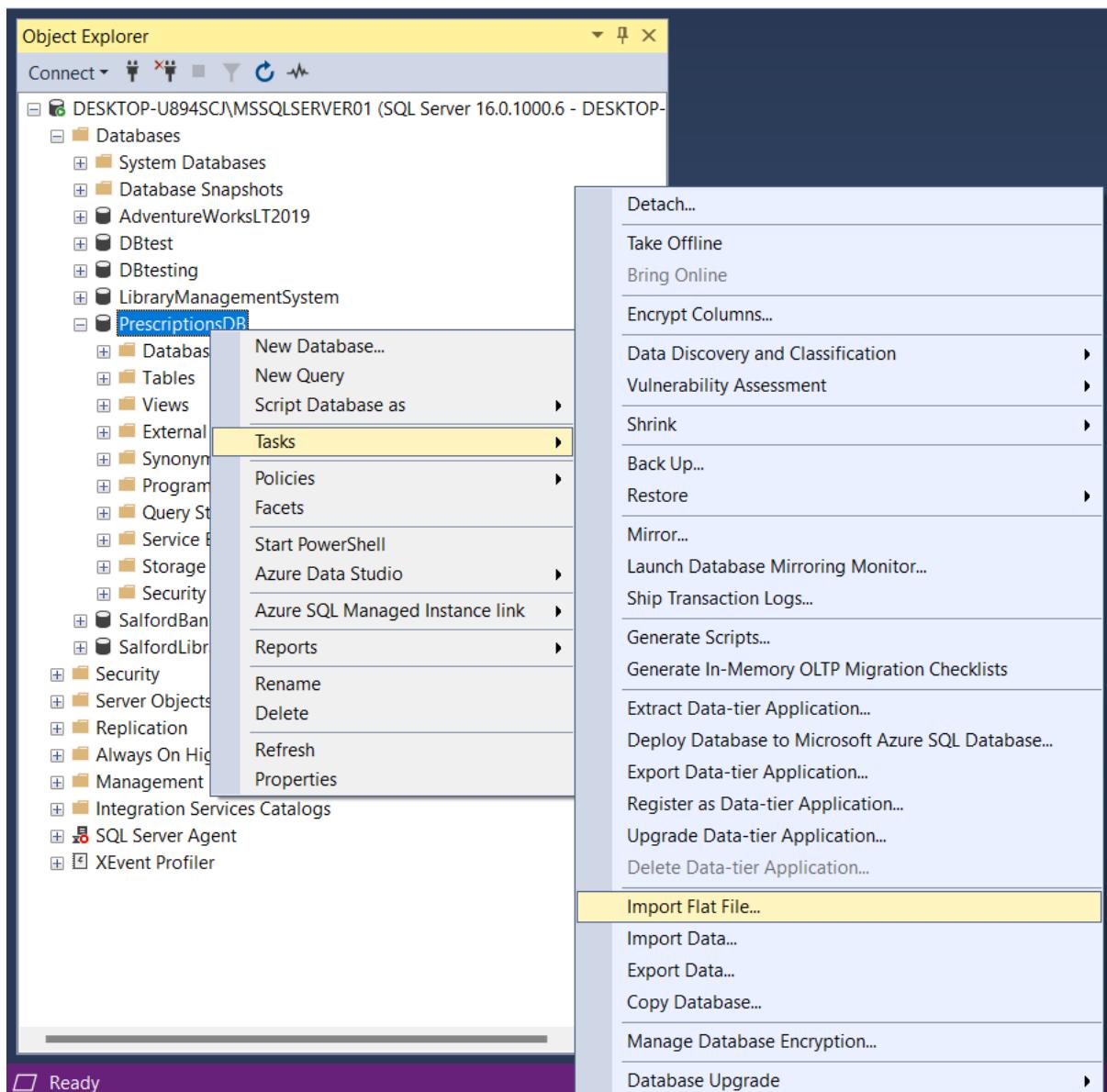


The screenshot shows the SSMS interface. On the left, the Object Explorer displays a tree view of the database structure under 'DESKTOP-U894SC\MSQLSERVER01'. A 'prescriptionDB.sq... (56)' connection node is selected. On the right, a query window titled 'prescriptionDB.sq... (56)' contains the following T-SQL code:

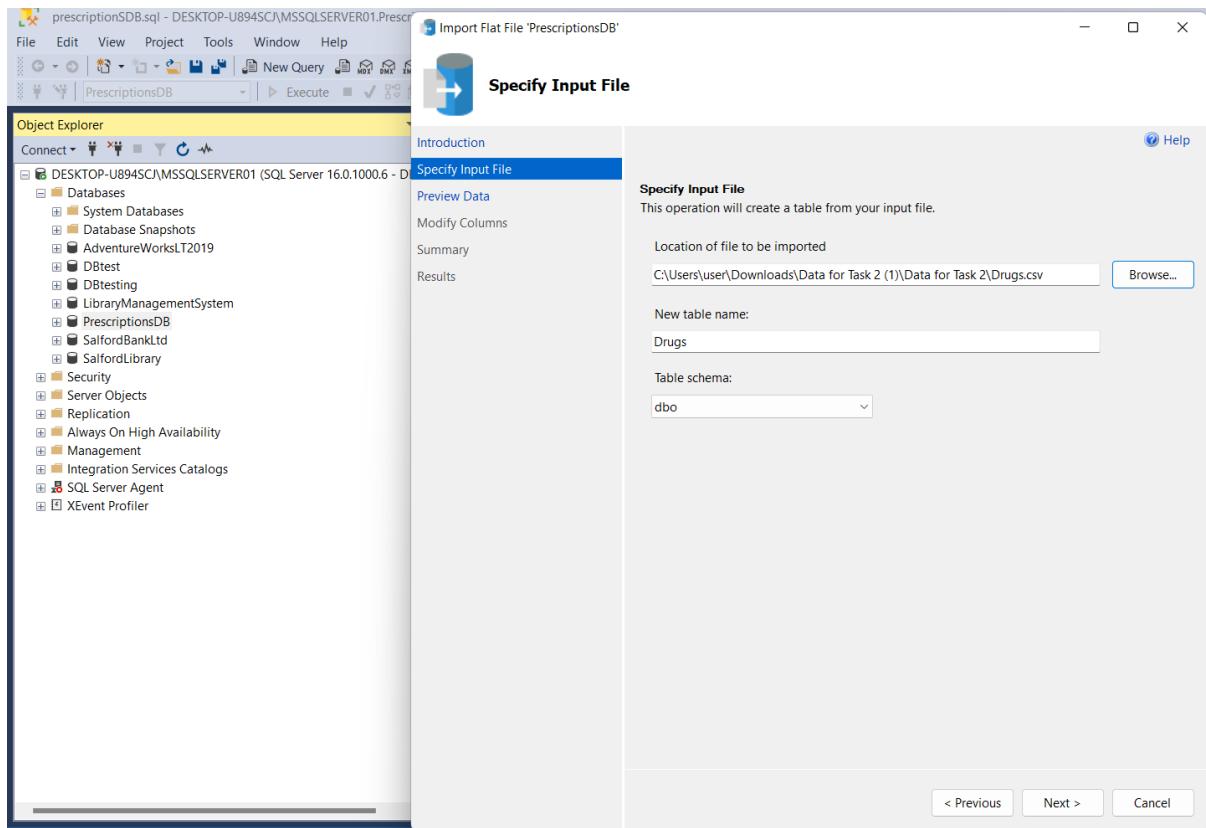
```
4 USE PrescriptionsDB;
5 GO
6
7
```

Below the code, the status bar shows 'Commands completed successfully.' and the completion time: 'Completion time: 2023-04-22T16:43:50.3735932+01:00'.

Import the three data from the csv file provided table. To do this, right click on the newly database created, navigate to the task and select import flat file. Click on browse and navigate to where the csv file was saved.



After clicking on the import flat file, in the window that opens click on Browse and then navigate to where you have saved the csv file downloaded from Blackboard. The three files that shall be imported are drugs, medical_practice, and prescriptions. The below screenshot shows that the drug table was uploaded.



Once you have selected the file, click next to preview the file.

Import Flat File 'PrescriptionsDB'

Preview Data

Introduction
Specify Input File
Preview Data
Modify Columns
Summary
Results

Help

Preview Data

This operation analyzed the input file structure to generate the preview below for up to the first 50 rows.

BNF_CODE	CHEMICAL_SL	BNF_DESCRIP	BNF_CHAPTE
0205051L0A...	Lisi	CHEMICAL_SUBSTANCE_BNF_DESCR ...	
0205051L0A...	Lisinopril	Lisinopril 10...	02: Cardio...
0205051L0A...	Lisinopril	Lisinopril 20...	02: Cardio...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardio...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardio...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardio...
0205051R0...	Ramipril	Ramipril 1.2...	02: Cardio...
0205051R0...	Ramipril	Ramipril 2.5...	02: Cardio...
0205051R0...	Ramipril	Ramipril 5m...	02: Cardio...
0205051R0...	Ramipril	Ramipril 10...	02: Cardio...
0205051R0...	Ramipril	Ramipril 1.2...	02: Cardio...
0205051R0...	Ramipril	Ramipril 2.5...	02: Cardio...
0205051R0...	Ramipril	Ramipril 10...	02: Cardio...
0205052AE...	Sacubitril/v...	Sacubitril 2...	02: Cardio...
0205052B0...	Olmesartan ...	Olmesartan ...	02: Cardio...
0205052C0...	Candesarta...	Candesarta...	02: Cardio...
0205052C0...	Candesarta...	Candesarta...	02: Cardio...
0205052C0...	Candesarta...	Candesarta...	02: Cardio...
0205052C0...	Candesarta...	Candesarta...	02: Cardio...
0205052C0...	Candesarta...	Candesarta...	02: Cardio...
0205052I0A...	Irbesartan	Irbesartan 7...	02: Cardio...

Use Rich Data Type Detection - may provide a closer type fit. However, cells with anomalous values may be dropped.

< Previous Next > Cancel

Leave 'Use Rich Data Type Detection' ticked and click Next again. This will take you to where you will see the Column name, data type, primary key, and allow nulls.

Import Flat File 'PrescriptionsDB'

Modify Columns

Introduction Help

Specify Input File

Preview Data

Modify Columns

Summary

Results

Modify Columns

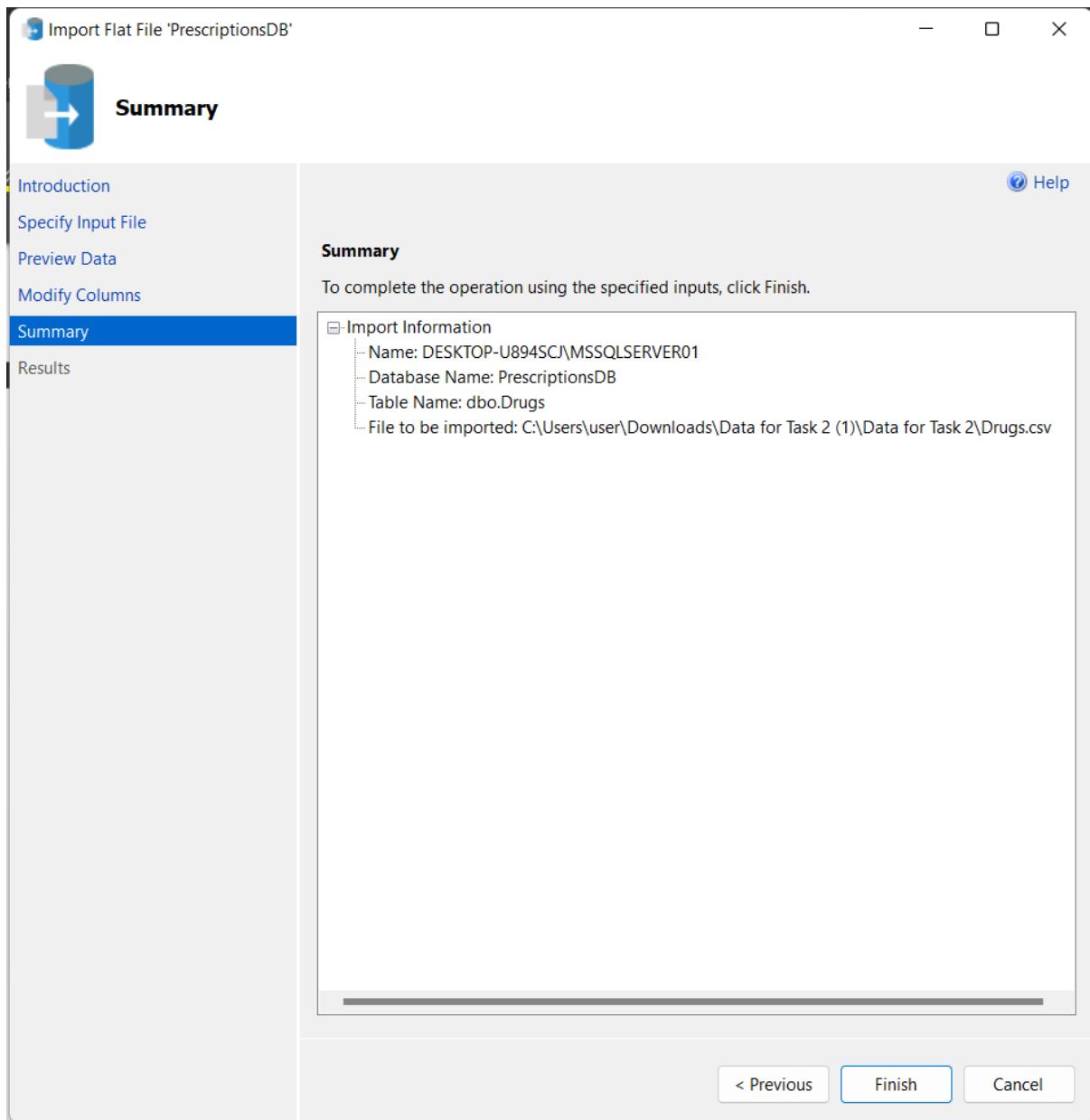
This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

Column Name	Data Type	Primary Key	<input type="checkbox"/> Allow Nulls
BNF_CODE	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
CHEMICAL_SUBSTANCE_BNF_DESCR	nvarchar(100)	<input type="checkbox"/>	<input type="checkbox"/>
BNF_DESCRIPTION	nvarchar(100)	<input type="checkbox"/>	<input type="checkbox"/>
BNF_CHAPTER_PLUS_CODE	nvarchar(100)	<input type="checkbox"/>	<input type="checkbox"/>

Row granularity of error reporting (performance impact with smaller ranges)

< Previous Next > Cancel

For the drug table, the following changes should be made for the data type. The columns are: BNF_CODE should have a data type of Nvarchar (50) while CHEMICAL_SUBSTANCE_BNF_DESCR, BNF_DESCRIPTION, BNF CHAPTER_PLUS_CODE should have a data type of Nvarchar (100)



The screenshot shows a software interface for importing flat files. The title bar reads "Import Flat File 'PrescriptionsDB'". On the left, a sidebar lists steps: Introduction, Specify Input File, Preview Data, Modify Columns, Summary, and Results. The "Results" step is selected, highlighted with a blue background. The main area displays a summary titled "Operation Complete" with a green checkmark icon. Below it is a table titled "Summary:" with two rows:

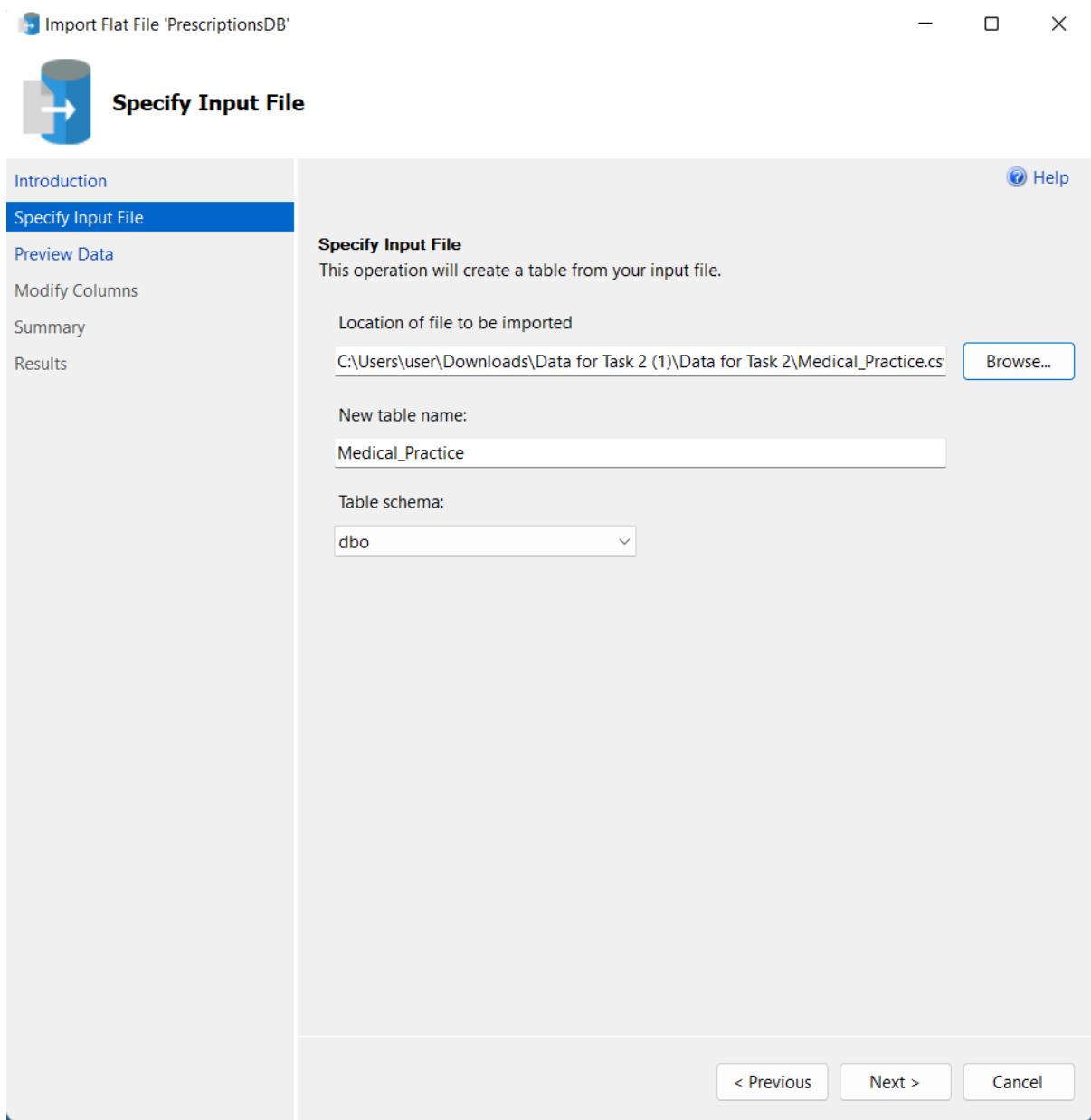
Name	Result
Insert Data	Success

At the bottom of the main area are navigation buttons: "< Previous", "Next >", and "Close".

The above table shows that the data for drugs table has been uploaded successfully.

For medical_practice

right click on the newly database created, navigate to the task and select import flat file. Click on browse and navigate to where the csv file was saved.



Once you have selected the file, click next to preview the file.

Import Flat File 'PrescriptionsDB'

Preview Data

[Introduction](#) [Help](#)

[Specify Input File](#)

Preview Data

[Modify Columns](#)

[Summary](#)

[Results](#)

This operation analyzed the input file structure to generate the preview below for up to the first 50 rows.

PRACTICE_CC	PRACTICE_NA	ADDRESS_1	ADDRESS_2	ADDRESS_3	ADDRESS_4	POS
P82034	EDGWORTH...	EGERTON/D...	DARWEN R...	BROMLEY C...	BOLTON	BL7 9
P82652	FARNWORT...	FARNWORT...	FREDRICK S...	BOLTON		BL4 9
P82037	FIG TREE M...	FARNWORT...	FREDERICK ...	FARNWORTH	BOLTON	BL4 9
P82633	GREAT LEVE...	GREAT LEVE...	RUPERT STR...	BOLTON	LANCASHIRE	BL3 6
P82022	HALLIWELL ...	THE HALLI...	LINDFIELD ...	HALLIWELL	BOLTON	BL1 1
P82029	HALLIWELL ...	THE HALLI...	LINDFIELD ...	BOLTON	LANCASHIRE	BL1 1
P82626	HALLIWELL ...	THE HALLI...	LINDFIELD ...	BOLTON	LANCASHIRE	BL1 1
P82016	HARWOOD ...	HARWOOD ...	HOUGH FO...	BOLTON	LANCASHIRE	BL2 6
P82031	HEATON M...	HEATON M...	2 LUCY STR...	HEATON, B...		BL1 1
P82030	DEANE ME...	155-157 DE...		BOLTON		BL3 9
P82007	KEARSLEY ...	KEARSLEY ...	JACKSON S...	BOLTON	LANCASHIRE	BL4 8
Y00747	INTERGRAT...	BREIGHTME...	BREIGHTME...	BOLTON	LANCASHIRE	BL2 6
P82661	INTERMEDI...	DARLEY CO...	SHEPHERD ...	BOLTON		BL1 1
P82004	SWAN LAN...	SWAN LAN...	SWAN LANE	BOLTON	LANCASHIRE	BL3 6
P82018	THE ALAST...	BREIGHTME...	BREIGHTME...	BRIGHTMET...	LANCASHIRE	BL2 6
P82001	THE DUNST...	BREIGHTME...	BREIGHTME...	BRIGHTMET...	LANCASHIRE	BL2 6
P82021	THE OAKS F...	CROMPTON...	501 CROMP...	BOLTON		BL1 8
P82643	BROMLEY ...	EGERTON/D...	DARWEN R...	BROMLEY C...	BOLTON	BL7 9
P82002	PIKES LANE 1	THE PIKES L...	DEANE RO...	BOLTON	LANCASHIRE	BL3 9
Y02790	BOLTON ME...	21 RUPERT ...	GREAT LEVER	BOLTON	LANCASHIRE	BL3 6

Use Rich Data Type Detection - may provide a closer type fit. However, cells with anomalous values may be dropped.

[< Previous](#) [Next >](#) [Cancel](#)

Leave 'Use Rich Data Type Detection' ticked and click Next again. This will take you to where you will see the Column name, data type, primary key, and allow nulls.

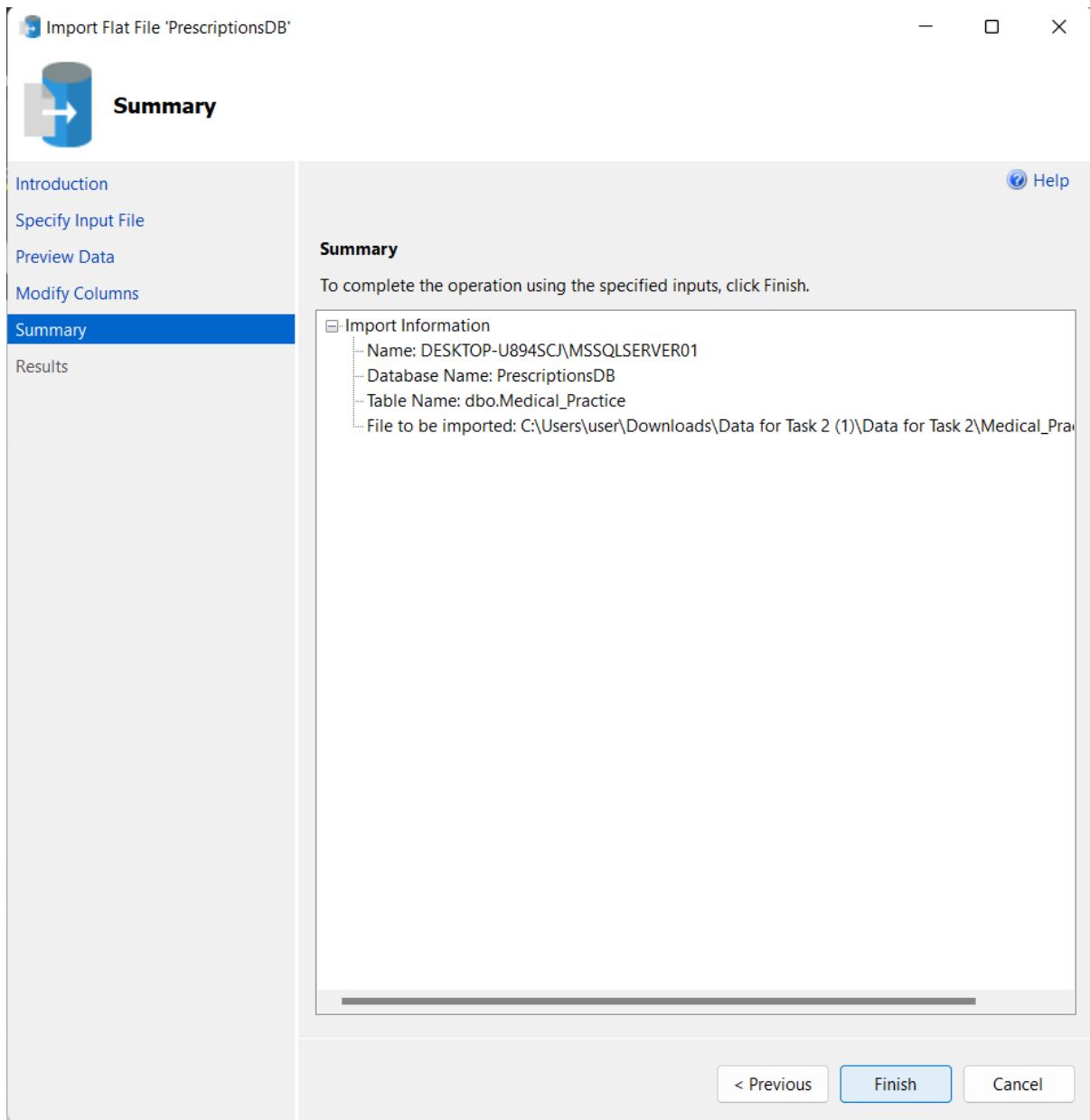
The screenshot shows the 'Modify Columns' step of an import wizard. The left sidebar has tabs for 'Introduction', 'Specify Input File', 'Preview Data', and 'Modify Columns', with 'Modify Columns' selected. The main area title is 'Modify Columns' with a sub-instruction: 'This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.' Below this is a table with columns: Column Name, Data Type, Primary Key, and Allow Nulls. The table rows are:

Column Name	Data Type	Primary Key	Allow Nulls
PRACTICE_CODE	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
PRACTICE_NAME	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
ADDRESS_1	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
ADDRESS_2	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ADDRESS_3	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ADDRESS_4	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
POSTCODE	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>

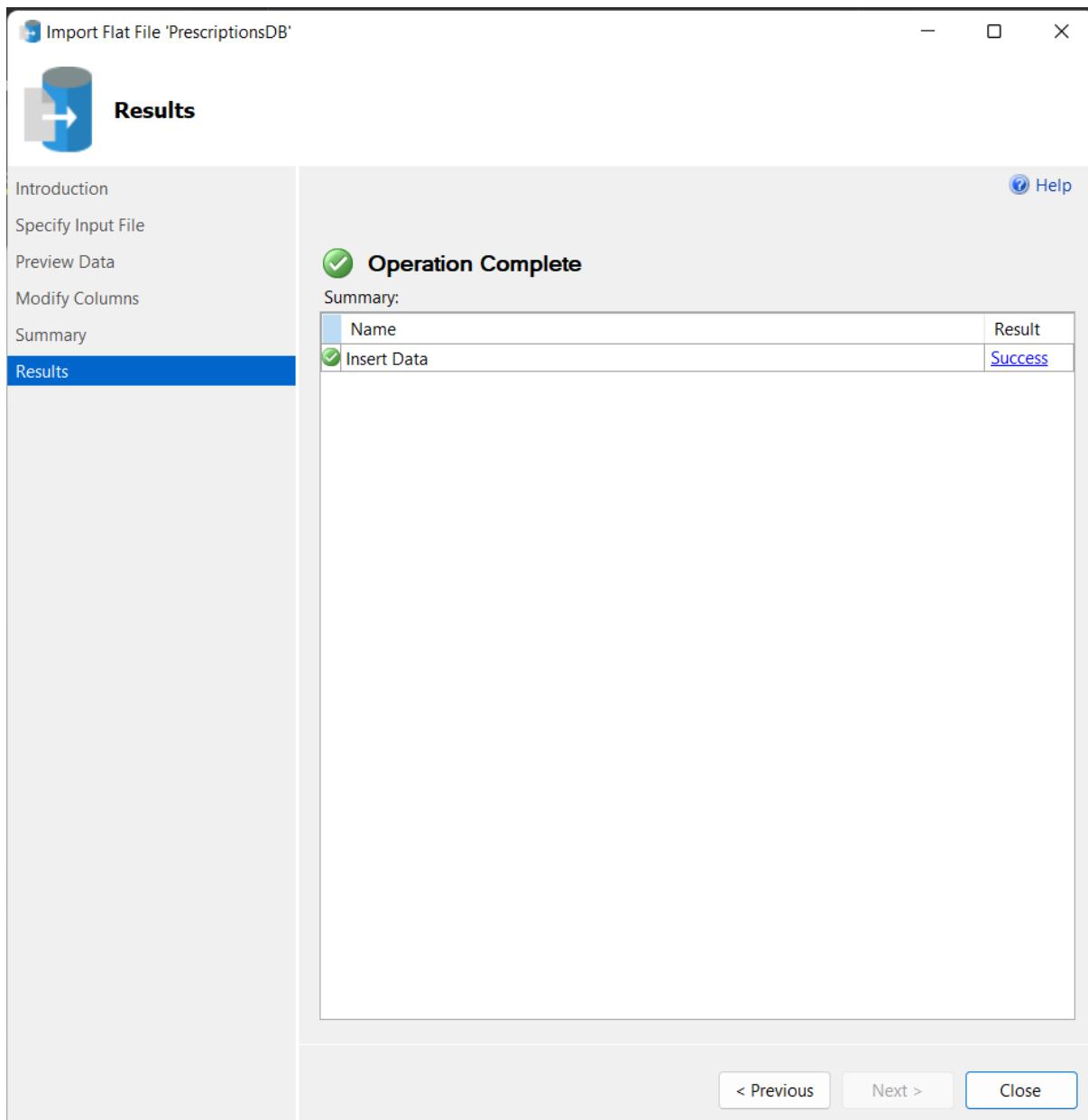
Below the table is a dropdown for 'Row granularity of error reporting (performance impact with smaller ranges)' set to 'No Range'. At the bottom are navigation buttons: '< Previous', 'Next >', and 'Cancel'.

From the above screenshots, all the columns have the same data type which is Nvarchar (50). They are listed below.

practice_code, practice_name, address_1, address_2, address_3, address_4, postcode. Click on allow nulls for address_2, address_3, address_4, postcode

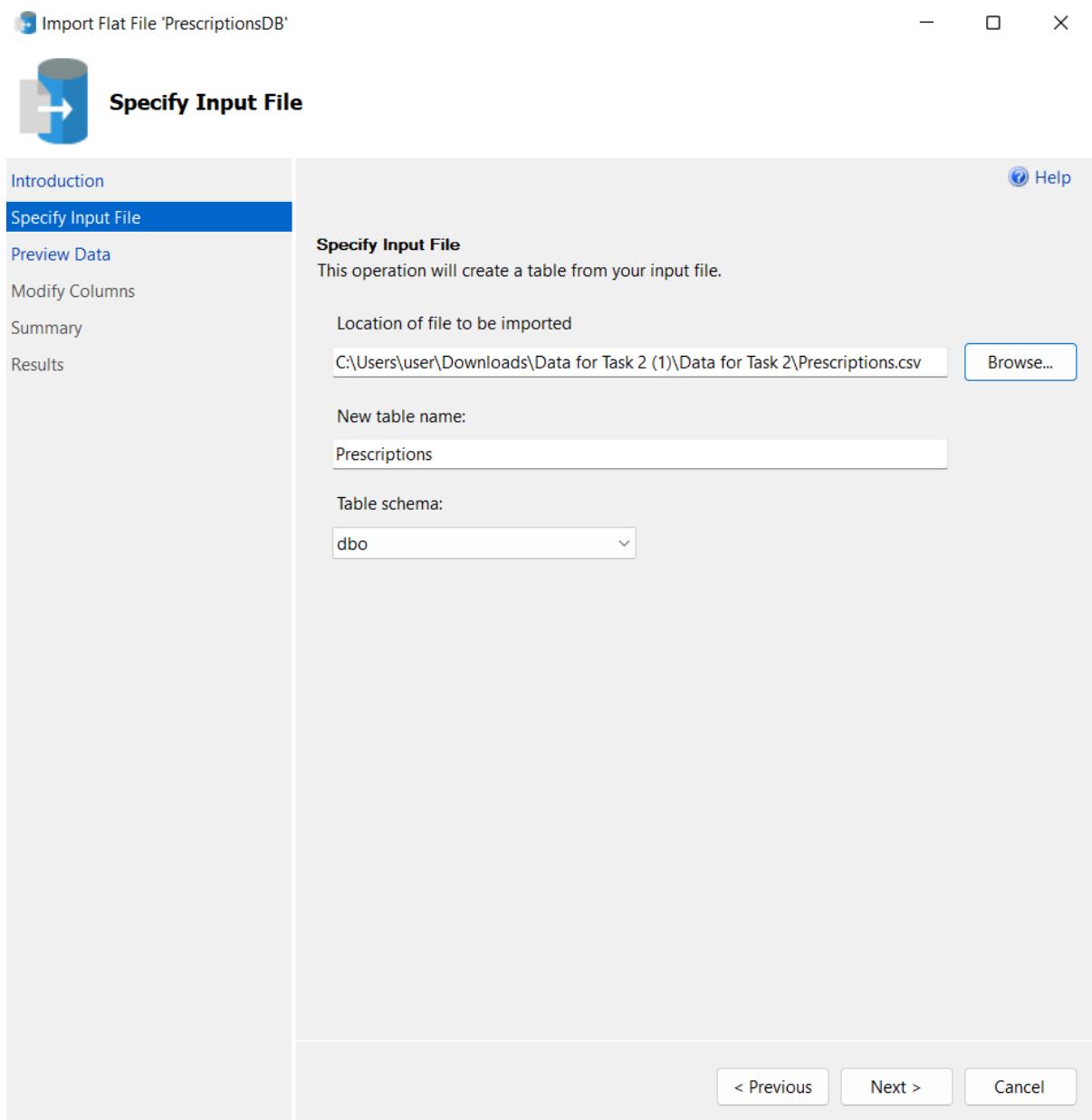


After that click on Next to indicates that the file has been uploaded successfully as shown below.



For prescription

Similarly, the same process should be applied for prescription table. Right click on the newly database created, navigate to the task and select import flat file. Click on browse and navigate to where the csv file was saved.



Click next to preview the file.

Import Flat File 'PrescriptionsDB'

Preview Data

Introduction Help

Specify Input File

Preview Data

Modify Columns

Summary

Results

This operation analyzed the input file structure to generate the preview below for up to the first 50 rows.

PRESRIPTION	PRACTICE_CO	BNF_CODE	QUANTITY	ITEMS	ACTUAL_COST
0	P82034	0205051L0A...	28	11	8.56974
1	P82034	0205051L0A...	56	4	6.18294
2	P82034	0205051L0A...	28	12	10.47075
3	P82034	0205051L0A...	84	1	2.59289
4	P82034	0205051L0A...	56	10	17.32726
5	P82034	0205051L0A...	84	2	5.52236
6	P82034	0205051L0A...	112	4	14.70977
7	P82034	0205051L0A...	56	12	22.13905
8	P82034	0205051L0A...	28	9	8.35794
9	P82034	0205051M0...	30	1	0.91931
10	P82034	0205051M0...	28	1	0.96321
11	P82034	0205051M0...	56	1	1.80468
12	P82034	0205051M0...	30	3	3.17866
13	P82034	0205051M0...	28	2	2.95488
14	P82034	0205051M0...	84	1	4.19817
15	P82034	0205051M0...	30	1	1.47094
16	P82034	0205051R0...	56	2	3.54025
17	P82034	0205051R0...	28	4	3.56505
18	P82034	0205051R0...	28	8	7.42928
19	P82034	0205051R0...	112	1	3.67744
20	P82034	0205051R0...	56	4	7.37968

Use Rich Data Type Detection - may provide a closer type fit. However, cells with anomalous values may be dropped.

< Previous Next > Cancel

Leave 'Use Rich Data Type Detection' ticked and click Next again. This will take you to where you will see the Column name, data type, primary key, and allow nulls.

Import Flat File 'PrescriptionsDB'

Modify Columns

Introduction Specify Input File Preview Data Modify Columns Summary Results

Help

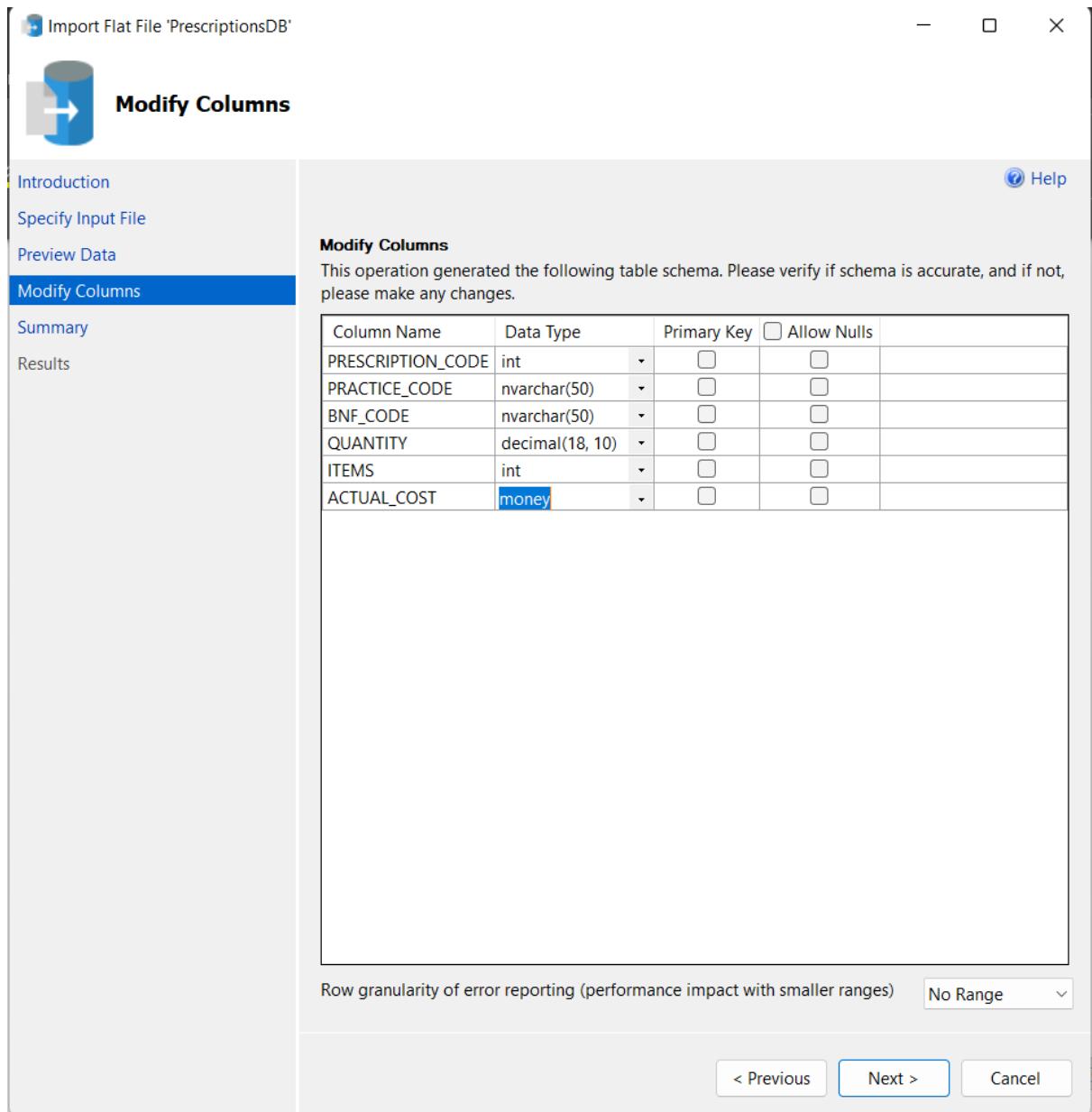
Modify Columns

This operation generated the following table schema. Please verify if schema is accurate, and if not, please make any changes.

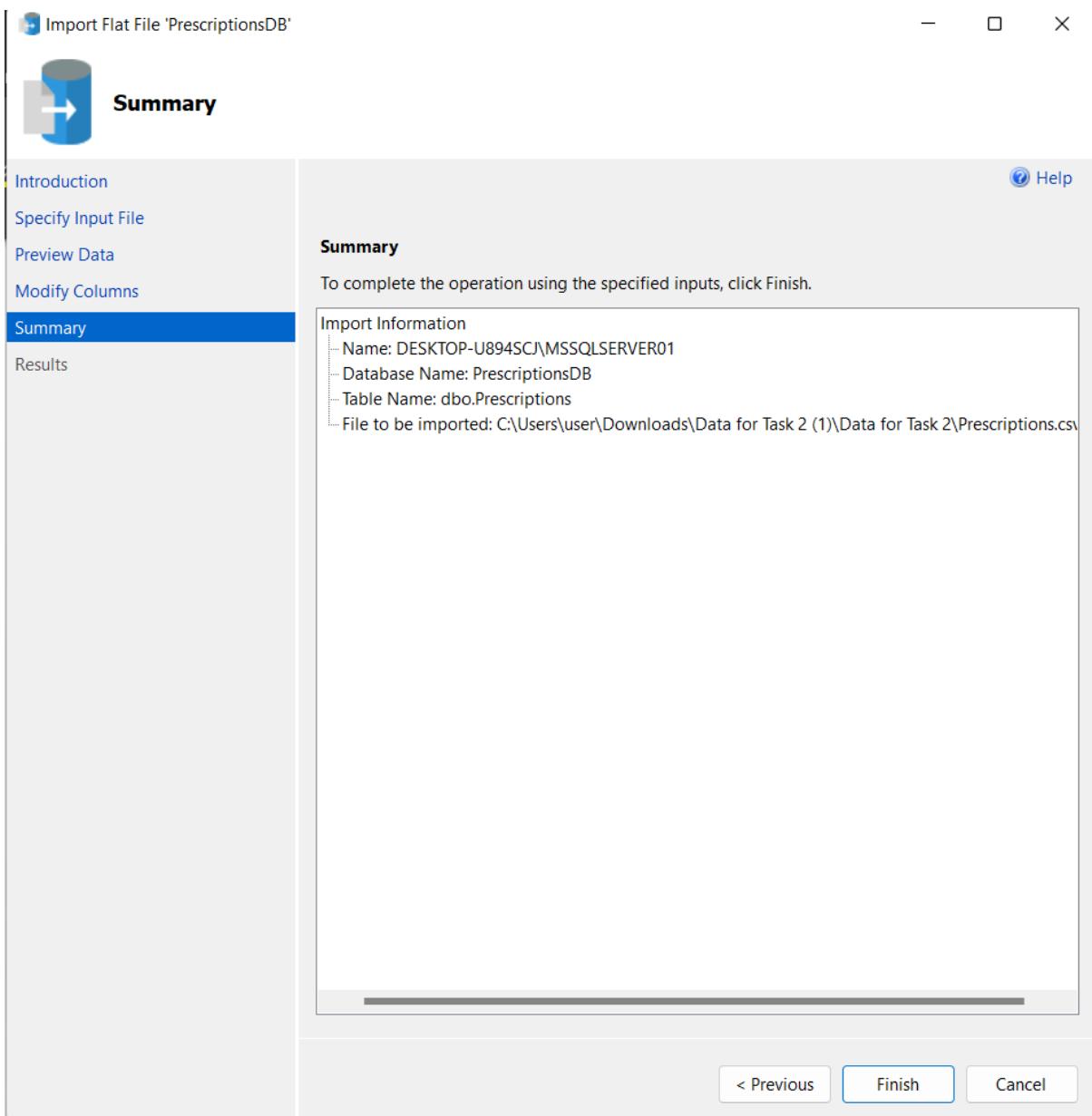
Column Name	Data Type	Primary Key	<input type="checkbox"/> Allow Nulls
PRESCRIPTION_CODE	int	<input type="checkbox"/>	<input type="checkbox"/>
PRACTICE_CODE	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
BNF_CODE	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>
QUANTITY	decimal(18, 10)	<input type="checkbox"/>	<input type="checkbox"/>
ITEMS	int	<input type="checkbox"/>	<input type="checkbox"/>
ACTUAL_COST	money	<input type="checkbox"/>	<input type="checkbox"/>

Row granularity of error reporting (performance impact with smaller ranges) No Range

< Previous Next > Cancel



The screenshot above shows the changes that was made for the data type.



Import Flat File 'PrescriptionsDB'

 **Results**

Introduction
Specify Input File
Preview Data
Modify Columns
Summary
Results

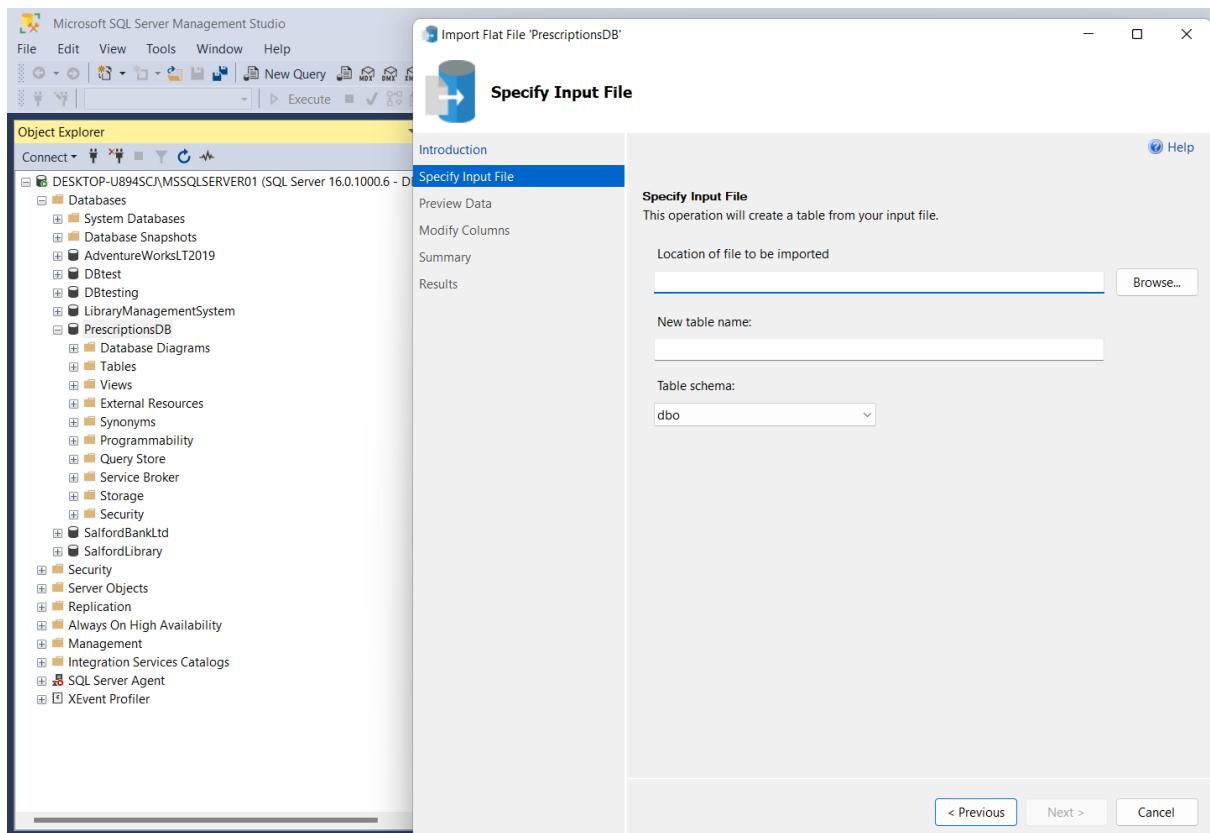
 Help

 **Operation Complete**

Summary:

Name	Result
Insert Data	Success

< Previous Next > Close



```
7
8 -- -- add primary key constraint to Medical_Practice table
9 ALTER TABLE Medical_Practice
10 ADD CONSTRAINT PK_Medical_Practice PRIMARY KEY (practice_code);
11
```

```
11  
12 -- add primary key constraint to Drugs table  
13 ALTER TABLE Drugs  
14 ADD CONSTRAINT PK_Drugs PRIMARY KEY (BNF_CODE)
```

156 %
Messages
Commands completed successfully.
Completion time: 2023-04-22T17:00:40.4767940+01:00

```
16 -- add primary key constraint to Prescriptions table  
17 ALTER TABLE Prescriptions  
18 ADD CONSTRAINT PK_Prescriptions PRIMARY KEY (PRESCRIPTION_CODE);  
19  
20 -- add foreign key constraint for PRACTICE_CODE
```

117 %
Messages
Commands completed successfully.
Completion time: 2023-04-22T17:02:18.5053274+01:00

```
20 foreign key constraint for PRACTICE_CODE  
21 ALTER TABLE Prescriptions  
22 CONSTRAINT FK_Prescriptions_Medical_Practice  
23 KEY (PRACTICE_CODE) REFERENCES Medical_Practice(practice_code);  
24
```

117 %
Messages
Commands completed successfully.
Completion time: 2023-04-22T17:02:57.9650958+01:00

```
25 --- add foreign key constraint for BNF_CODE  
26 ALTER TABLE Prescriptions  
27 ADD CONSTRAINT FK_Prescriptions_Drugs  
28 FOREIGN KEY (BNF_CODE) REFERENCES Drugs(BNF_CODE);  
29
```

117 %
Messages
Commands completed successfully.
Completion time: 2023-04-22T17:05:16.8675537+01:00

2. Write a query that returns details of all drugs which are in the form of tablets or capsules. You can assume that all drugs in this form will have one of these words in the BNF_DESCRIPTION column.

The screenshot shows a SQL query being run in a database environment. The query is designed to find all drugs in tablet or capsule form. It uses the LIKE operator with wildcards (%) and the OR operator to search the BNF_DESCRIPTION column for the words 'tablet' and 'capsule'. The results are displayed in a table with columns: BNF_CODE, CHEMICAL_SUBSTANCE_BNF_DESCR, BNF_DESCRIPTION, and BNF_CHAPTER_PLUS_CODE. The results show various drug entries, mostly from the Gastro-Intestinal System, such as Magnesium oxide, Oromag 160 capsules, Simeticone 100mg capsules, Simeticone 125mg capsules, WindSetlers 100mg capsules, Rennie Peppermint chewable tablets, Rennie Spearmint chewable tablets, Gaviscon Advance Mint chewable tablets, Gaviscon Double Action chewable tablets mint, Gaviscon Peppermint chewable tablets, Gaviscon Strawberry chewable tablets, Calcium carbonate 500mg chewable tablets, Rennie Orange 500mg chewable tablets, Alverine 60mg capsules, Alverine 120mg capsules, Spasmonal 60mg capsules, and Sennaceal Forte 120mg capsules.

```
prescriptionSDB.sq...-U894SC\user (56) 29
29 --- Write a query that returns details of all drugs which are in
30
31
32 SELECT *
33 FROM Drugs
34 WHERE BNF DESCRIPTION LIKE '%tablet%'
35     OR BNF DESCRIPTION LIKE '%capsule%';
36
37 --- Write a query that returns the total quantity for each of pr
38 SELECT PRESCRIPTION_CODE, CAST(ROUND(SUM(CAST(QUANTITY AS FLOAT)
39 FROM Prescriptions
```

BNF_CODE	CHEMICAL_SUBSTANCE_BNF_DESCR	BNF_DESCRIPTION	BNF_CHAPTER_PLUS_CODE
1	01010100AAEAE	Magnesium oxide	01: Gastro-Intestinal System
2	01010100BEAAC	Magnesium oxide	01: Gastro-Intestinal System
3	0101010R0AAAAAA	Simeticone	01: Gastro-Intestinal System
4	0101010R0AAAHAA	Simeticone	01: Gastro-Intestinal System
5	0101010R0BHAHAA	Simeticone	01: Gastro-Intestinal System
6	010102100BAQAF	Compound alginates and proprietary indigestion p...	01: Gastro-Intestinal System
7	010102100BBARAF	Compound alginates and proprietary indigestion p...	01: Gastro-Intestinal System
8	0101021B0BEAQAP	Alginic acid compound preparations	01: Gastro-Intestinal System
9	0101021B0BEARAO	Alginic acid compound preparations	01: Gastro-Intestinal System
10	0101021B0BEAUJAR	Alginic acid compound preparations	01: Gastro-Intestinal System
11	0101021B0BEAWAR	Alginic acid compound preparations	01: Gastro-Intestinal System
12	0101021C0AAAFAP	Calcium carbonate	01: Gastro-Intestinal System
13	0101021C0B1ABA	Calcium carbonate	01: Gastro-Intestinal System
14	0102000A0AAAAAA	Alverine citrate	01: Gastro-Intestinal System
15	0102000A0AAABAB	Alverine citrate	01: Gastro-Intestinal System
16	0102000A0BBAAAA	Alverine citrate	01: Gastro-Intestinal System
17	0102000A0PRAAPD	Alverine citrate	01: Gastro-Intestinal System

Query executed successfully.

The screenshot shown above is a SQL statement that can be used to return the details of all drugs either in the form of tablets or capsules. This query can also be applied in database management system to extract only the relevant data from a large dataset. However, in this context, it would return all records in the "Drugs" table where the BNF_DESCRIPTION column contains the specified keywords, such as drug names that come in tablet or capsule form. This can be done by using wildcard characters (%) and the LIKE operator to search for the specified word (tablets and capsules). The % character will match any string of zero or more characters. The OR operator is used to retrieve rows that contain either of these words. This query will return all columns from the Drugs table for any matching rows. If you only need to retrieve specific columns, you can replace the * with a comma-separated list of column names.

3

Write a query that returns the total quantity for each of prescriptions – this is given by the number of items multiplied by the quantity. Some of the quantities are not integer values and your client has asked you to round the result to the nearest integer value.

The screenshot shows a SQL query window with the following content:

```
36
37 | --- Write a query that returns the total quantity for each of prescriptions
38 | SELECT PRESCRIPTION_CODE, CAST(ROUND(SUM(CAST(QUANTITY AS FLOAT) * CAST(ITEMS AS
39 | FROM Prescriptions
40 | GROUP BY PRESCRIPTION_CODE;
41
```

The results pane displays a table with two columns: PRESCRIPTION_CODE and total_quantity. The data is as follows:

	PREScription_CODE	total_quantity
1	0	308
2	1	224
3	2	336
4	3	84
5	4	560
6	5	168
7	6	448
8	7	672
9	8	252
10	9	30
11	10	28
12	11	56
13	12	90
14	13	56
15	14	84
16	15	30
17	16	112

At the bottom of the results pane, it says "Query executed successfully." and shows the session details: DESKTOP-U894SC\MSQLSERVER... | DESKTOP-U894SC\user (56) | PrescriptionsDB | 00:00:00 | 116,815 rows.

This query above is used to calculate the total quantity of medication being prescribed for each of the prescriptions in the table known as “Prescription”. To do this, it will cast the “QUANTITY” and the “ITEMS” using the “CAST FUNCTION” columns to Float points numbers. After this has been done, it can now be multiplied together which will give room for non-integer values to be joined. The result obtained will then be rounded to the nearest integer using the function known as “ROUND FUNCTION”. Finally, it will cast the rounded total quantity to an integer using another function known as the “CAST FUNCTION” and returns it with the PRESCRIPTION_CODE column using the “GROUP BY CLAUSE” which will then show the resulting output. By executing this query, a summary of the total quantities of the medication prescribed for each prescription can be obtained as shown in the resulting output.

4. Write a query that returns a list of the distinct chemical substances which appear in the Drugs table (the chemical substance is listed in the CHEMICAL_SUBSTANCE_BNF_DESCR column

```

41
42 ---Write a query that returns a list of the distinct chemical substances which appear in the Drugs
43 =SELECT DISTINCT CHEMICAL_SUBSTANCE_BNF_DESCR
44 FROM Drugs;

```

Results

CHEMICAL_SUBSTANCE_BNF_DESCR
1 Sulpiride
2 Almotriptan
3 Hydrocortisone sodium phosphate
4 Loprazolam mesilate
5 Trimethoprim
6 Potassium permanganate
7 Discharge Solidifying Agents
8 Phased formulations of ethinylestradiol
9 Hydrocortisone acetate
10 Beclometasone dipropionate (Systemic)
11 Terbutaline sulphate
12 Antazoline
13 Ivermectin
14 Co-tenidone (Atenolol/chlortalidone)
15 Formoterol/glycopyrronium/budesonide
16 Amissulpride
17 Umanitin budezolidine

Query executed successfully.

The query above is a SQL statement that can be used to retrieve a list of distinct chemical substances that appear in the “DRUGS” table. The keyword is Distinct which will be used to execute two commands which are to returns only the unique value of the “CHEMICAL_SUBSTANCE_BNF_DESCR” column and remove any duplicate values. By executing this query, we can have a summary of the list of all the distinct chemical substances that are appears in the drugs table. The resulting output of this query will provide a simple and effective way to extract relevant information from the database.

5. Write a query that returns the number of prescriptions for each BNF CHAPTER PLUS CODE, along with the average cost for that chapter code, and the minimum and maximum prescription costs for that chapter code.

```

45
46 ---Write a query that returns the number of prescriptions for each BNF CHAPTER PLUS CODE,
47 =SELECT d.BNF CHAPTER PLUS_CODE,
48 COUNT(p.PRESCRIPTION_CODE) AS num_prescriptions,
49 AVG(p.ACTUAL_COST) AS avg_cost,
50 MIN(p.ACTUAL_COST) AS min_cost,
51 MAX(p.ACTUAL_COST) AS max_cost
52 FROM Prescriptions p
53 INNER JOIN Drugs d ON p.BNF_CODE = d.BNF_CODE
54 GROUP BY d.BNF CHAPTER PLUS_CODE;

```

The multi-part identifier “d.BNF CHAPTER PLUS_CODE” could not be bound.

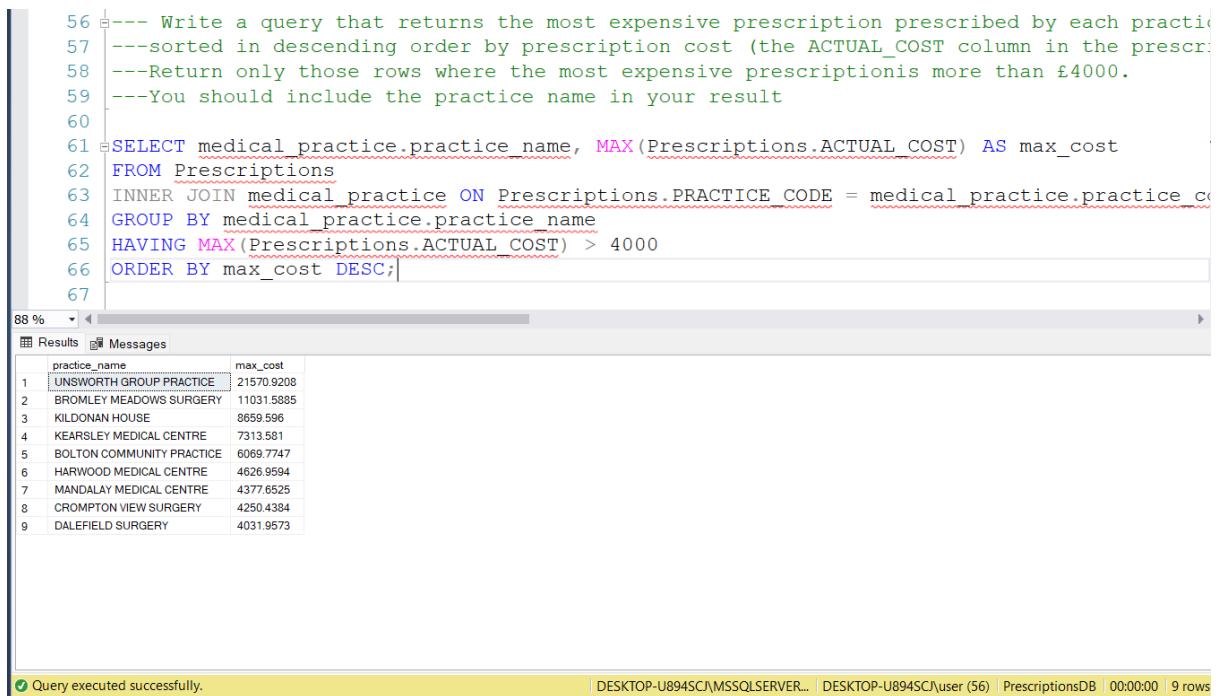
Results

BNF CHAPTER PLUS_CODE	num_prescriptions	avg_cost	min_cost	max_cost
1 11: Eye	2676	21.4403	1.34	434.7997
2 13: Skin	5692	22.2507	0.4864	1402.4766
3 21: Appliances	5856	38.3528	0.2057	2552.9234
4 20: Dressings	1014	40.2214	0.2181	2599.1877
5 19: Other Drugs and Preparations	338	32.2498	0.4331	1193.3481
6 02: Cardiovascular System	19186	35.9956	0.1311	11094.7521
7 05: Infections	4657	21.2474	0.1966	1262.2079
8 23: Stoma Appliances	1697	82.2616	1.5894	882.5271
9 08: Malignant Disease and Immunosuppression	754	54.9382	0.29	2197.1551
10 22: Incontinence Appliances	535	42.8654	2.8703	538.8857
11 01: Gastro-Intestinal System	8777	35.1252	0.1405	2777.6906
12 10: Musculoskeletal and Joint Diseases	3634	16.2459	0.2713	1476.6376
13 09: Nutrition and Blood	7944	41.8156	0.1405	4250.4384
14 12: Ear, Nose and Oropharynx	1274	23.6805	1.6766	315.7777
15 07: Obstetrics, Gynaecology and Urinary-Tract Dis...	3999	25.4563	0.1498	803.4917
16 03: Respiratory System	7057	75.873	0.1498	4161.8103
17 04: Central Nervous System	26966	29.2004	0.1211	3765.632

Query executed successfully.

This code above is used to retrieve aggregate statistics on medication prescriptions in the "Prescriptions" and "Drugs" tables. The "Prescription" and "Drugs" table is being joined together by a clause known as "INNER JOIN" based on the matching value in the "BNF_CODE" column. The resulting data will be grouped by "BNF CHAPTER PLUS CODE" column in the "Drugs" table and applies several aggregate functions to the "ACTUAL_COST" column in the "Prescriptions" table, including COUNT, AVG, MIN, and MAX. By executing this query, a summary of medication prescriptions grouped by the BNF chapter code in the "Drugs" table, along with various statistics on the costs of those prescriptions can be obtained as shown above.

6. Write a query that returns the most expensive prescription prescribed by each practice, sorted in descending order by prescription cost (the ACTUAL_COST column in the prescription table.) Return only those rows where the most expensive prescription is more than £4000. You should include the practice name in your result.



```

56 --- Write a query that returns the most expensive prescription prescribed by each practice
57 ---sorted in descending order by prescription cost (the ACTUAL_COST column in the prescription table)
58 ---Return only those rows where the most expensive prescriptionis more than £4000.
59 ---You should include the practice name in your result
60
61 SELECT medical_practice.practice_name, MAX(Prescriptions.ACTUAL_COST) AS max_cost
62 FROM Prescriptions
63 INNER JOIN medical_practice ON Prescriptions.PRACTICE_CODE = medical_practice.practice_code
64 GROUP BY medical_practice.practice_name
65 HAVING MAX(Prescriptions.ACTUAL_COST) > 4000
66 ORDER BY max_cost DESC;
67

```

practice_name	max_cost
1 UNSWORTH GROUP PRACTICE	21570.9208
2 BROMLEY MEADOWS SURGERY	11031.5885
3 KILDONAN HOUSE	8659.596
4 KEARSLEY MEDICAL CENTRE	7313.581
5 BOLTON COMMUNITY PRACTICE	6069.7747
6 HARWOOD MEDICAL CENTRE	4626.9594
7 MANDALAY MEDICAL CENTRE	4377.6525
8 CROMPTON VIEW SURGERY	4250.4384
9 DALEFIELD SURGERY	4031.9573

Query executed successfully. | DESKTOP-U894SC\MSQLSERVER... | DESKTOP-U894SC\user (56) | PrescriptionsDB | 00:00:00 | 9 rows

This code is a SQL query. It joins the Prescriptions and "MEDICAL_PRACTICE" tables on the "PRACTICE_CODE" column and groups the result by the "PRACTICE_CODE" column. The "MAX" function is used to calculate the maximum "ACTUAL_COST" for each group used to retrieve the name of medical practices and the maximum actual cost of prescriptions associated with each practice. The query uses the INNER JOIN clause to join the "Prescriptions" and "medical_practice" tables based on a matching value in the "PRACTICE_CODE" column. The HAVING clause filters the results to only include groups with a maximum actual cost greater than 4000. The ORDER BY clause sorts the results

in descending order of maximum actual cost. By executing this query, we can identify medical practices that have prescribed medications with high costs. This information can be used to investigate prescribing patterns and identify opportunities for cost savings. Additionally, this query can provide insights into the effectiveness of cost-control measures implemented by medical practices.

7. ADDITIONAL FIVE QUERIES

1. Query to find the average cost of prescription for each BNF chapter.

```

68 ---Five additional Queries
69 ---Query to find the average cost of prescriptions for each BNF chapter:
70 SELECT Drugs.BNF CHAPTER PLUS CODE, AVG(Prescriptions.ACTUAL COST) AS avg_cost
71 FROM Drugs
72 INNER JOIN Prescriptions ON Drugs.BNF_CODE = Prescriptions.BNF_CODE
73 GROUP BY Drugs.BNF CHAPTER PLUS CODE
74 ORDER BY avg_cost DESC;
75
76 ---Query to find all drugs that have been prescribed by a specific practice:

```

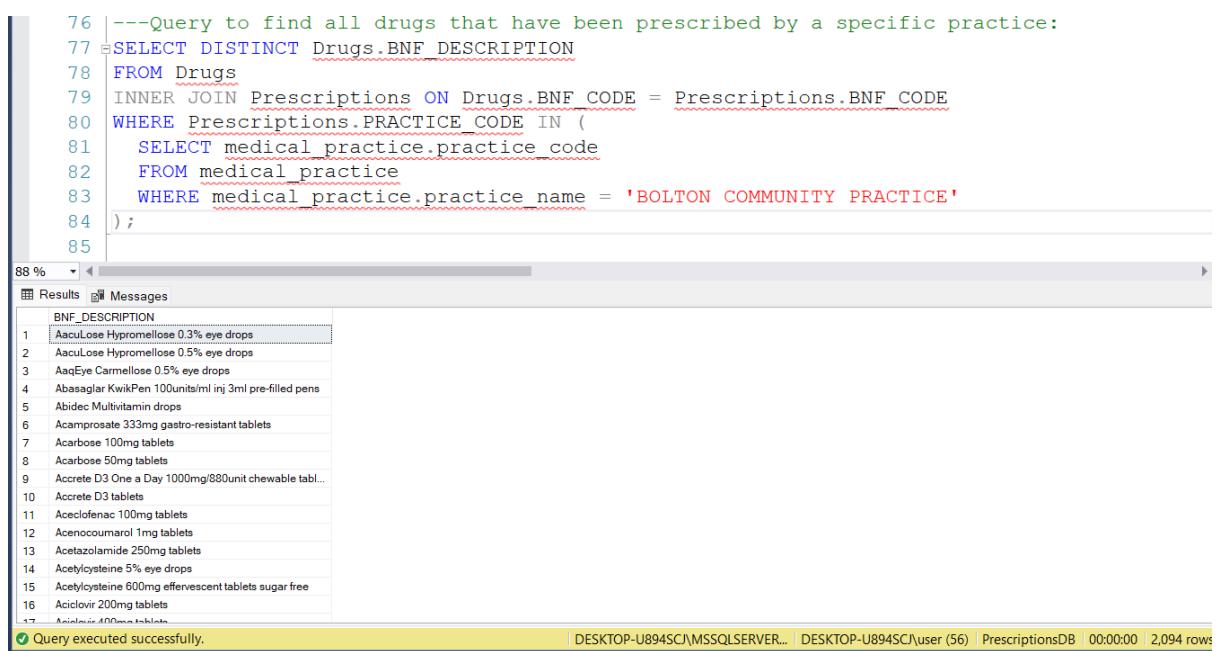
Results Messages

BNF CHAPTER PLUS CODE	avg_cost
14: Immunological Products and Vaccines	1049.5574
23: Stoma Appliances	82.2616
03: Respiratory System	75.873
06: Endocrine System	61.1661
08: Malignant Disease and Immunosuppression	54.9382
22: Incontinence Appliances	42.8654
09: Nutrition and Blood	41.8156
20: Dressings	40.2214
21: Appliances	38.3528
10: Cardiovascular System	35.9956
15: Anaesthesia	35.6672
01: Gastro-Intestinal System	35.1252
19: Other Drugs and Preparations	32.2498
14: Central Nervous System	28.3994
07: Obstetrics, Gynaecology and Urinary-Tract Dis...	25.4563
12: Ear, Nose and Oropharynx	23.6805
17: Skin	22.2607

Query executed successfully. DESKTOP-U894SC1\MSQLSERVER... DESKTOP-U894SC1\user (56) PrescriptionsDB | 00:00:00 | 20 rows

This query above can be used to calculate the average cost of medications prescribed for each BNF chapter code. The “DRUGS” and “PRESCRIPTION” table are being joined together using based on a matching value in the “BNF_CODE” column using the clause known as “INNER CLAUSE”. The query then groups the result by the “BNF CHAPTER PLUS CODE” column in the Drugs table and uses the AVG function to calculate the average “ACTUAL_COST” for each group. The result is sorted in descending order by the average cost using the “ORDER BY CLAUSE”. Executing this code, it will enable us to identify the BNF chapter codes that are related to only the highest average cost of prescriptions. Cost effectiveness of the medication of the different categories can be identify.

2. Query to find all drugs that have been prescribed by a specific practice:



The screenshot shows a SQL query in the query editor and its execution results. The query retrieves distinct BNF descriptions of drugs prescribed by the 'BOLTON COMMUNITY PRACTICE'. The results grid displays 2,094 rows of medication names.

```
76 ---Query to find all drugs that have been prescribed by a specific practice:
77 SELECT DISTINCT Drugs.BNF_DESCRIPTION
78 FROM Drugs
79 INNER JOIN Prescriptions ON Drugs.BNF_CODE = Prescriptions.BNF_CODE
80 WHERE Prescriptions.PRACTICE_CODE IN (
81     SELECT medical_practice.practice_code
82     FROM medical_practice
83     WHERE medical_practice.practice_name = 'BOLTON COMMUNITY PRACTICE'
84 );
85
```

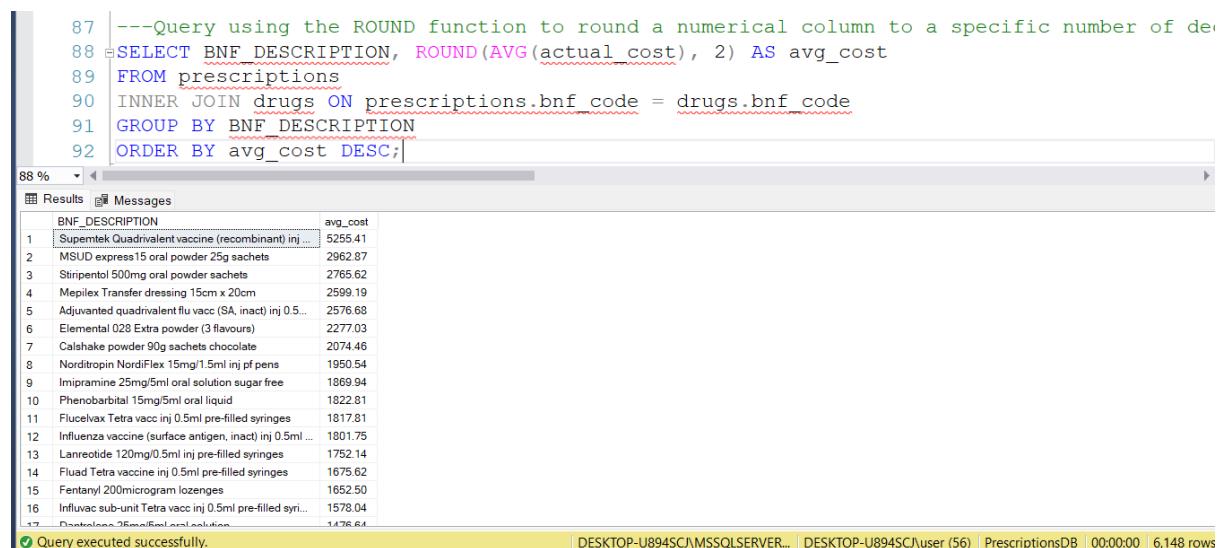
BNF_DESCRIPTION
1 AacuLoss Hypromellose 0.3% eye drops
2 AacuLoss Hypromellose 0.5% eye drops
3 AaqEye Carmellose 0.5% eye drops
4 Abasaglar KwikPen 100units/ml in 3ml pre-filled pens
5 Abidec Multivitamin drops
6 Acamprostate 333mg gastro-resistant tablets
7 Acarbose 100mg tablets
8 Acarbose 50mg tablets
9 Accrete D3 One a Day 1000mg/880unit chewable tabl...
10 Accrete D3 tablets
11 Acetclofenac 100mg tablets
12 Acenocoumarol 1mg tablets
13 Acetazolamide 250mg tablets
14 Acetylcysteine 5% eye drops
15 Acetylcysteine 600mg effervescent tablets sugar free
16 Aciclovir 200mg tablets
17 Aciclovir 400mg tablets

Query executed successfully. | DESKTOP-U894SC\MSQLSERVER... | DESKTOP-U894SC\user (56) | PrescriptionsDB | 00:00:00 | 2,094 rows

This code is a SQL query used to retrieve a list of distinct medication names linked with prescriptions issued by the "BOLTON COMMUNITY PRACTICE". A clause known as "INNER JOIN" joins the "DRUGS" and "PRESCRIPTIONS" table together. The resulting data is filtered by selecting only prescriptions issued by the "BOLTON COMMUNITY PRACTICE". The query then filters the result to only include rows where the PRACTICE_CODE column in the Prescriptions table matches any of the practice codes returned by a subquery that retrieves the code for a specific practice. The DISTINCT keyword is used to remove duplicates, and the SELECT statement retrieves the BNF_DESCRIPTION column. By executing this query, we can identify the distinct medication names prescribed by the "BOLTON COMMUNITY PRACTICE" and gain insights into prescribing patterns for this medical practice.

3.

Query using the ROUND function to round a numerical column to a specific number of decimal places:



The screenshot shows a SQL query results window. The query is as follows:

```
87 ---Query using the ROUND function to round a numerical column to a specific number of decimal places
88 SELECT BNF_DESCRIPTION, ROUND(AVG(actual_cost), 2) AS avg_cost
89 FROM prescriptions
90 INNER JOIN drugs ON prescriptions.bnf_code = drugs.bnf_code
91 GROUP BY BNF_DESCRIPTION
92 ORDER BY avg_cost DESC;
```

The results table has two columns: BNF_DESCRIPTION and avg_cost. The data is as follows:

BNF_DESCRIPTION	avg_cost
Supemetek Quadrivalent vaccine (recombinant) inj	5255.41
MSUD express15 oral powder 25g sachets	2962.87
Stripentol 500mg oral powder sachets	2765.62
Mepilex Transfer dressing 15cm x 20cm	2599.19
Adjuvanted quadrivalent flu vacc (SA, inact) inj 0.5ml	2576.68
Elemental 028 Extra powder (3 flavours)	2277.03
Calshake powder 900g sachets chocolate	2074.46
Norditropin Nordiflex 15mg/1.5ml inj pf pens	1950.54
Imipramine 25mg/5ml oral solution sugar free	1869.94
Phenobarbital 15mg/5ml oral liquid	1822.81
Flucelvax Tetra vacc inj 0.5ml pre-filled syringes	1817.81
Influenza vaccine (surface antigen, inact) inj 0.5ml	1801.75
Lanreotide 120mg/0.5ml inj pre-filled syringes	1752.14
Fluad Tetra vaccine inj 0.5ml pre-filled syringes	1675.62
Fentanyl 200microgram lozenges	1652.50
Influvac sub-unit Tetra vac inj 0.5ml pre-filled syringes	1578.04
Drotaverine 25mg oral solution	1476.64

At the bottom of the window, it says "Query executed successfully." and shows the session details: DESKTOP-U894SCJ\SQLSERVER... | DESKTOP-U894SCJ\user (56) | PrescriptionsDB | 00:00:00 | 6,148 rows.

This code is a SQL query that extracts the details of medication names from the "DRUGS" table using the "BNF_CODE" and calculates the average actual cost of prescriptions for each medication. The ROUND function is used to round the resulting average cost values to two decimal places and orders the result by the average cost in descending order using the "ORDER BY" clause. The query uses the INNER JOIN clause to join the "DRUGS" and "PRESCRIPTIONS" tables based on a matching value in the "BNF_CODE" column. It then applies the AVG aggregate function to the "ACTUAL_COST" column in the "PRESCRIPTION" table and groups the resulting data by the "BNF_DESCRIPTION" column in the "DRUGS" table. By executing this query, we can identify the medications that have the highest average cost per prescription.

4. query to Return the names of all medical practices in Bolton that have not prescribed any drug in the "06: Endocrine system" BNF chapter code:

```

94 ---Return the names of all medical practices in Bolton that have not prescribed any drug
95 SELECT practice_name
96 FROM medical_practice
97 WHERE NOT EXISTS (
98     SELECT *
99     FROM prescriptions
100    INNER JOIN drugs ON prescriptions.BNF_CODE = drugs.BNF_CODE
101   WHERE prescriptions.PRACTICE_CODE = medical_practice.PRACTICE_CODE
102     AND drugs.BNF CHAPTER_PLUS_CODE = '06'
103 )

```

The screenshot shows a SQL query being run in a database environment. The query retrieves the names of medical practices from the 'medical_practice' table where no prescriptions exist for medications in the '06: Endocrine system' BNF chapter. The results are displayed in a table titled 'Results'.

practice_name
1 UNIDENTIFIED DOCTORS
2 THE DUNSTAN PARTNERSHIP
3 PIKES LANE 1
4 KILDONAN HOUSE
5 SWAN LANE MEDICAL CENTRE
6 STABLE FOLD SURGERY
7 DR MALHOTRA & PARTNERS
8 KEARSLEY MEDICAL CENTRE
9 STONEHILL MEDICAL CENTRE
10 ST HELENS ROAD PRACTICE
11 DALEFIELD SURGERY
12 TONGE FOLD HEALTH CENTRE
13 DR EARNSHAW AND PARTNERS
14 LEVER CHAMBERS 2
15 SPRING HOUSE SURGERY
16 UNSWORTH GROUP PRACTICE
17 HAMWOOD MEDICAL CENTRE

Query executed successfully.

This code is a SQL query that retrieves the names of medical practices that have not prescribed any medication belonging to the BNF chapter “06”. The query uses a subquery with the NOT EXISTS operator to identify all prescription records that match a specific condition: the BNF code of the prescribed medication belongs to the BNF chapter 06, and the prescription is associated with a medical practice based on the matching "PRACTICE_CODE" values. The outer query then retrieves the "practice_name" from the "medical_practice" table for all practices that do not have

any matching prescription records in the subquery. By executing this query, we can identify medical practices that have not prescribed any medications belonging to the BNF chapter 06, which includes medications for the treatment of endocrine disorders. This information can be used to identify potential gaps in patient care, evaluate prescribing patterns for different medical practices, and inform decision-making around drug formularies and treatment guidelines.

1. Query to get the average cost of prescriptions for each medical practice in the Medical_Practice table, ordered by the average cost in ascending order:

The screenshot shows a SQL query being run in a database environment. The query retrieves the average cost of prescriptions for each medical practice, filtering for those with an average cost less than 10, and ordering the results by average cost in ascending order. The results table shows four entries: BOLTON HOSPICE, NEURO-REHAB SERVICE, TURTON PCN HUB, and UNIDENTIFIED DOCTORS, with their respective average prescription costs.

```
--> 105 --- Query to get the average cost of prescriptions for each medical practice in the Medical_Practice table
106 <-- SELECT
107     practice_name,
108     AVG(ACTUAL_COST) AS avg_cost
109 FROM
110     Medical_Practice
111 JOIN Prescriptions ON Medical_Practice.practice_code = Prescriptions.practice_code
112 GROUP BY
113     practice_name
114 HAVING
115     AVG(ACTUAL_COST) < 10
116 ORDER BY
117     avg_cost ASC;
```

	practice_name	avg_cost
1	BOLTON HOSPICE	1.8888
2	NEURO-REHAB SERVICE	2.562
3	TURTON PCN HUB	9.5266
4	UNIDENTIFIED DOCTORS	9.9915

This code is a SQL query that retrieves the average cost of prescriptions for each medical practice in the Medical_Practice table and orders the results by the average cost in ascending order. The query joins the Medical_Practice and Prescriptions tables on the matching "practice_code" values and calculates the average cost of all prescriptions associated with each medical practice using the AVG function. The HAVING clause is used to filter the results by only including medical practices whose average prescription cost is less than 10.

By executing this query, we can identify the medical practices with the lowest average prescription costs. This information can be useful for evaluating the efficiency of medical practices, identifying cost-saving opportunities, and informing decision-making around drug formularies and treatment guidelines. However, it's important to note that the results of this query may be affected by factors such as differences in patient populations and prescribing patterns, so further analysis may

be necessary to fully understand the underlying causes of differences in prescription costs between medical practices.