

TASK 2

Introduction

In Task 2, the objective is to create a database for a food service company based on provided CSV files containing information about restaurants, consumers, ratings, and restaurant cuisines. The dataset consists of four related tables: Restaurants, Consumers, Ratings, and Restaurant_Cuisines.

To tackle this task, we need to perform the following steps:

1. Create a database named FoodserviceDB.
2. Import the four CSV files into separate tables in the database.
3. Ensure the appropriate primary and foreign key constraints are added to maintain data integrity.
4. Provide a database diagram illustrating the relationships between the tables.

I'll start by setting up the database, importing the CSV files, defining the table structures, and establishing the necessary relationships between them. Then, I'll generate a visual representation of the database schema to demonstrate the relationships effectively. Let's proceed with implementing these steps.

1. Creating a database:

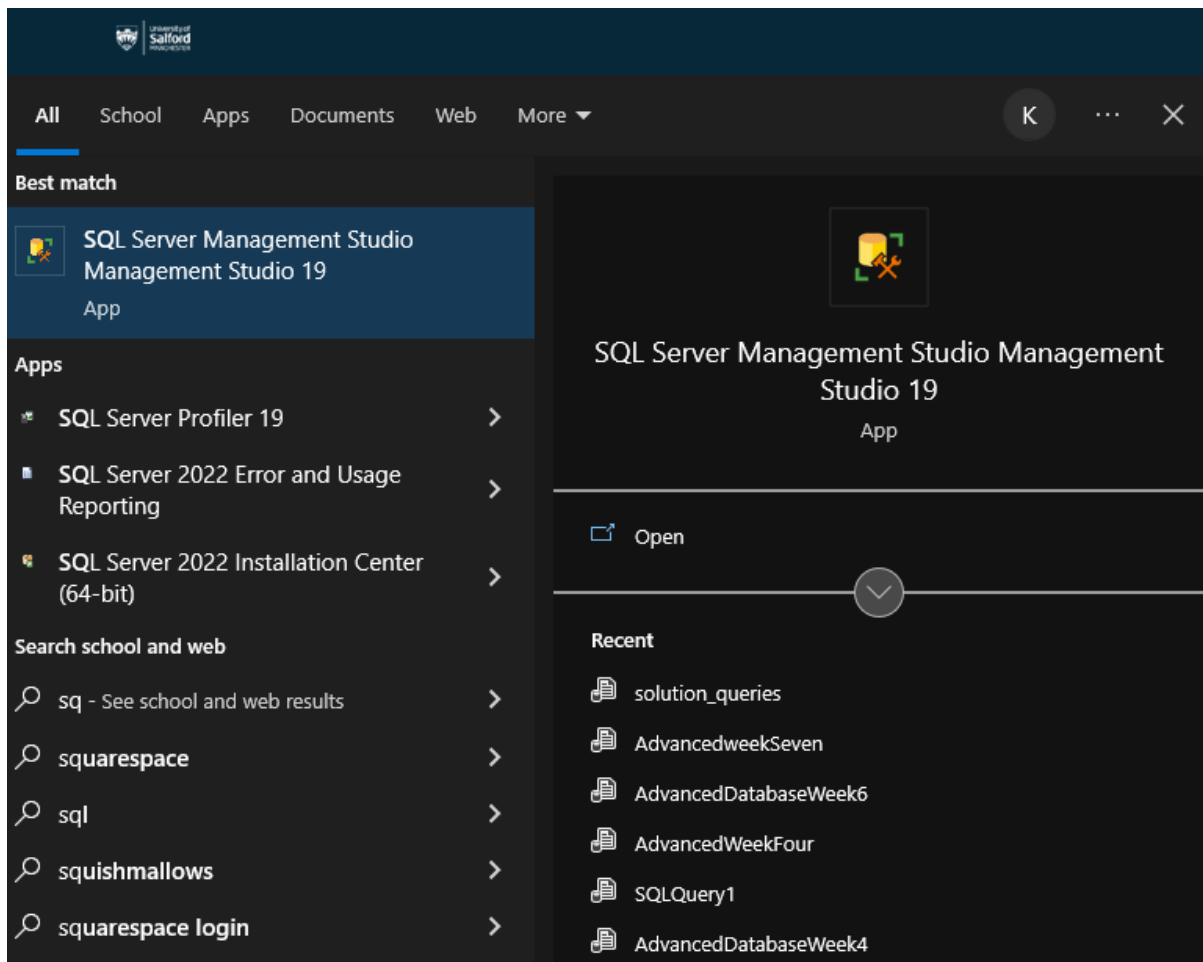


Fig 1.1.1

Searching for “SQL Server Managenment Studio”

The first thing I did on my device was launch SSMS by typing the word "SQL Server Management Studio" into the Windows search box.

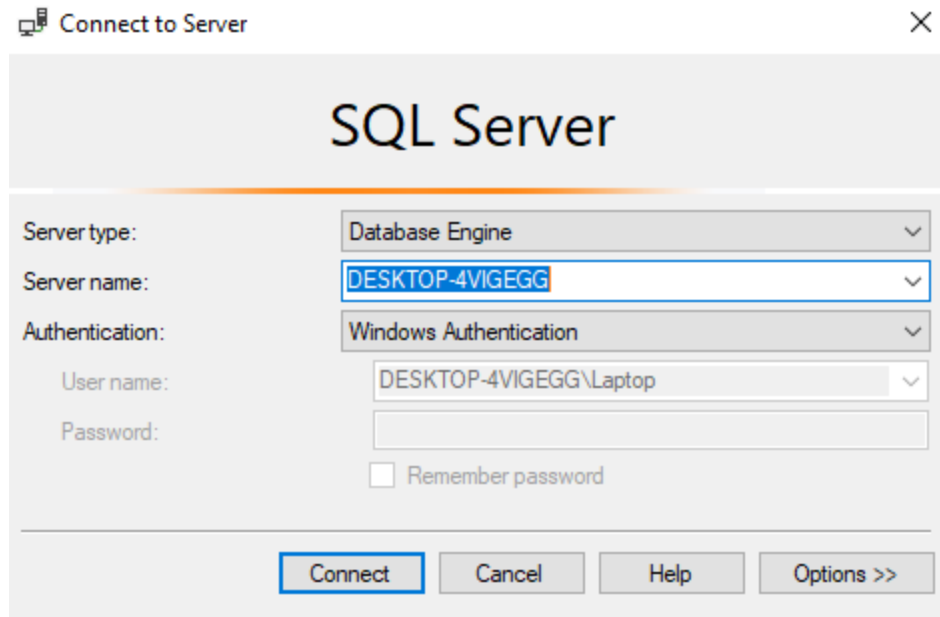


Fig 1.1.2

Connecting to the server

On the next screen, you will find the option to connect to the SQL server by clicking on the connect button.

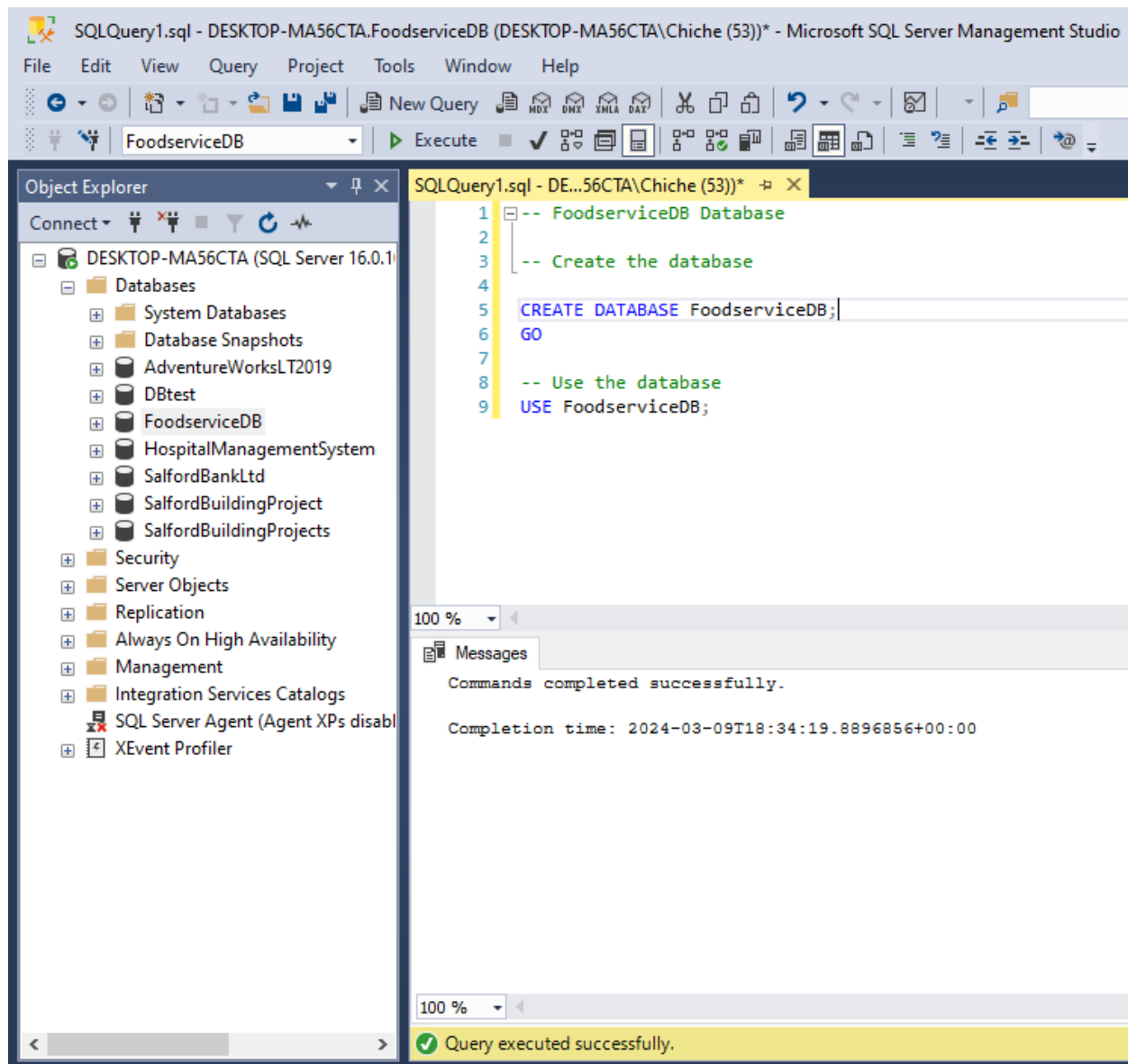
Creating the database using the CREATE DATABASE statement below:

-- Create the database

```
CREATE DATABASE FoodserviceDB;
```

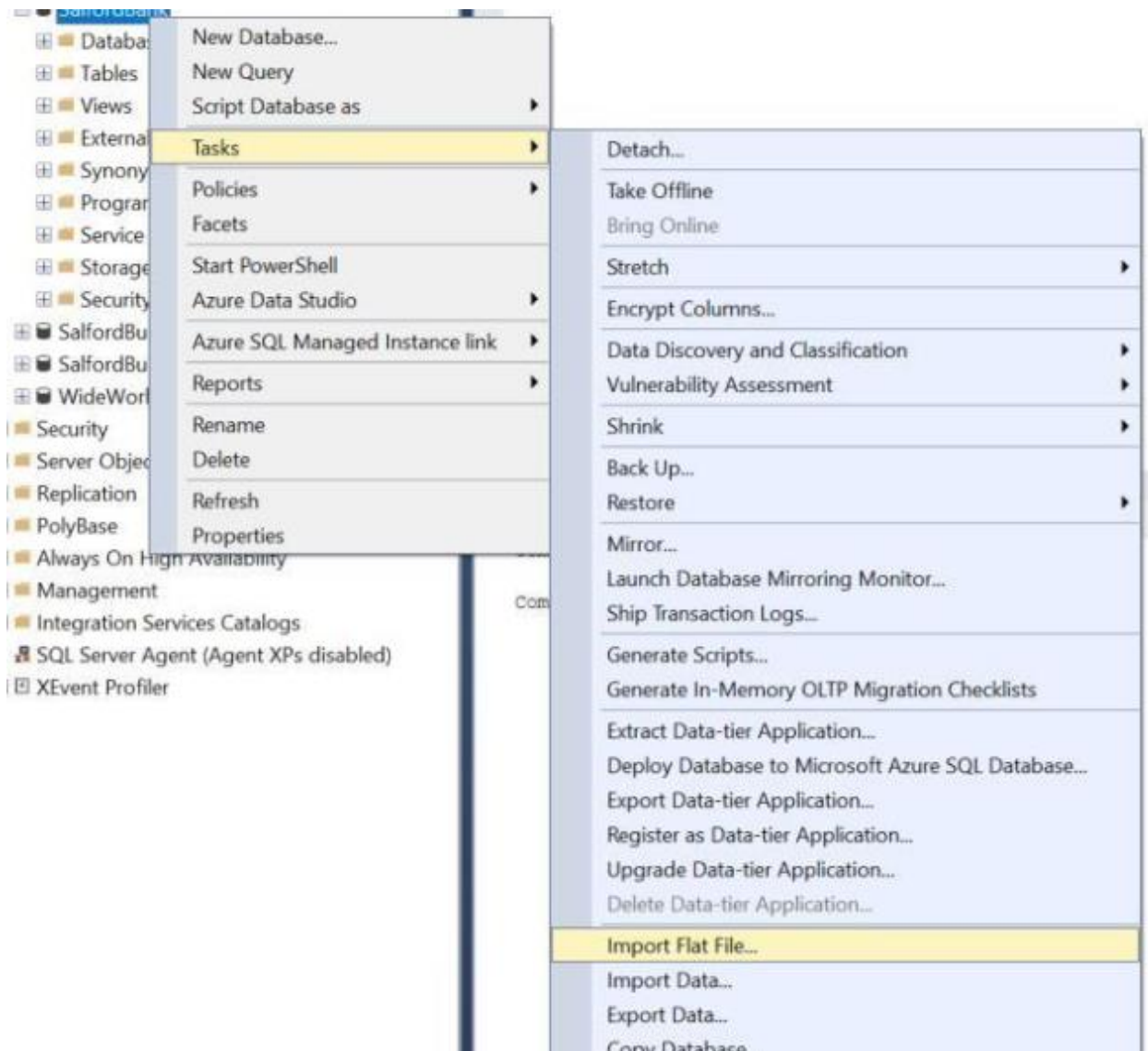
The below command is use to set the newly created FoodserviceDB database:

```
USE FoodserviceDB; GO
```

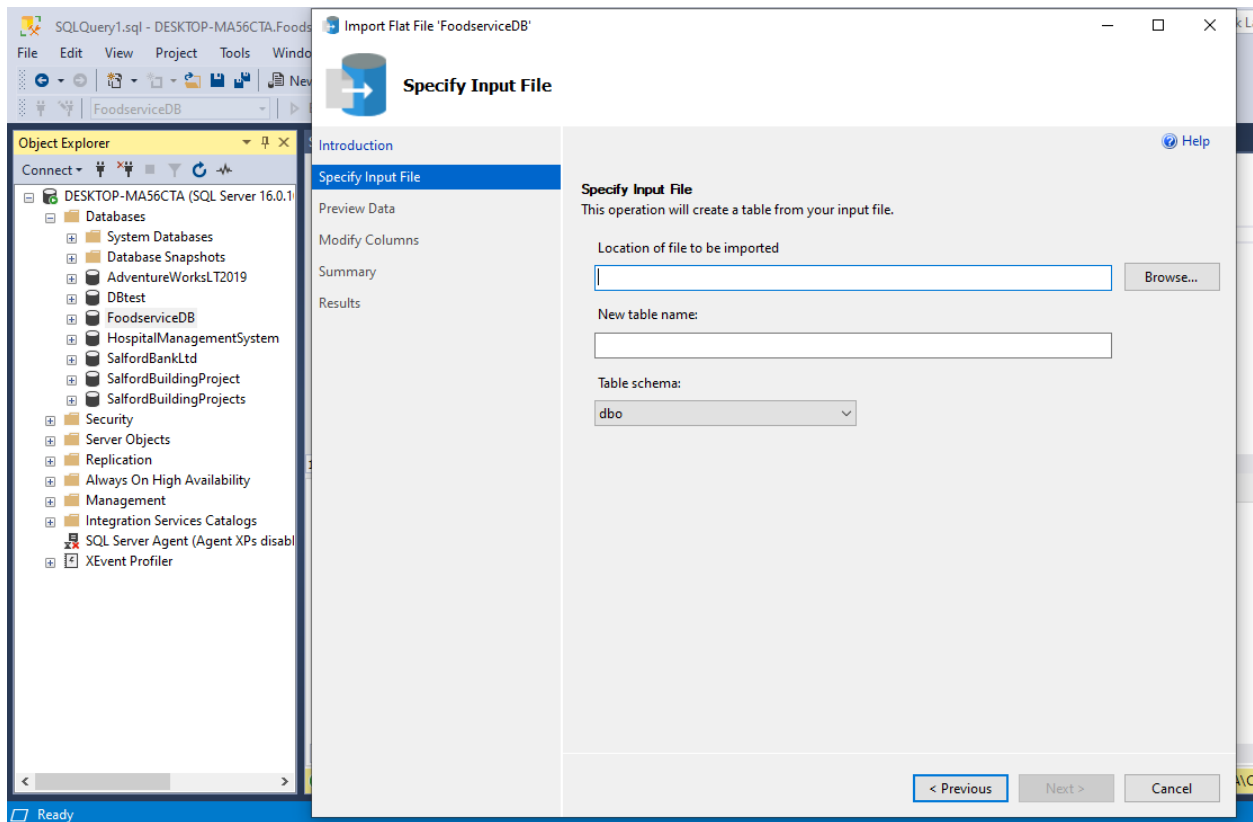


2. Importing Flat Files:

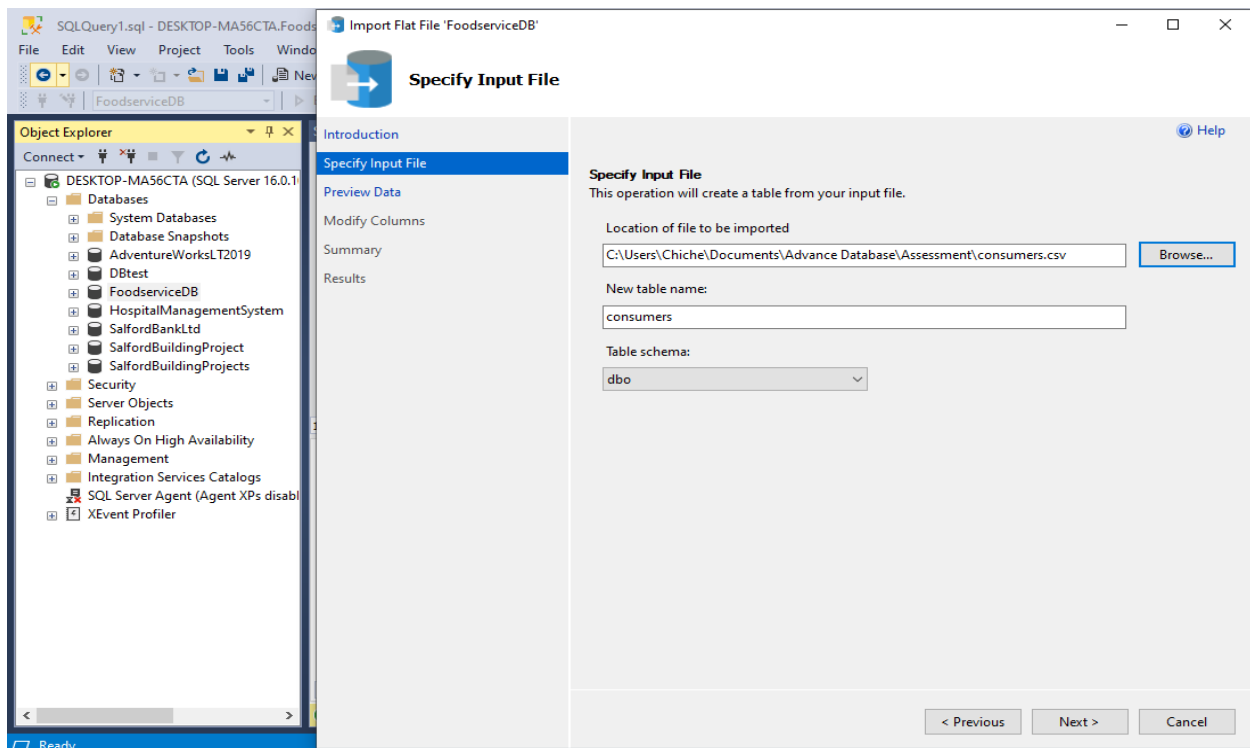
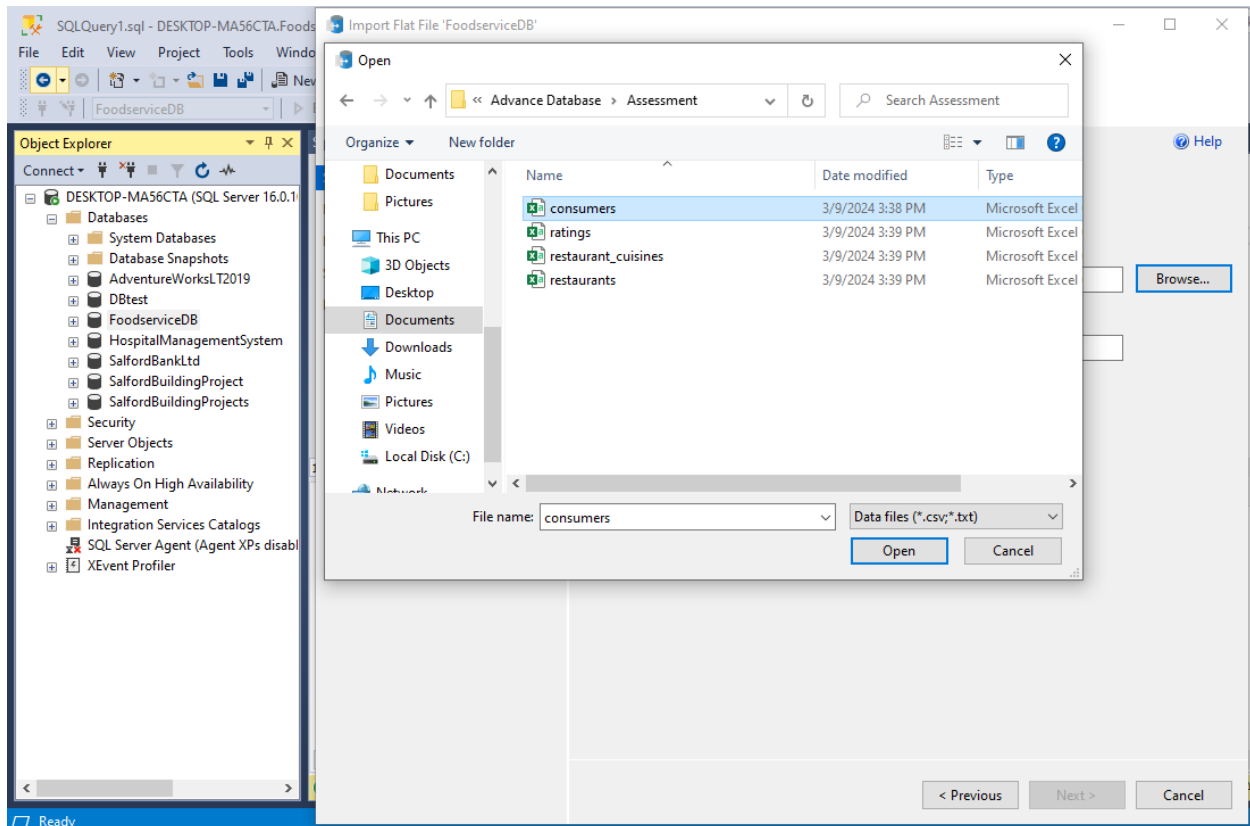
I imported the flat files using the SSMS interface. This is achieved by right clicking on the database, select Tasks from the menu and then select Import Flat File.

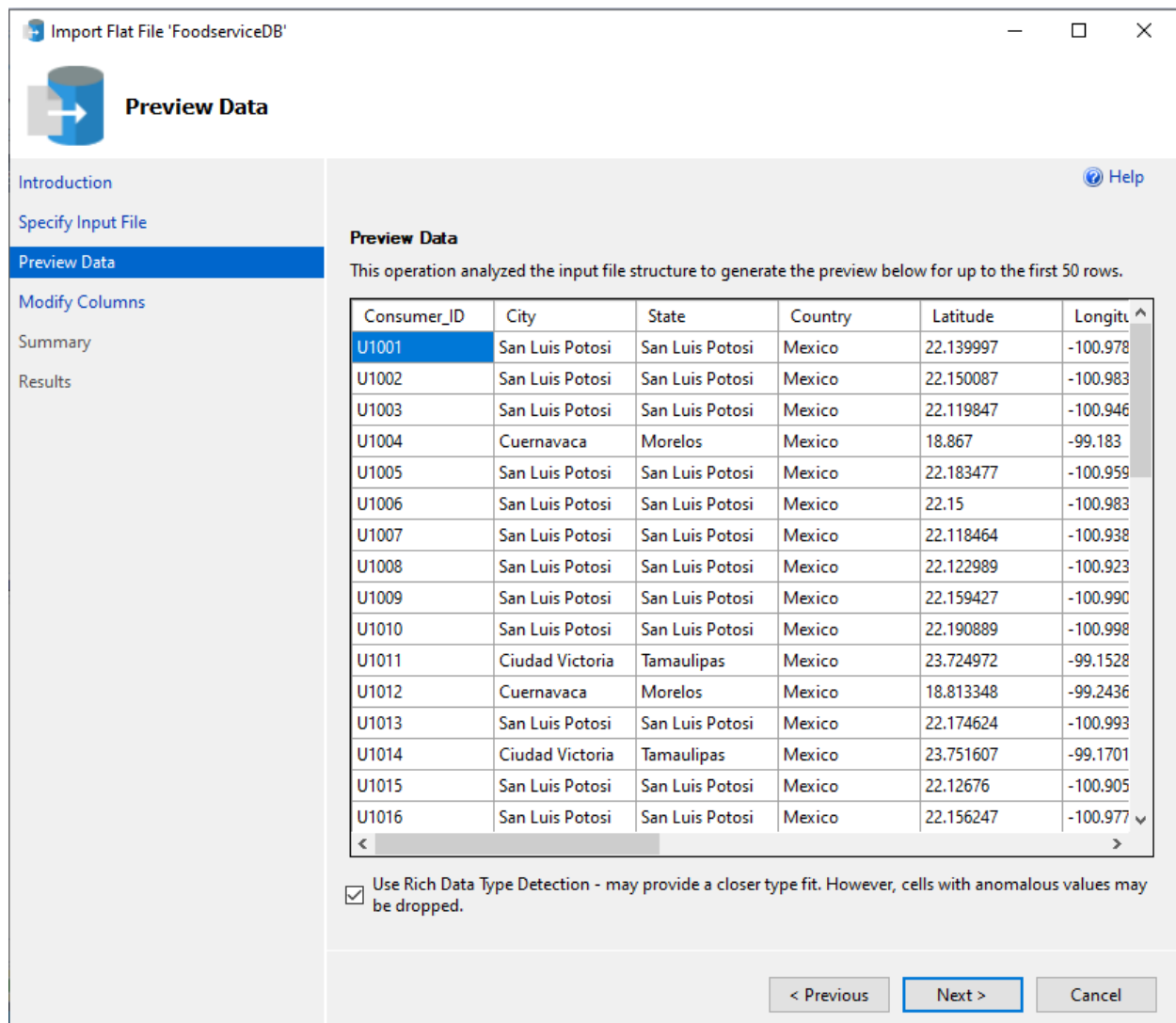


On doing this, in the window that opens we click on Browse and then navigate to where I have saved the csv file downloaded from Blackboard.



On selecting the file, select Next. You can check the preview of the file we are importing. Leave 'Use Rich Data Type Detection' ticked and click Next again.





On the next screen, you should check the data types it has identified for each of the columns. We allow most of these as they are. However, you should change `AccountNumber` and `BranchCode` to `nvarchar` as we don't want to treat these as numbers (some account numbers may start with a leading zero, which will be dropped if we store this in a numeric format). You should change `AccountBalance` to `money` and `AccountInterestRate` to `decimal(4,2)`.

**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	
Consumer_ID	nvarchar(50) ▾	<input type="checkbox"/>	<input type="checkbox"/>	
City	nvarchar(50) ▾	<input type="checkbox"/>	<input type="checkbox"/>	
State	nvarchar(50) ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Country	nvarchar(50) ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Latitude	float ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Longitude	float ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Smoker	bit ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Drink_Level	nvarchar(50) ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Transportation_Method	nvarchar(50) ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Marital_Status	nvarchar(50) ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Children	nvarchar(50) ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Age	tinyint ▾	<input type="checkbox"/>	<input type="checkbox"/>	
Occupation	nvarchar(50) ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Budget	nvarchar(50) ▾	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

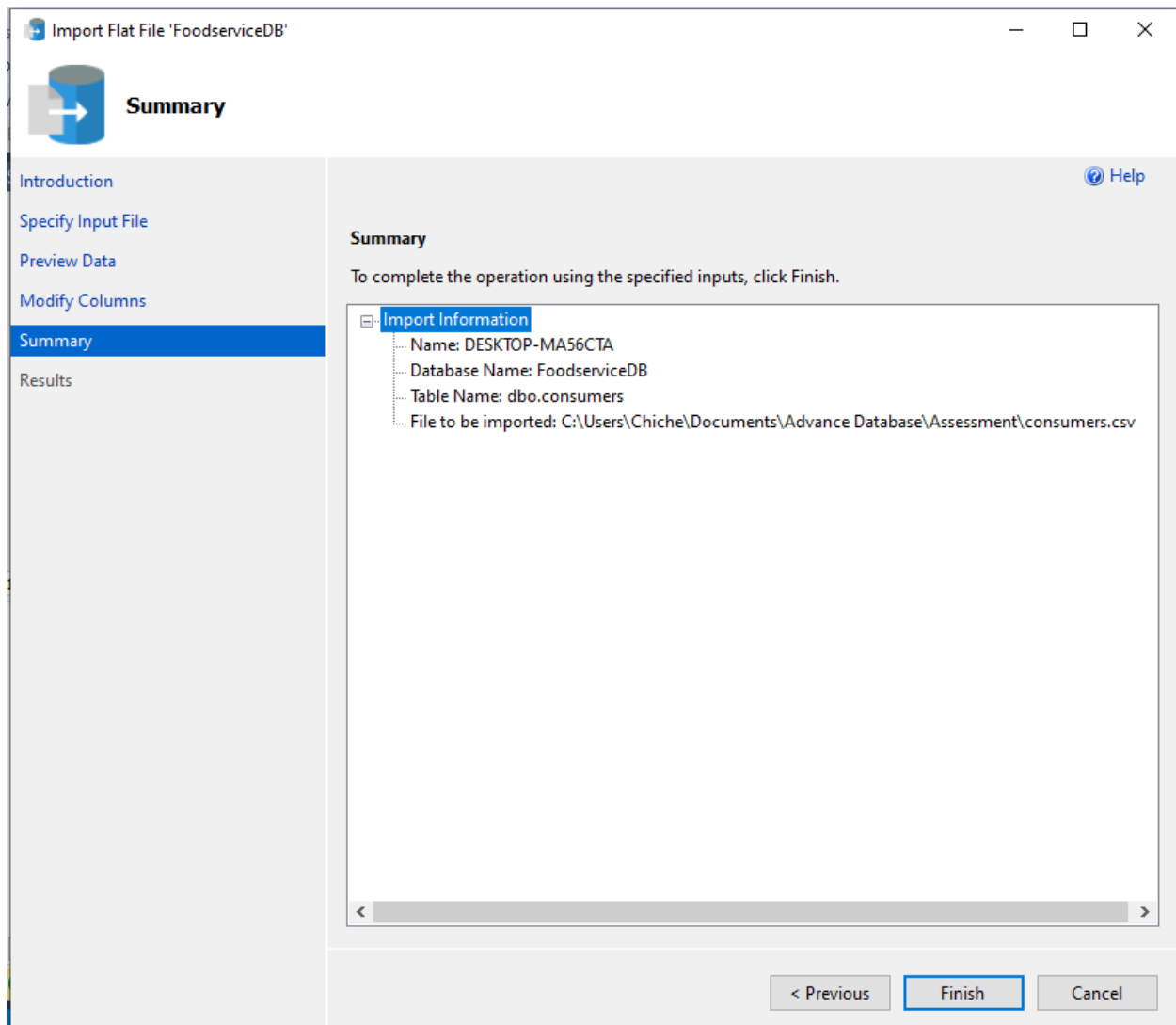
Row granularity of error reporting (performance impact with smaller ranges)

No Range ▾

< Previous

Next >

Cancel



1. After the import has been completed, the execution status indicates that the import has been successful.
2. Lastly, you click on the "close" button on the bottom of the screen.

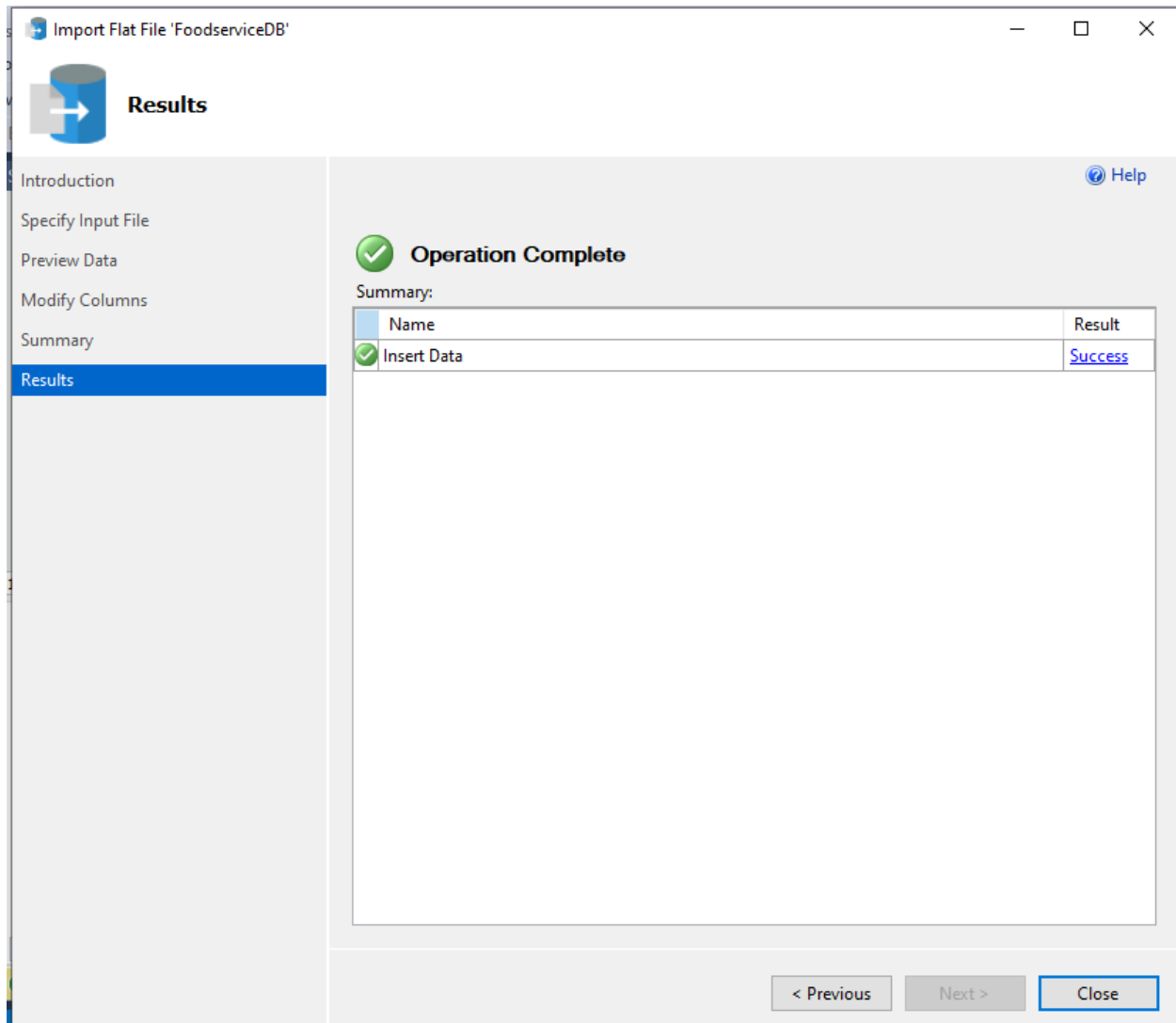


Fig 1.1.11

1. After the import has been completed, the execution status indicates that the import has been successful.
2. Lastly, you click on the "close" button on the bottom of the screen.

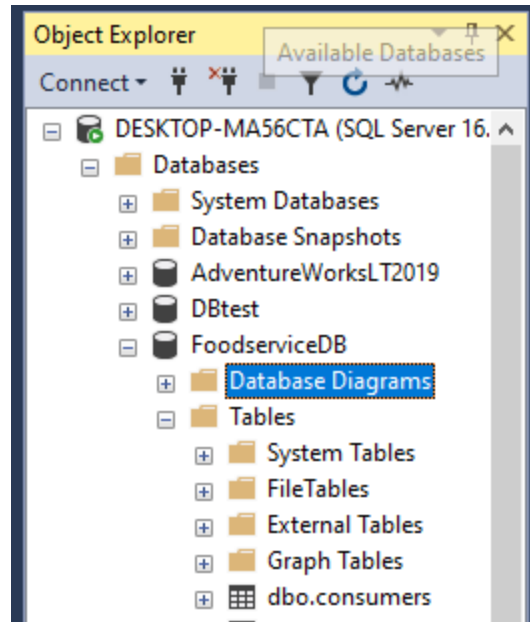


Fig 1.1.12

1. As a result, the table has been imported as dbo.customers.

Following the same procedure, import the three other tables (restaurants, ratings and restaurant_cuisines) in the same manner.

1.2 Primary and Foreign key Constraints:

```

12 -- Add primary key constraint to Restaurant table
13 ALTER TABLE restaurants
14 ADD PRIMARY KEY (Restaurant_ID);
15
16 -- Add primary key constraint to Consumers table
17 ALTER TABLE consumers
18 ADD PRIMARY KEY (Consumer_ID);
19
20 -- Add primary key constraint to Ratings table
21 ALTER TABLE ratings
22 ADD CONSTRAINT PK_ratings PRIMARY KEY (Consumer_ID, Restaurant_ID);
23
24 -- Add foreign key constraint to Ratings table referencing Consumers table
25 ALTER TABLE ratings
26 ADD CONSTRAINT FK_Consumer_ID FOREIGN KEY (Consumer_ID) REFERENCES consumers(Consumer_ID);
27
28 -- Add foreign key constraint to Ratings table referencing Restaurant table
29 ALTER TABLE ratings
30 ADD CONSTRAINT FK_Restaurant_ID FOREIGN KEY (Restaurant_ID) REFERENCES restaurants(Restaurant_ID);
31
32 -- Add primary key constraint to Restaurant_Cuisines table
33 ALTER TABLE restaurant_cuisines
34 ADD CONSTRAINT PK_restaurant_cuisines PRIMARY KEY (Restaurant_ID, Cuisine);
35
36 -- Add foreign key constraint to Restaurant_Cuisines table referencing Restaurant table
37 ALTER TABLE restaurant_cuisines
38 ADD CONSTRAINT FK_Restaurant_ID_Cuisines FOREIGN KEY (Restaurant_ID) REFERENCES restaurants(Restaurant_ID);
39

```

Fig 1.2.1

Primary and Foreign key Constraints

Creating relationships among tables is an essential aspect of database design, and it involves defining the primary keys and foreign keys in the tables. A primary key is a unique identifier for a record in a table, while a foreign key is a field in one table that refers to the primary key in another table.

To create a relationship between two tables, you need to specify the foreign key in the child table that refers to the primary key in the parent table. This is typically done using the ALTER TABLE statement in SQL. Once the relationship is established, you can use it to retrieve data from multiple tables using JOIN queries.

With these ALTER TABLE commands, primary key constraints are added to the Restaurants, Consumers, Ratings, and Restaurant_Cuisines tables. Foreign key constraints are also added to ensure referential integrity between the Ratings table and the Consumers and Restaurant tables, and between the Restaurant_Cuisines table and the Restaurant table.

It's good to know that you only need to run the ALTER queries once to create the relationships among the tables in your database. You can visualize the database schema under the database diagram in the SQL server GUI interface to ensure that the relationships are correctly established.



Fig 1.2.2

Database Diagram

In a relational database, tables are related to each other based on common fields or columns. The "Restaurants" table serves as the main repository for restaurant information, with the "Restaurant_ID" column acting as the primary key, ensuring each restaurant has a unique identifier within the table. This "Restaurant_ID" column is referenced in the "Ratings" table as a foreign key, establishing a relationship between restaurants and ratings. Additionally, the "Ratings" table also references the "Consumers" table

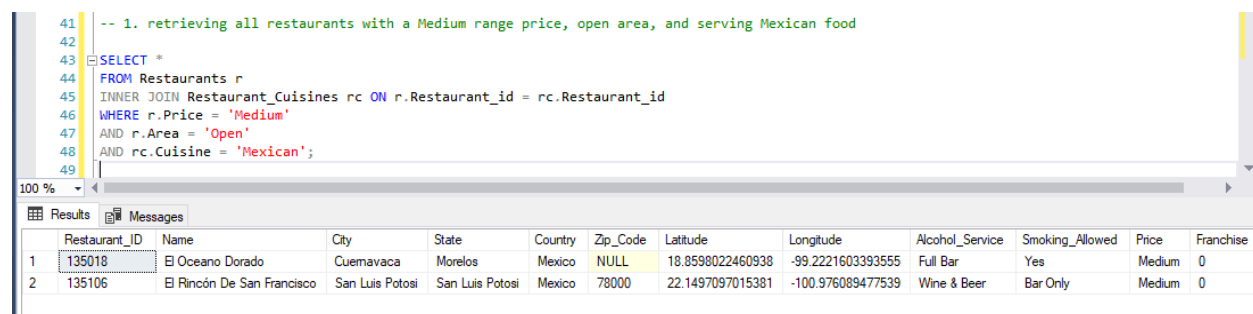
through the "Consumer_ID" foreign key, indicating which consumers provided ratings for various restaurants.

Furthermore, the "Restaurant Cuisines" table is linked to the "Restaurants" table via the "Restaurant_ID" column. This relationship signifies the types of cuisines offered by each restaurant.

It's crucial to choose primary keys that have unique values and are not null to ensure the integrity and consistency of your data. By using primary and foreign keys correctly, you can maintain the relationships between tables in your database, allowing for accurate and reliable data management. These relationships enable the retrieval of interconnected data using JOIN queries, facilitating comprehensive analysis and reporting across multiple tables within the database.

Part 2:

Q1. To retrieve all restaurants with a medium range price, open area, and serving Mexican food, you can use the following SQL query:



```
-- 1. retrieving all restaurants with a Medium range price, open area, and serving Mexican food
SELECT *
FROM Restaurants r
INNER JOIN Restaurant_Cuisines rc ON r.Restaurant_id = rc.Restaurant_id
WHERE r.Price = 'Medium'
AND r.Area = 'Open'
AND rc.Cuisine = 'Mexican';
```

	Restaurant_ID	Name	City	State	Country	Zip_Code	Latitude	Longitude	Alcohol_Service	Smoking_Allowed	Price	Franchise
1	135018	El Oceano Dorado	Cuernavaca	Morelos	Mexico	NULL	18.8598022460938	-99.2221603393555	Full Bar	Yes	Medium	0
2	135106	El Rincón De San Francisco	San Luis Potosi	San Luis Potosi	Mexico	78000	22.1497097015381	-100.976089477539	Wine & Beer	Bar Only	Medium	0

Query Explanation:

- **SELECT * FROM Restaurants r INNER JOIN Restaurant_Cuisines rc ON r.Restaurant_id = rc.Restaurant_id:**
 - This part of the query selects all columns from the 'Restaurants' table and joins it with the 'Restaurant_Cuisines' table based on the 'Restaurant_id' column. This join ensures that we get restaurants serving Mexican cuisine.

- **WHERE r.Price = 'Medium' AND r.Area = 'Open' AND rc.Cuisine = 'Mexican':**
 - This part of the query filters the results to include only restaurants with a medium range price, open area, and serving Mexican food.
 - **r.Price = 'Medium'** ensures that only restaurants with a Medium price range are included.
 - **r.Area = 'Open'** filters the results to include only restaurants with an open area.
 - **rc.Cuisine = 'Mexican'** ensures that only restaurants serving Mexican cuisine are included.

Result Analysis:

- The query successfully retrieves two restaurants (El Oceano Dorado and El Rincón De San Francisco) with a medium range price, open area, and serving Mexican food.
- Both restaurants are located in Mexico.
- Each restaurant has its unique alcohol service, smoking policy, and parking availability.
- This query helps identify suitable restaurants meeting specific criteria for potential customers looking for Mexican cuisine in an open setting within a medium price range.

Q2. To retrieve the total number of restaurants with an overall rating of 1 and serving Mexican food, and comparing it with restaurants of same ratings serving Italian food:


```
51 -- Q2. retrieving the total number of restaurants with an overall rating of 1 and serving Mexican food
52 SELECT COUNT(*) AS Total_Mexican_Restaurants_Rating_1
53 FROM Restaurants r
54 INNER JOIN Ratings ra ON r.Restaurant_id = ra.Restaurant_id
55 INNER JOIN Restaurant_Cuisines rc ON r.Restaurant_id = rc.Restaurant_id
56 WHERE ra.Overall_Rating = 1
57 AND rc.Cuisine = 'Mexican';
58
59 -- comparing the results with the total number of restaurants with an overall rating of 1 serving Italian food
60 SELECT COUNT(*) AS Total_Italian_Restaurants_Rating_1
61 FROM Restaurants r
62 INNER JOIN Ratings ra ON r.Restaurant_id = ra.Restaurant_id
63 INNER JOIN Restaurant_Cuisines rc ON r.Restaurant_id = rc.Restaurant_id
64 WHERE ra.Overall_Rating = 1
65 AND rc.Cuisine = 'Italian';
66
```

100 %

Results Messages

Total_Mexican_Restaurants_Rating_1	
1	87

Total_Italian_Restaurants_Rating_1	
1	11

- Total number of restaurants with an overall rating of 1 serving Mexican food: 87
- Total number of restaurants with an overall rating of 1 serving Italian food: 11

This analysis indicates a significant difference in the number of restaurants with an overall rating of 1 between Mexican and Italian cuisines. Here are some possible explanations for this difference:

- **Preference:** It's possible that consumers have a higher expectation or preference for Mexican cuisine compared to Italian cuisine in this dataset. This could lead to a higher number of low-rated Mexican restaurants compared to Italian restaurants.
- **Quality Variation:** The quality and standards of Mexican and Italian restaurants in the dataset might vary significantly. It's possible that there are more low-quality Mexican restaurants compared to Italian restaurants, resulting in a higher number of low ratings.
- **Sample Size:** The dataset might have a larger number of Mexican restaurants compared to Italian restaurants, leading to more instances of low-rated Mexican restaurants.
- **Consumer Expectations:** Consumers might have different expectations or standards for Mexican and Italian cuisines, leading to different rating distributions.

- **Other Factors:** There could be other factors such as location, pricing, service quality, etc., that influence the ratings and contribute to the observed difference.

Q3. To calculate the average age of consumers who have given a 0 rating to the 'Service_rating' column, you can use the following SQL query:

```

67 -- Q3. calculating the average age of consumers who have given a 0 rating to the 'Service_rating'
68 SELECT ROUND(AVG(c.Age), 0) AS average_age
69
70 FROM consumers c
71
72 JOIN ratings r ON c.Consumer_id = r.Consumer_id
73
74 WHERE r.Service_Rating = 0;
75

```

100 %

Results Messages

	average_age
1	26

- **SELECT ROUND(AVG(c.Age), 0) AS Average_Age:**
 - This part of the query calculates the average age of consumers who have given a 0 rating to the 'Service_rating' column and rounds off the result to the nearest integer.
 - **AVG(c.Age)** calculates the average age of consumers by taking the average of the 'Age' column in the 'Consumers' table and in this case the result is 26.
 - **ROUND(..., 0)** rounds off the calculated average to the nearest integer. The second argument '0' specifies that we want to round to 0 decimal places.
 - **AS Average_Age** aliases the result column as 'Average_Age' for easier reference.
- **FROM Consumers c INNER JOIN Ratings ra ON c.Consumer_id = ra.Consumer_id:**
 - This part of the query specifies the tables involved and how they are joined.
 - **Consumers c** aliases the 'Consumers' table as 'c'.
 - **Ratings ra** aliases the 'Ratings' table as 'ra'.

- **ON c.Consumer_id = ra.Consumer_id** specifies the join condition, linking records in the 'Consumers' table with those in the 'Ratings' table based on the 'Consumer_id' column.
- **WHERE ra.Service_Rating = 0:**
 - This part of the query filters the rows to include only those where the 'Service_Rating' column in the 'Ratings' table is equal to 0.
 - It ensures that we're only considering ratings where the service was rated 0.

Q4. To retrieve the restaurants ranked by the youngest consumer along with the food rating given by that customer to the restaurant, sorted by food rating from high to low, the following SQL query can be use:

```

78 -- Q4. Retrieving the restaurants ranked by the youngest consumer along with the food rating
79 -- by that customer to the restaurant, sorted by food rating from high to low
80 SELECT
81     r.Name AS Restaurant_Name,
82     MIN(c.Age) AS Youngest_Consumer_Age,
83     ra.Food_Rating
84 FROM
85     restaurants r
86 JOIN
87     ratings ra ON r.Restaurant_id = ra.Restaurant_id
88 JOIN
89     consumers c ON ra.Consumer_id = c.Consumer_id
90 GROUP BY
91     r.Restaurant_id, r.Name, ra.Food_Rating
92 ORDER BY
93     ra.Food_Rating DESC;
94

```

100 %

Results Messages

	Restaurant_Name	Youngest_Consumer_Age	Food_Rating
1	Puesto de Gorditas	23	2
2	Cafe Ambar	23	2
3	Church's	21	2
4	Cafe Chaires	22	2
5	McDonalds Centro	20	2
6	Gorditas Doña Tota	23	2
7	Tacos De Barbacoa Enfrente Del Tec	23	2
8	Hamburguesas La Perica	21	2
9	Pollo Frito Buenos Aires	23	2
10	Camitas Mata	23	2
11	La Perica Hamburguesa	23	2

✓ Query executed successfully. | DESKTOP-MA56CTA (16.0 RTM) | DESKTC

- **SELECT r.Name AS Restaurant_Name, MIN(c.Age) AS Youngest_Consumer_Age, ra.Food_Rating:**
 - This part of the query selects three columns: the restaurant name (aliased as Restaurant_Name), the minimum age of consumers (to find the youngest consumer) for each restaurant (aliased as Youngest_Consumer_Age), and the food rating given by that consumer to the restaurant.
- **FROM restaurants r JOIN ratings ra ON r.Restaurant_id = ra.Restaurant_id JOIN consumers c ON ra.Consumer_id = c.Consumer_id:**
 - This part of the query specifies the tables involved in the query and how they are joined. It joins the 'restaurants' table with the 'ratings' table based on the 'Restaurant_id', and then joins the 'ratings' table with the 'consumers' table based on the 'Consumer_id'.
- **GROUP BY r.Restaurant_id, r.Name, ra.Food_Rating:**
 - This part of the query groups the results by 'Restaurant_id', 'Name' (restaurant name), and 'Food_Rating'. It ensures that each group represents a unique combination of restaurant and food rating.
- **ORDER BY ra.Food_Rating DESC:**
 - This part of the query orders the results by food rating in descending order, meaning restaurants with higher food ratings appear first.

Results Analysis:

- The results show each restaurant's name along with the age of the youngest consumer who rated the restaurant and the food rating given by that consumer.
- The food ratings range from 0 to 2, with 2 being the highest rating and 0 being the lowest.
- The restaurants are listed in descending order of food rating, meaning those with higher ratings appear first. If two restaurants

have the same food rating, they are ordered by the age of the youngest consumer, with younger consumers listed first.

Q5. Writing a stored procedure for the query given as:

Update the Service_rating of all restaurants to '2' if they have parking available

```
96 -- Q5. Writing a stored procedure for the query given as:
97 -- Update the Service_rating of all restaurants to '2' if they have parking available
98
99 CREATE PROCEDURE UpdateServiceRatingWithParking
100 AS
101 BEGIN
102     SET NOCOUNT ON;
103     -- Updating Service_Rating for restaurants with parking available
104     UPDATE ratings
105     SET Service_Rating = '2'
106     WHERE Restaurant_id IN (
107         SELECT r.Restaurant_id
108         FROM restaurants r
109         WHERE r.Parking IN ('yes', 'public')
110     );
111 END;
112
```

100 %

Messages

Commands completed successfully.

Completion time: 2024-04-22T14:29:02.9637021+01:00

This stored procedure, named "UpdateServiceRatingWithParking," updates the Service_Rating of all restaurants to '2' if they have parking available. Breaking down the procedure:

- **CREATE PROCEDURE UpdateServiceRatingWithParking:**
 - This initiates the creation of a stored procedure named "UpdateServiceRatingWithParking."
- **AS BEGIN:**
 - Begins the definition of the stored procedure.
- **SET NOCOUNT ON:**
 - This is used to suppress the message indicating the number of rows affected by the SQL statements inside the stored procedure. It's often used to reduce network traffic when the number of rows affected is not needed.

- **UPDATE ratings SET Service_Rating = '2' WHERE Restaurant_id IN (SELECT r.Restaurant_id FROM restaurants r WHERE r.Parking IN ('yes', 'public')):**
 - This is the main logic of the stored procedure.
 - It updates the Service_Rating column in the "ratings" table to '2' for all restaurants that have parking available.
 - The WHERE clause filters the restaurants based on their parking availability. It checks if the Parking column in the "restaurants" table contains values 'yes' or 'public'. If so, it retrieves the corresponding Restaurant_id.
- **END:**
 - Marks the end of the stored procedure definition.

Q6. The Four Queries

Query 1: The average overall rating for restaurants that serve Mexican cuisine and have a price level of 'Medium'.

```

114  -- Query 1: Find the average overall rating for restaurants that serve Mexican cuisine
115  -- and have a price level of 'Medium'.
116
117  SELECT AVG(ra.Overall_Rating) AS Avg_Overall_Rating
118  FROM Ratings ra
119  WHERE ra.Restaurant_id IN (
120      SELECT r.Restaurant_id
121      FROM Restaurants r
122      WHERE r.Price = 'Medium'
123      AND r.Restaurant_id IN (
124          SELECT rc.Restaurant_id
125          FROM Restaurant_Cuisines rc
126          WHERE rc.Cuisine = 'Mexican'
127      )
128  );
129

```

100 %

Results Messages

	Avg_Overall_Rating
1	1

- This query calculates the average overall rating for restaurants that serve Mexican cuisine and have a price level of 'Medium'.
- It uses nested queries with the IN operator to filter restaurants based on the specified criteria.

- The outer query calculates the average overall rating for the filtered restaurants.

Query 2: List restaurants with the highest food rating.

```

130  -- Query 2: List restaurants with the highest food rating.
131  SELECT Top 5 r.Name AS Restaurant_Name, MAX(ra.Food_Rating) AS Max_Food_Rating
132  FROM Restaurants r
133  JOIN Ratings ra ON r.Restaurant_id = ra.Restaurant_id
134  GROUP BY r.Name
135  ORDER BY Max_Food_Rating DESC
136  ;
137

```

	Restaurant_Name	Max_Food_Rating
1	Cafe Punta Del Cielo	2
2	Cafe Chaires	2
3	Cafe Ambar	2
4	Cabana Huasteca	2
5	Arrachela Grill	2

- This query retrieves the top 5 restaurants with the highest food rating.
- It uses GROUP BY to group the results by restaurant name and calculates the maximum food rating for each restaurant.
- The results are sorted in descending order of food rating, and the top 5 restaurants are selected.

Query 3: Finding the number of consumers who have rated a restaurant's service higher than its food.

```

138  -- Query 3: Find the number of consumers who have rated a restaurant's service higher than its food.
139  SELECT COUNT(*) AS Num_Consumers
140  FROM (
141      SELECT ra.Consumer_id
142      FROM Ratings ra
143      WHERE ra.Service_Rating > ra.Food_Rating
144      GROUP BY ra.Consumer_id
145  ) AS Subquery;
146

```

	Num_Consumers
1	71

- This query counts the number of consumers who have rated a restaurant's service higher than its food.

- It uses a nested query with EXISTS to filter ratings where the service rating is higher than the food rating.
- The outer query counts the distinct consumers from the filtered ratings.

System functions

```

237 -- System functions
238 SELECT Name, City, State, LEN(Name) AS Name_Length
239 FROM restaurants
240 SELECT Name, City, State
241 FROM restaurants
242 WHERE Restaurant_id IN (
243     SELECT Restaurant_id
244     FROM ratings
245     GROUP BY Restaurant_id
246     HAVING AVG(Overall_Rating) < 3
247 );
248

```

100 %

Results Messages

	Name	City	State	Name_Length
1	Puesto de Gorditas	Ciudad Victoria	Tamaulipas	18
2	Cafe Ambar	Ciudad Victoria	Tamaulipas	10
3	Church's	Ciudad Victoria	Tamaulipas	8
4	Cafe Chaires	San Luis Potosi	San Luis Potosi	12
5	McDonalds Centro	Cuernavaca	Morelos	16
6	Gorditas Doña Tota	Ciudad Victoria	Tamaulipas	18
7	Tacos De Barbacoa Enfrente Del Tec	Ciudad Victoria	Tamaulipas	34
8	Hamburguesas La Perica	Ciudad Victoria	Tamaulipas	22

	Name	City	State
1	Puesto de Gorditas	Ciudad Victoria	Tamaulipas
2	Cafe Ambar	Ciudad Victoria	Tamaulipas

✓ Query executed successfully.

The main purpose of this query is to select the names, cities, and states of restaurants whose average overall rating is less than 3. Additionally, it calculates the length of the restaurant names. The subquery is used to filter restaurants based on their average ratings, and the outer query fetches the desired columns for those filtered restaurants.

Use of GROUP BY, HAVING and ORDER BY clauses

This query utilizes the GROUP BY, HAVING, and ORDER BY clauses to retrieve information about restaurant prices where the count of restaurants with each price is greater than 10.

```
160  -- Use of GROUP BY, HAVING and ORDER BY clauses
161
162  SELECT COUNT(*), Price
163
164  FROM restaurants
165
166  GROUP BY Price
167
168  HAVING COUNT(*) > 10;
169
```

100 %

Results Messages

	(No column name)	Price
1	25	High
2	45	Low
3	60	Medium

This query retrieves the count of restaurants for each price category where the count is greater than 10, and it orders the result set by price. It's useful for identifying price categories with a significant number of restaurants.

Conclusion:

The database design solution includes four tables: Restaurants, Consumers, Ratings, and Restaurant_Cuisines. Each table captures specific information related to restaurants, consumers, their ratings, and the cuisines served by restaurants. Primary and foreign key constraints ensure data integrity and establish relationships between tables.

Key functionality provided by the database includes:

1. **Storing Restaurant Information:** The Restaurants table stores details such as name, location, pricing, services, and parking availability of restaurants.
2. **Capturing Consumer Data:** The Consumers table records information about consumers, including their demographics, preferences, and ratings given to restaurants.

3. **Recording Ratings:** The Ratings table links consumers to restaurants and stores their ratings for overall experience, food quality, and service.
4. **Tracking Cuisines:** The Restaurant_Cuisines table associates restaurants with the cuisines they serve, allowing for easy retrieval of restaurants based on cuisine type.
5. **Querying and Analysis:** The database supports various types of queries, including filtering restaurants by specific criteria (e.g., price range, cuisine type), analyzing consumer ratings, and calculating aggregate statistics (e.g., average ratings, count of restaurants).

Overall, the database design solution provides a platform for managing restaurant-related data, facilitating analysis, and assisting in decision-making processes for food service companies.