**USER MANUAL FOR**

**COMMUNITY GARDENING DATABASE**

****

**TABLE OF CONTENTS**

[**The Business**](#_in21guj15wa0) **3**

[**Database Description**](#_a34zunxehv2n) **3**

[**Entity Relationship Diagram**](#_8uqrrgpsl4i9) **3**

[**Relationship Diagram from SQL Server**](#_nh4bghao3zlv) **5**

[**Data Dictionary for X Community Project Database**](#_lvdvrleyxv4i) **6**

[**Tables**](#_23bd1cuvzbe2) **7**

[ConnectionRegistry](#_921voni1geyx) 9

[DailySales](#_gbk6d9yxkbqs) 9

[Equipment](#_yb53m7aeqtr1) 10

[EquipmentRegistry](#_nh2ew37kd0na) 10

[FertilizerRegistry](#_1iaw5nrmmwsz) 11

[Fertilizers](#_kyk29h9j7l71) 11

[Fruits](#_27z8mw23mbqk) 11

[Lands](#_6y1j6rl12sdl) 11

[NutritionType](#_hrqdasydsm1m) 12

[Products](#_zgkn2ek1rv8t) 12

[Residents](#_esqn7ig5r4ai) 13

[Vegitables](#_6c202iutsbjh) 13

[Volunteers](#_cr51gjbl2ifm) 14

[**Stored Procedures**](#_yezv0jr293lq) **16**

[1.Stored Procedure to identify residents who are connected to volunteers with their names](#_dc8kjfd7fvu8) 16

[2. Stored procedure to get the sub-query of the residents based on the city they work for](#_ywceiwwsg3ba) 17

[3. Stored Procedure to combine volunteers in both resident table and volunteer table:](#_wfvkphy1a6hi) 18

[4. Stored Procedure to get volunteer details from the volunteer table based on the location:](#_bm37h1u78iw7) 20

[5. Stored procedure to insert a new resident to the table](#_t78w9beujoyf) 22

[6. Stored Procedure to find out profits per products:](#_5z0awee20uiu) 23

[7.Stored Procedure get the location of resident based on the location](#_e3vo0xzdx836) 24

[**Views:**](#_pxilmre4md83) **25**

[View 1- Profit By Location and Product:](#_ffuwu4r6i73g) 25

[View 2- To identify the Sales in Descending Order](#_h9p5cr7seu08) 26

[View 3- To View Product Total Sales in Descending Order](#_nq5d1uvszyqx) 26

[**Reports**](#_gcgj7ap8l7bx) **27**

[Report1: Sales Report](#_lwp7j7gltdae) 27

[Report2: Sales by Location and Products](#_tqb8gckxt24p) 29

[Report3: Product Sales Profit Information](#_dk5qzmeemgyb) 30

[Report4: Residents and Volunteers Connection Report](#_smqhjo9doe5m) 30

[Report5: Volunteer Information](#_9niqqiynynfw) 31

[**User Defined Function**](#_n9aey7jtz9gl) **31**

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# The Business

X Community Welfare Association, a non-profit organization serving the residents of their community by providing them the opportunity to participate in community gardening and to eat fresh produce, have a healthy lifestyle, good communication with each other etc. It has hired system consultants to create a database to help organize work systems and maximize efficiency for its volunteers and residents.

The requested database will help to keep track of the community garden’s activities. It will track the participating residents, volunteers, lands available in the community area, plantations, details of the produce, health benefits, details of the equipment required for gardening, fertilizers and the favourable weather conditions. The following documents details the design of the requested database.

# Database Description

We, the systems consultants, put forth the idea to create a database information system to assist the X Community Welfare Association. Our proposed database will house the following main entities:

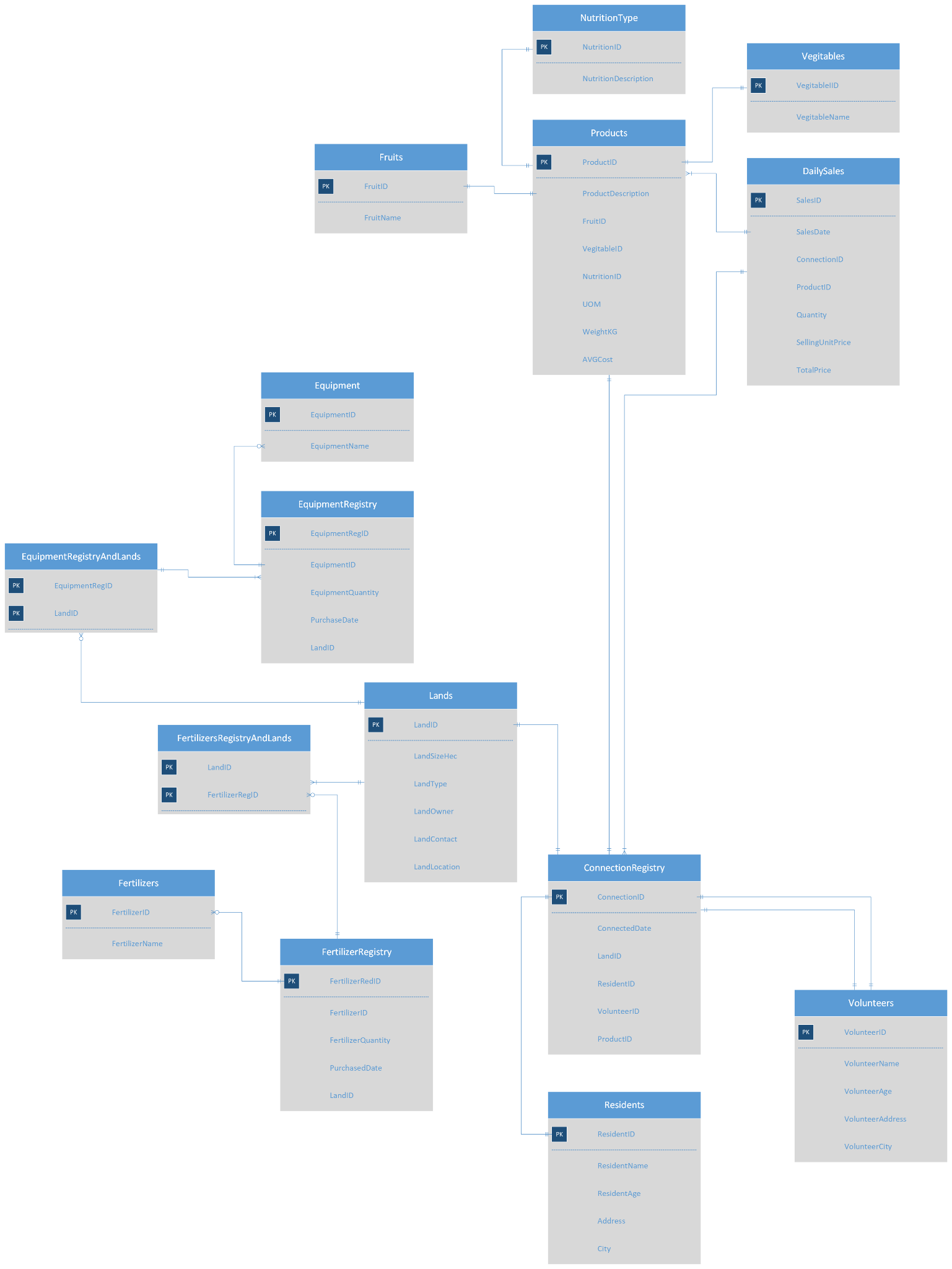
* ConnectionRegistry
* DailySales
* Equipment
* EquipmentRegistry
* FertilizerRegistry
* Fertilizers
* Fruits
* Lands
* Nutrition Type
* Products
* Residents
* Vegetables
* Volunteers

These entities revolve around several key assumptions as noted in the below business rules.

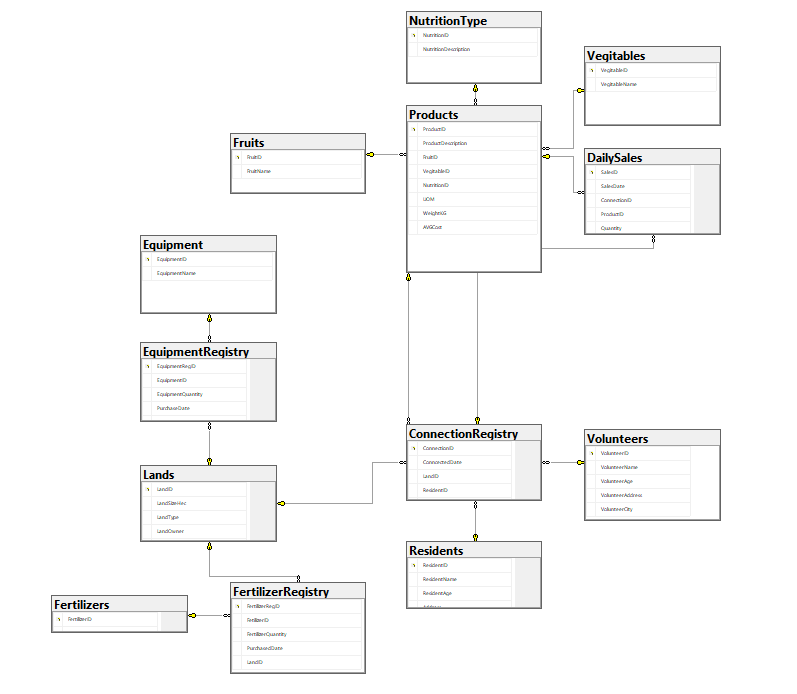
# Entity Relationship Diagram

Entity relationship diagram is a diagram that represents the entities of a database and the relationships among the tables. An ERD will help the user to understand what is in a database. The entity relationship diagram for the Community gardening database and its associated business rules has been given.

* An equipmentregistry can have 1 or more equipment and an equipment can be included in one equipmentregistry.
* A fertilizer can be used for one or more lands and a land may require one or more fertilizers.
* A ConnectionRegistry can have 1 resident and a resident can have 1 connectionregistry.
* A ConnectionRegistry can have only 1 volunteer and a volunteer can have only one connectionregistry.
* A ConnectionRegistry can have only 1 land and a land can have only one connectionregistry.



# Relationship Diagram from SQL Server

****

# Data Dictionary for X Community Project Database

# 

ConnectionRegistry = represents the data of the connection between lands, residents,

volunteers and products through their respective primary key

identifiers.

DailySales = represents the data of the sales of the products obtained from

the gardens.

Equipment = represents the equipment used for gardening purposes.

EquipmentRegistry = represents the data of equipment such as the purchase details,

equipment quantity.

FertilizerRegistry = represents the data of the fertilizers and their purchase details used for gardening.

Fertilizers = represents the details of the fertilizers used.

Fruits = represents the details of the fruits produced during the

gardening.

Lands = represents the data of the lands being used for gardening.

Nutrition Type = represents the data of the nutritions associated with the products.

Products = represents the data of the products that can be obtained through

Gardening.

Residents = represents the data of the residents.

Vegitables = represents the data of the vegetables obtained in gardening

Volunteers = represents the data of the volunteers

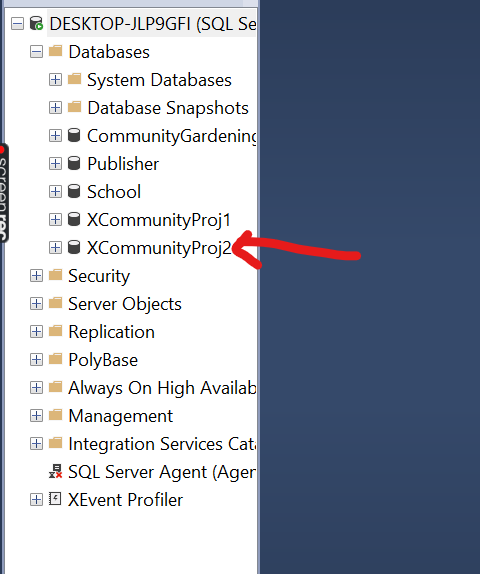
# Tables

Tables of the entities are the building blocks of any database. The following are the entities present in the community gardening database. If you want to know the ddesign of the entities or the database, follow the below steps:

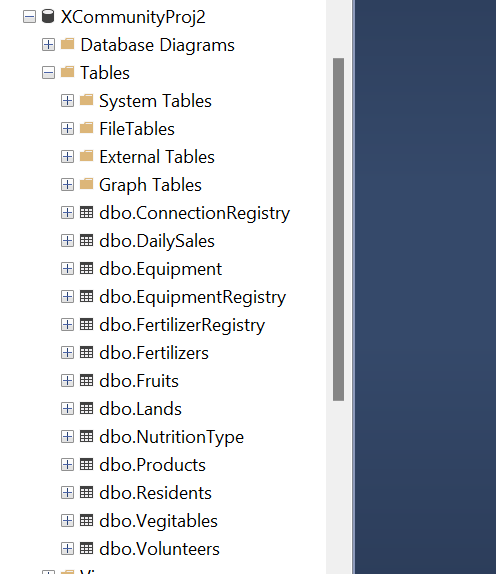
Step 1: Open the SSMS - Sql Server Management Studio



Step 2: Connect to the XCommunityProj2 database



Step 3: To view Tables, Click on tables

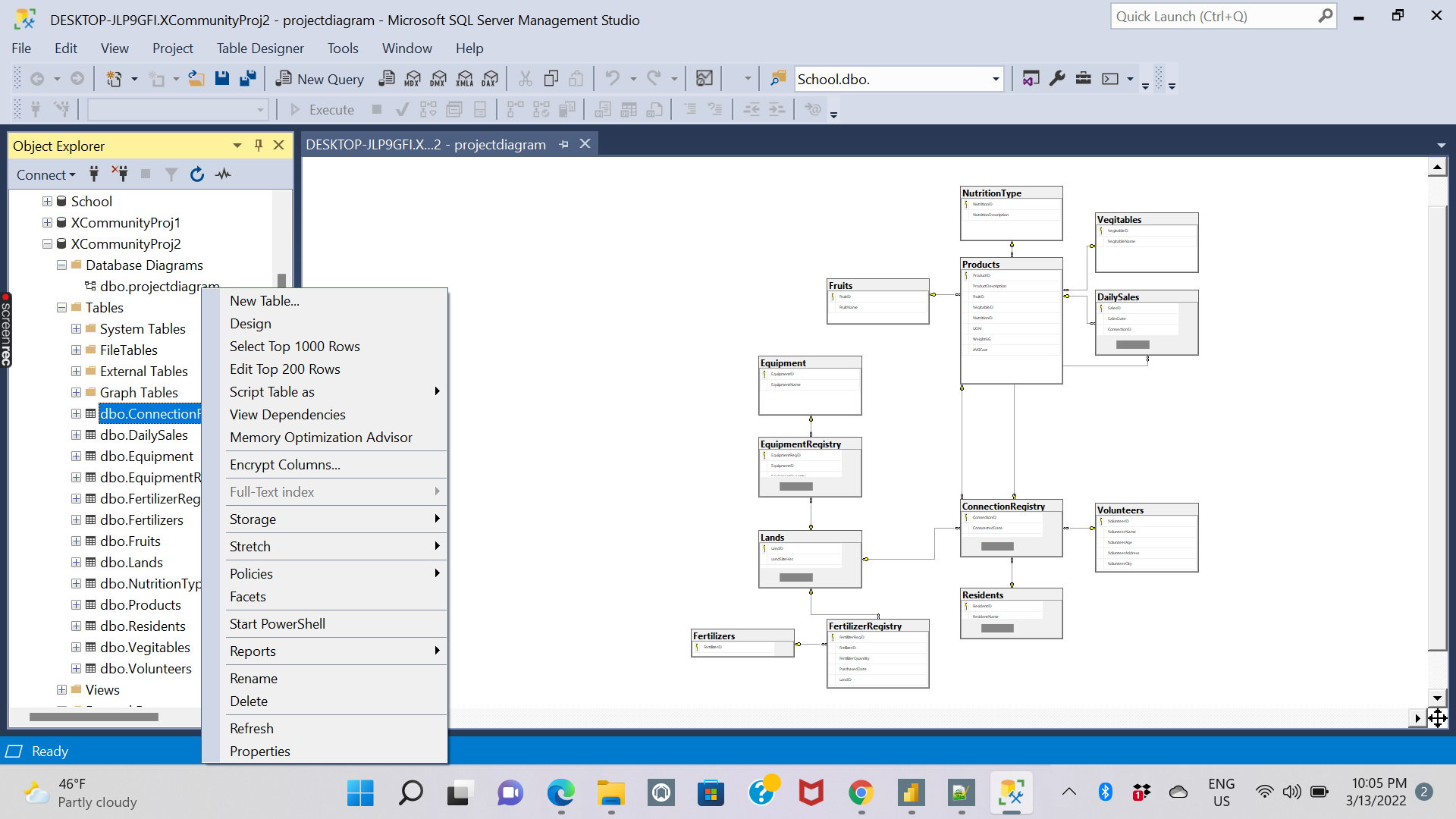


Step 4: To view the SQL relationship diagram, Click on Database diagrams

Then, click on the projectdiagram, you can then view the database diagram as shown below.



Step 5: To know about the structure of a table, right click on a desired table and click on “Design.”



For better understanding, the design of all the tables is provided for the user.

## ConnectionRegistry

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| ConnectionID | PK | Unique identifier for the connection | INT | Y | Unique. |
| ConnectedDate |  | The identifier for the date | DATE | Y |  |
| LandID | FK | The identifier for the Land used for gardening. | INT | Y |  |
| ResidentID | FK | The identifier for the resident | INT | Y |  |
| VolunteerID | FK | The identifier for the volunteer | INT | Y |  |
| ProductID | FK | The identifier for the product | INT | Y |  |

## DailySales

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| SalesID | PK | The identifier for the sales of the products | INT | Y | Unique.  Value. |
| SalesDate |  | The identifier for the date of the sales | DATE | Y |  |
| ConnectionID | FK | The identifier for the connection | INT | Y |  |
| ProductID | FK | The identifier for the product | INT | Y |  |
| Quantity |  | The identifier for the quantity | INT | Y |  |
| SellingUnitPrice |  | The identifier for selling unit price | INT | Y |  |
| TotalPrice |  | The identifier for total price | INT | Y |  |

## Equipment

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| EquipmentID | PK | The identifier for the equipment | INT | Y | Unique  Value. |
| EquipmentName |  | Name of the Equipment | VARCHAR(50) | Y |  |

## EquipmentRegistry

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| EquipmentRegID | PK | The identifier for the ID of the registered equipment | INT | Y | Unique.  Value. |
| EquipmentID | FK | The identifier for the equipment | INT | Y |  |
| EquipmentQuantity |  | The identifier for the quantity of the equipment | INT | Y |  |
| PurchaseDate |  | The identifier for the date purchased | DATE | Y |  |
| LandID | FK | The identifier for the Land | INT | Y |  |

## FertilizerRegistry

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| FertilizerRegID | PK | The identifier for the registry of the fertilizers | INT | Y | Unique.  Value. |
| FertilizerID | FK | The identifier for the ID of the fertilizer | INT | Y |  |
| FertilizerQuantity |  | The identifier for the quantity of the fertilizer purchased | INT | Y |  |
| PurchasedDate |  | The identifier of the date of the fertilizer purchased | DATE | Y |  |
| LandID | FK | The identifier of the land | INT | Y |  |

## Fertilizers

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| FertilizerID | PK | The identifier for the fertilizer | INT | Y | Unique.  Value. |
| FertilizerName |  | The identifier of the name of the fertilizer | VARCHAR(50) | N |  |

## Fruits

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| FruitID | PK | The identifier for the fruit | INT | Y | Unique.  Value. |
| FruitName |  | The identifier of the name of the fruit | VARCHAR(50) | Y |  |

## Lands

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| LandID | PK | The identifier for the land | INT | Y | Unique.  Value. |
| LandSizeHec |  | The identifier for the size of the land in hectares | INT | N |  |
| LandType |  | The identifier for the type of the land | VARCHAR(50) | N |  |
| LandOwner |  | The identifier of the owner of the land | VARCHAR(50) | N |  |
| LandContact |  | The identifier for the contact of the land | INT | N |  |
| LandLocation |  | The identifier for the location of the Land | VARCHAR(50) | N |  |

## NutritionType

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| NutritionID | PK | The identifier for the nutrition | INT | Y | Unique.  Value. |
| NutritionDescription |  | The identifier for the description of the nutrition | VARCHAR(50) | Y |  |

## Products

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| ProductID | PK | The identifier for the ID of the Product | INT | Y | Unique.  Value. |
| ProductDescription |  | The identifier for the description of the product | VARCHAR(50) | N |  |
| FruitID | FK | The identifier for the ID of the Fruit | INT | N |  |
| VegitableID | FK | The identifier for the ID of the vegitable | INT | N |  |
| NutritionID | FK | The identifier for the ID of the nutrition | INT | N |  |
| UOM |  | The identifier for the Unit Of Mass | VARCHAR(50) | N |  |
| WeightKG |  | The identifier for the weight of the Product in Kilogram | INT | N |  |
| AVGCost |  | The identifier for the average cost of the products | INT | N |  |

## Residents

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| ResidentID | PK | The identifier for the resident | INT | Y | Unique |
| ResidentName |  | The identifier for the name of the resident | VARCHAR(50) | N |  |
| ResidentAge |  | The identifier for the age of the resident | INT | N |  |
| Address |  | The identifier for the address of the resident | VARCHAR(50) | N |  |
| City |  | The identifier for the city | VARCHAR(50) | N |  |

## Vegitables

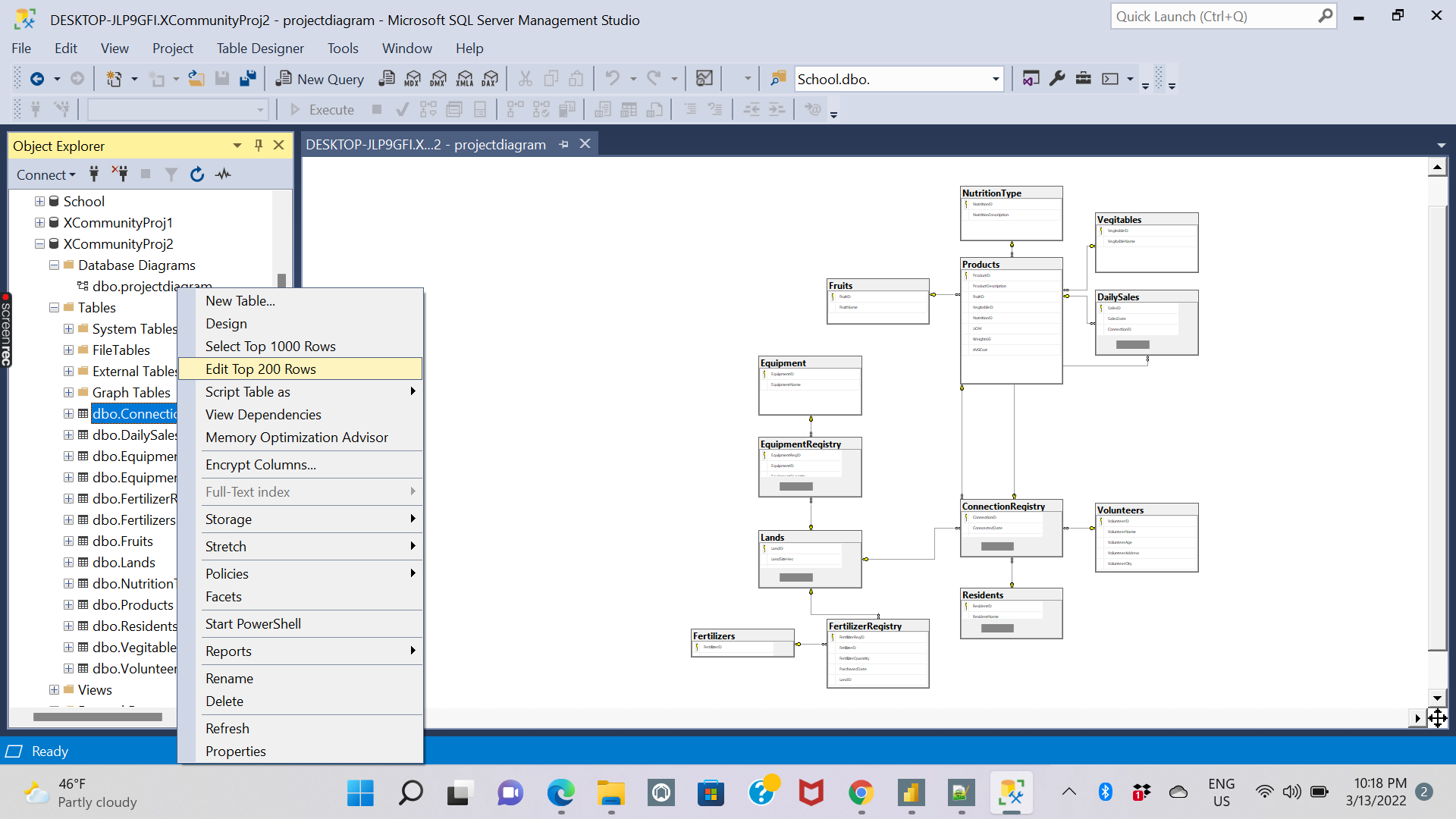
| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| VegitableID | PK | The identifier for the ID of the vegitable | INT | Y |  |
| VegitableName |  | The identifier for the name of the vegitable | VARCHAR(150) | N |  |

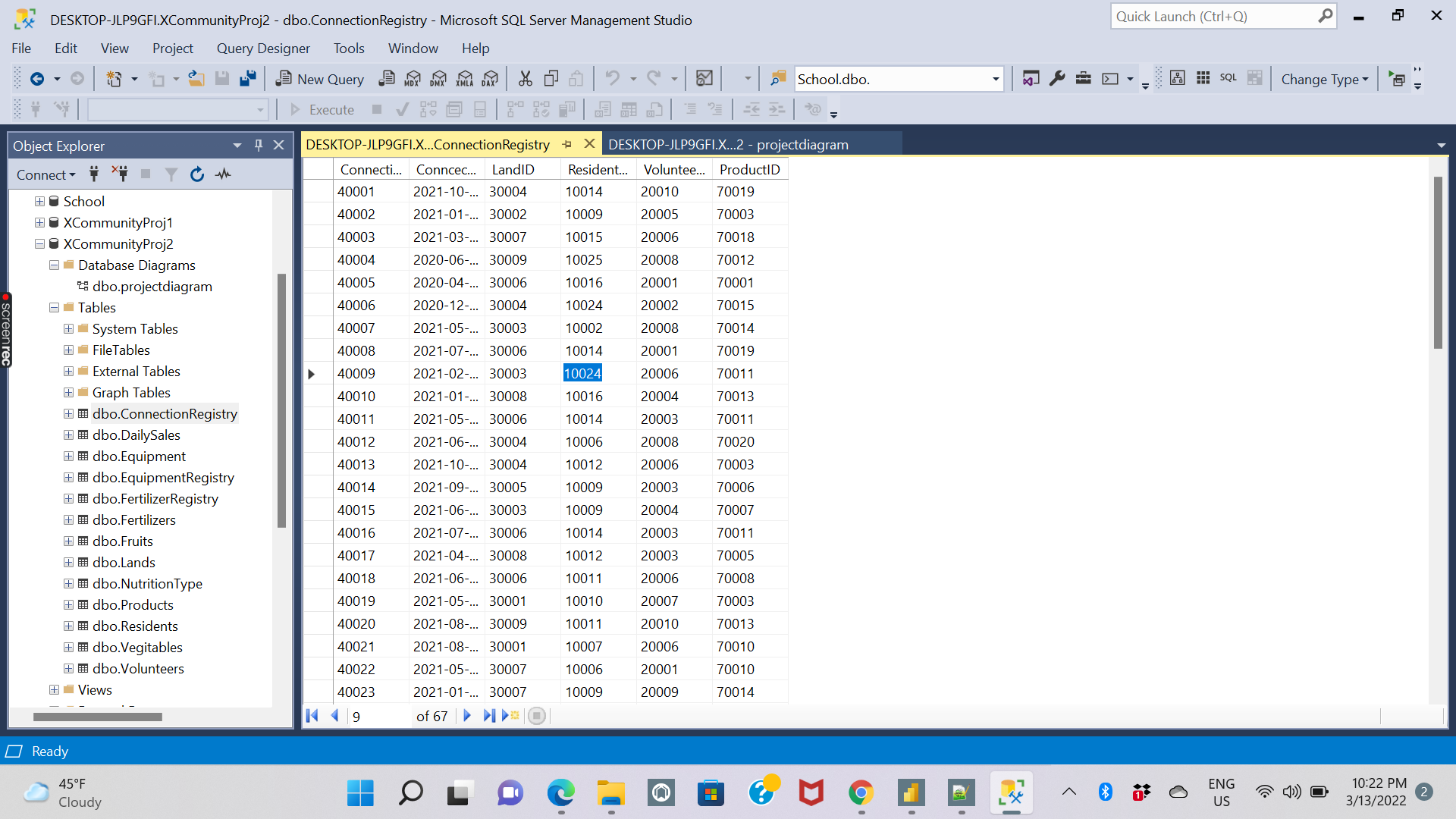
## Volunteers

| **Attribute Name** | **Key Type  (if any)** | **Definition** | **Data Type** | **Required**  **(Y/N)** | **Constraints  (if any)** |
| --- | --- | --- | --- | --- | --- |
| VolunteerID | PK | The identifier for the ID of the volunteer | INT | Y | Unique |
| VolunteerName |  | The identifier for the name of the volunteer | VARCHAR(50) | N |  |
| VolunteerAge |  | The identifier for the age of the volunteer | INT | N |  |
| VolunteerAddress |  | The identifier for the address of the volunteer | VARCHAR(50) | N |  |
| VolunteerCity |  | The identifier for the city of the volunteer | VARCHAR(50) | N |  |

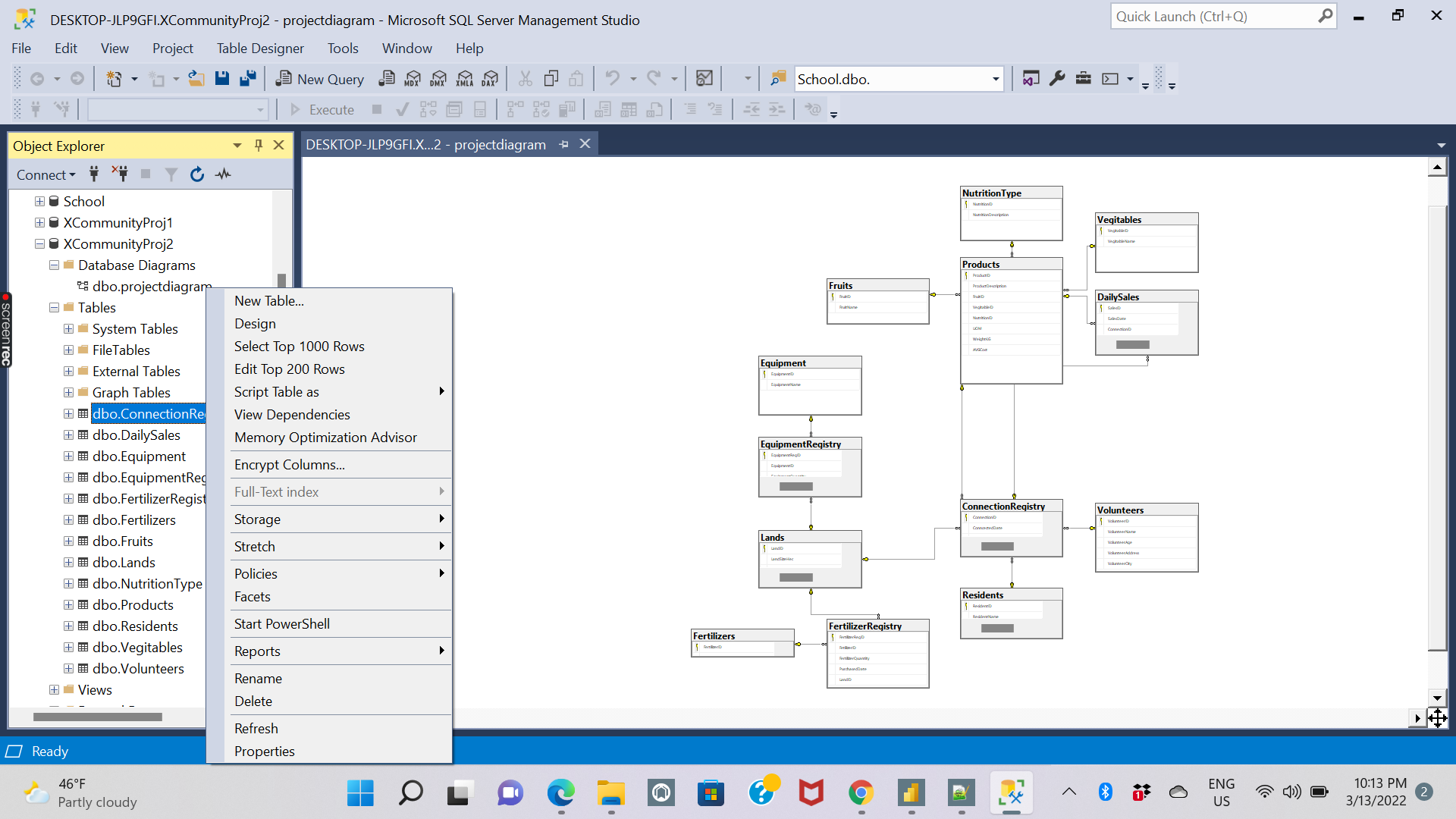
# 

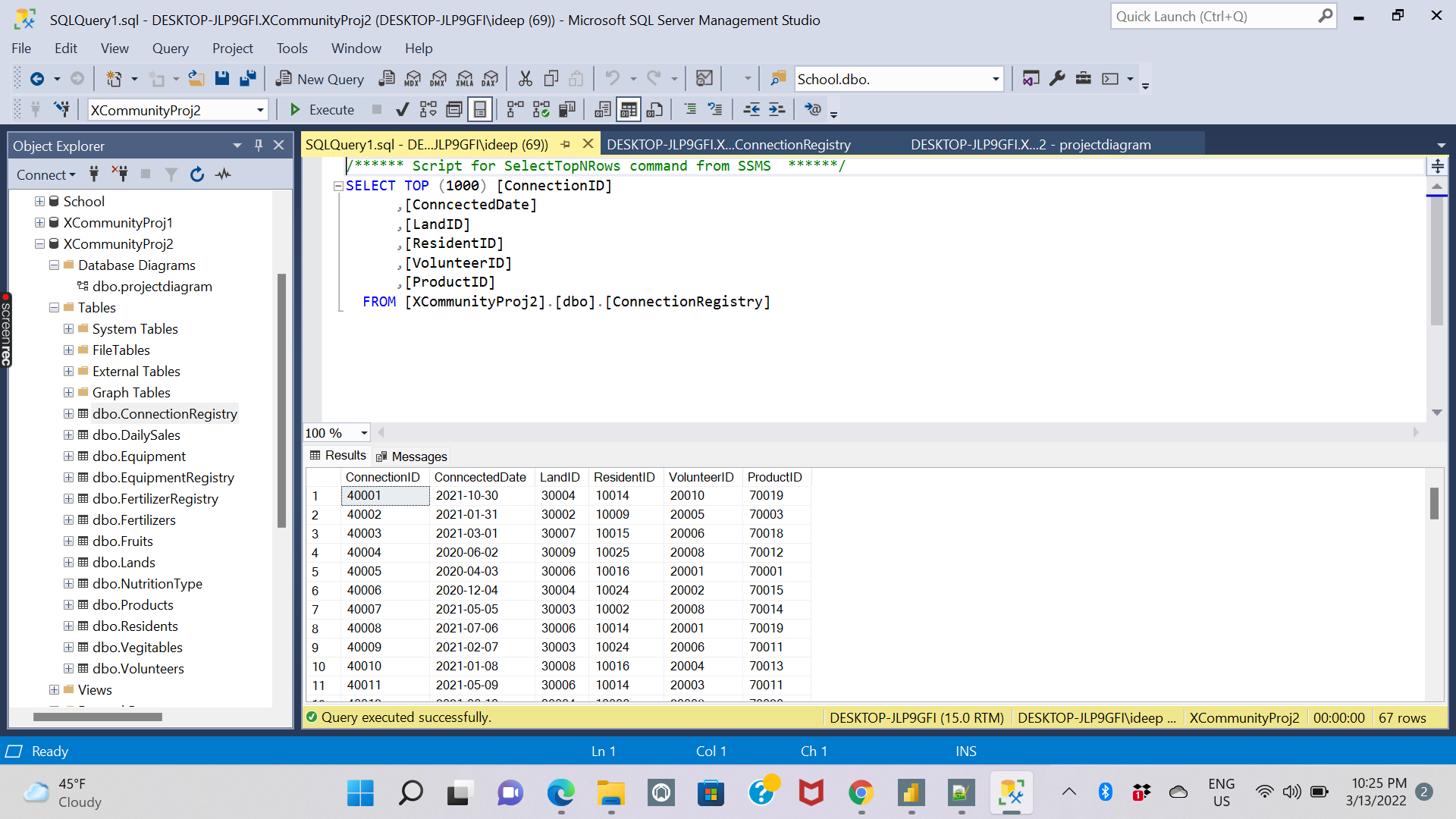
Step 6: To populate the tables with data, right click on a table, click on “Edit Top 200 Rows” and enter the desired data according to the data type mentioned and execute as shown below.





Step 7: To view the data inside an entity or a table, right click on the desired table and click on “Select Top 1000 Rows”- It will generate an automatic select statement and displays the data inside the table as shown below





# Stored Procedures

The user can run various diverse queries on the database and observe the results. For instance, if the user wants to identify all the residents who are connected to the volunteers, we can create stored procedures or the queries. The following is the stored procedure for the above example

## 1.Stored Procedure to identify residents who are connected to volunteers with their names

**Steps to stored procedure to identify residents who are connected to volunteers with their names**

**1.1 Step1 - Run code to identify the residents who are connected to volunteers with their names**

/\*this proc is to identify Residents who are connected to volunteers with their names\*/

CREATE PROC ConnectedResidentsVolunteers AS

SELECT

A.ConncectedDate,A.ResidentID,B.ResidentName,A.VolunteerID,C.VolunteerName

FROM ConnectionRegistry AS A

INNER JOIN Residents AS B ON A.ResidentID =B.ResidentID

INNER JOIN Volunteers AS C ON A.VolunteerID = C.VolunteerID

/\*Please run below seperately\*/

EXEC ConnectedResidentsVolunteers

**1.2 Step 2 - Step by Step process for the code executed**

1. The user should select on the **Execute** button after the below is code is entered

CREATE PROC ConnectedResidentsVolunteers AS

SELECT

A.ConncectedDate,A.ResidentID,B.ResidentName,A.VolunteerID,C.VolunteerName

FROM ConnectionRegistry AS A

INNER JOIN Residents AS B ON A.ResidentID =B.ResidentID

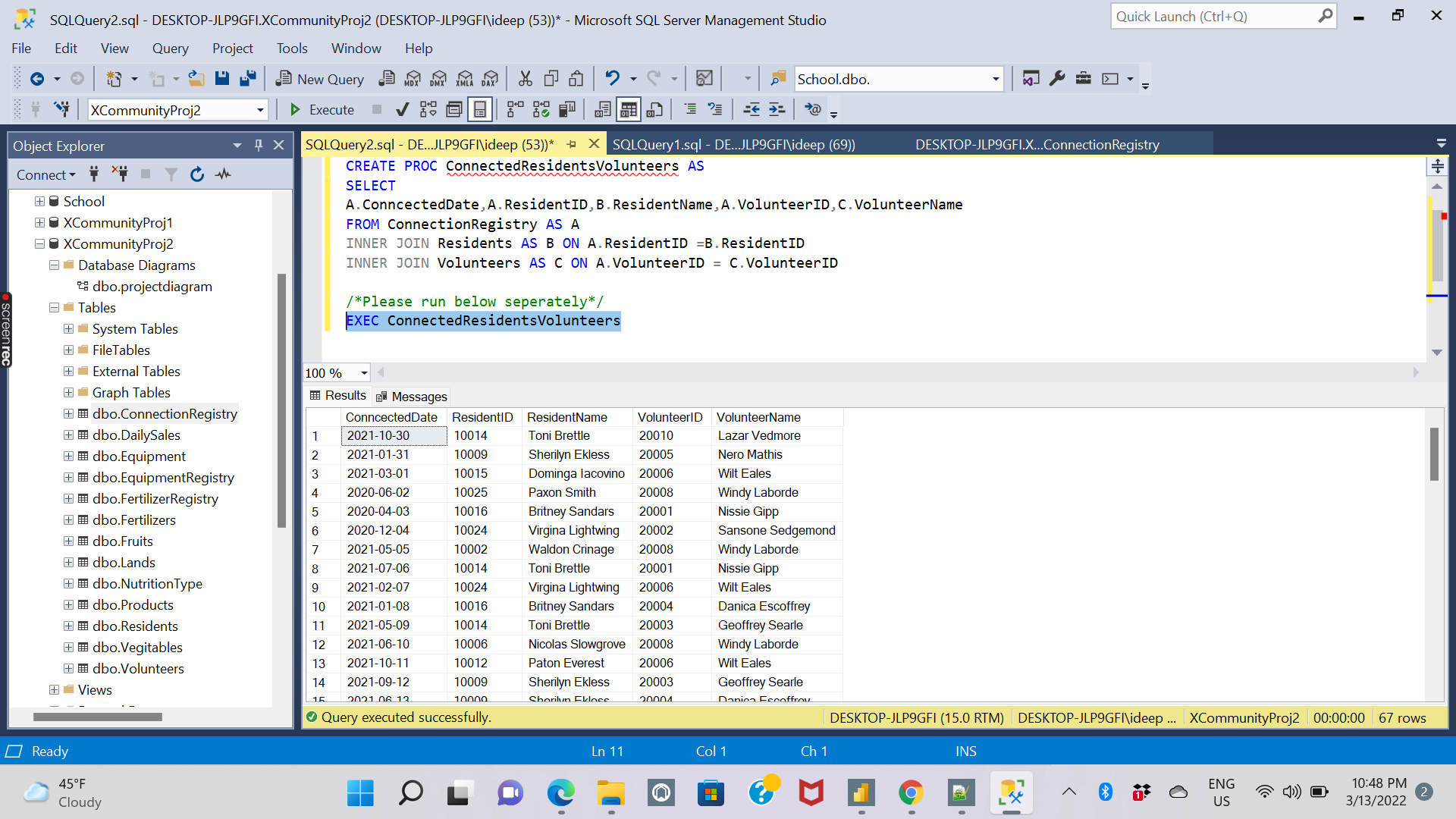
INNER JOIN Volunteers AS C ON A.VolunteerID = C.VolunteerID

1. The user should again select the below code and click on **Execute** to run the below code separately

Once the stored procedure gets executed, then, just running the below statement will retrieve the results.

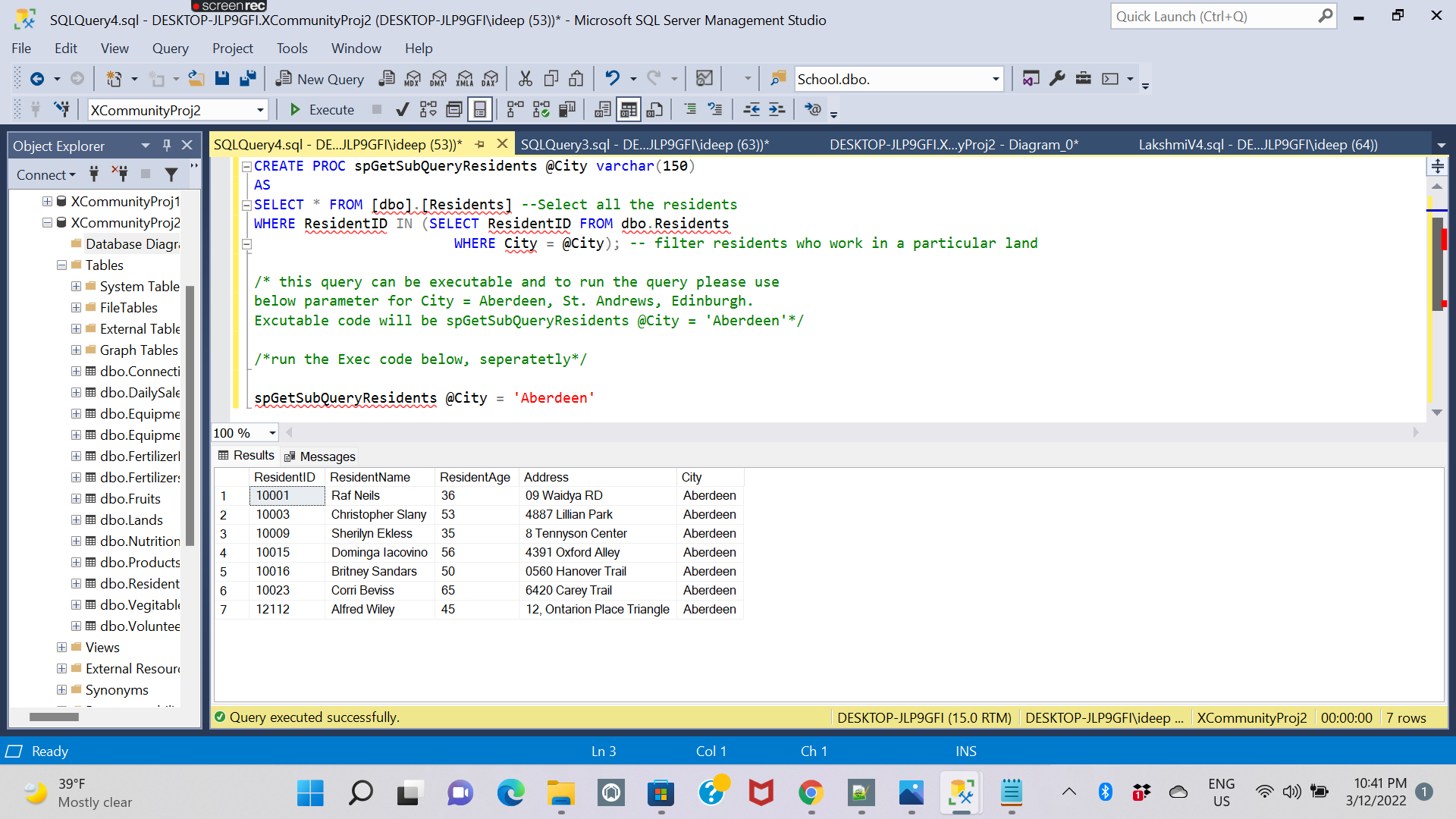
**EXEC ConnectedResidentsVolunteers**

1. After the code is executed the Database retrieved the data of 67 residents those are connected to the volunteers

****

## 2. Stored procedure to get the sub-query of the residents based on the city they work for

**2.1 Step 1- Run code to Stored procedure to get the sub-query of the re sidents based on the city they work for**



/\*Create a stored procedure to get the subquery

of the residents based on the city they work for\*/

CREATE PROC spGetSubQueryResidents @City varchar(150)

AS

SELECT \* FROM [dbo].[Residents] --Select all the residents

WHERE ResidentID IN (SELECT ResidentID FROM dbo.Residents

WHERE City = @City); -- filter residents who work in a particular land

/\* this query can be executable and to run the query please use

below parameter for City = Aberdeen, St. Andrews, Edinburgh.

Excutable code will be spGetSubQueryResidents @City = 'Aberdeen'\*/

/\*run the Exec code below, seperatetly\*/

spGetSubQueryResidents @City = 'Aberdeen'

**2.2 Step 2 - Step by Step process to Stored procedure to get the sub-query of the residents based on the city they work for**

1. The User should select the below code and click on **Execute** button

CREATE PROC spGetSubQueryResidents @City varchar(150)

AS

SELECT \* FROM [dbo].[Residents] --Select all the residents

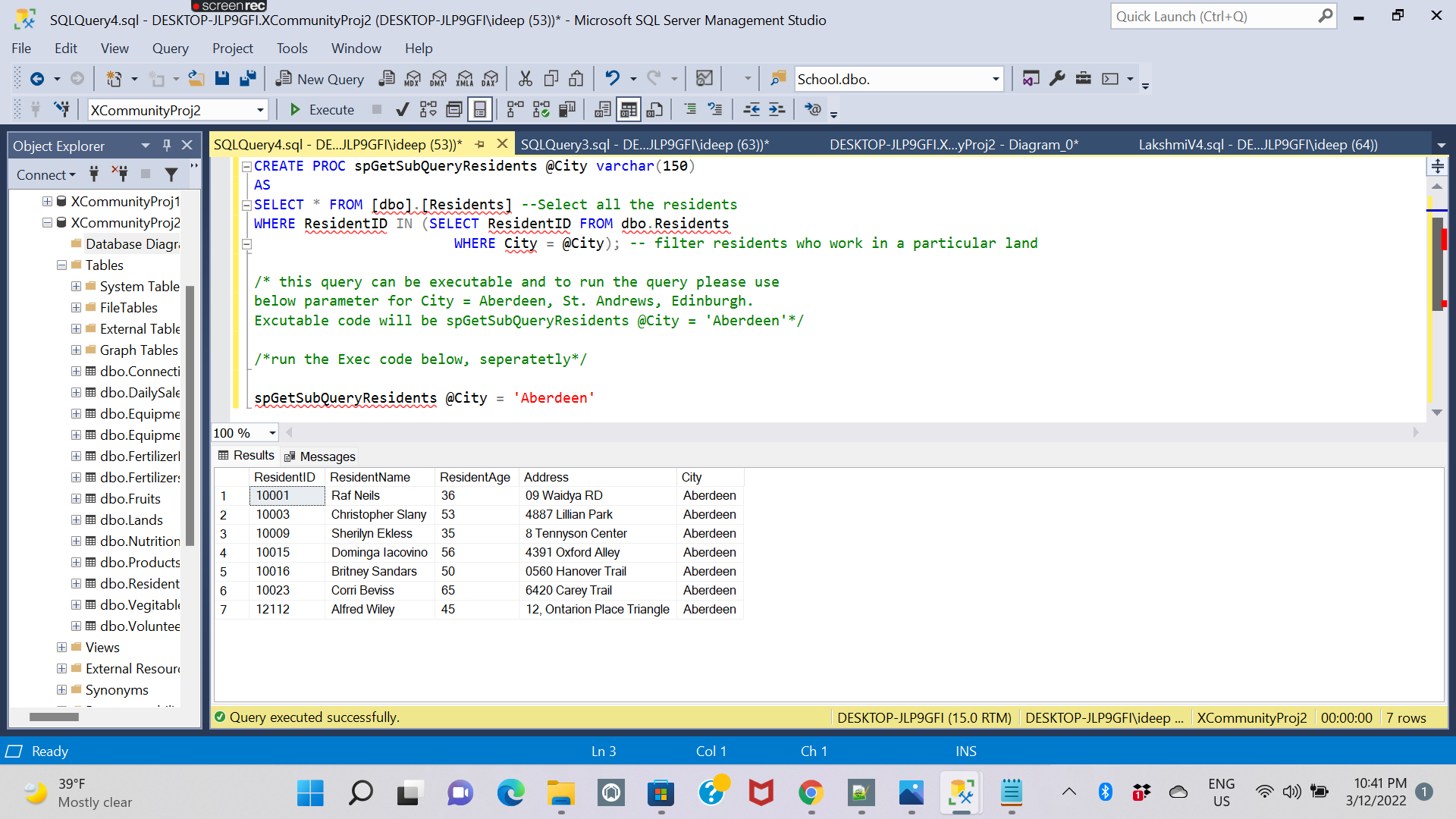
WHERE ResidentID IN (SELECT ResidentID FROM dbo.Residents

WHERE City = @City);

1. The user should select the below code and execute separately

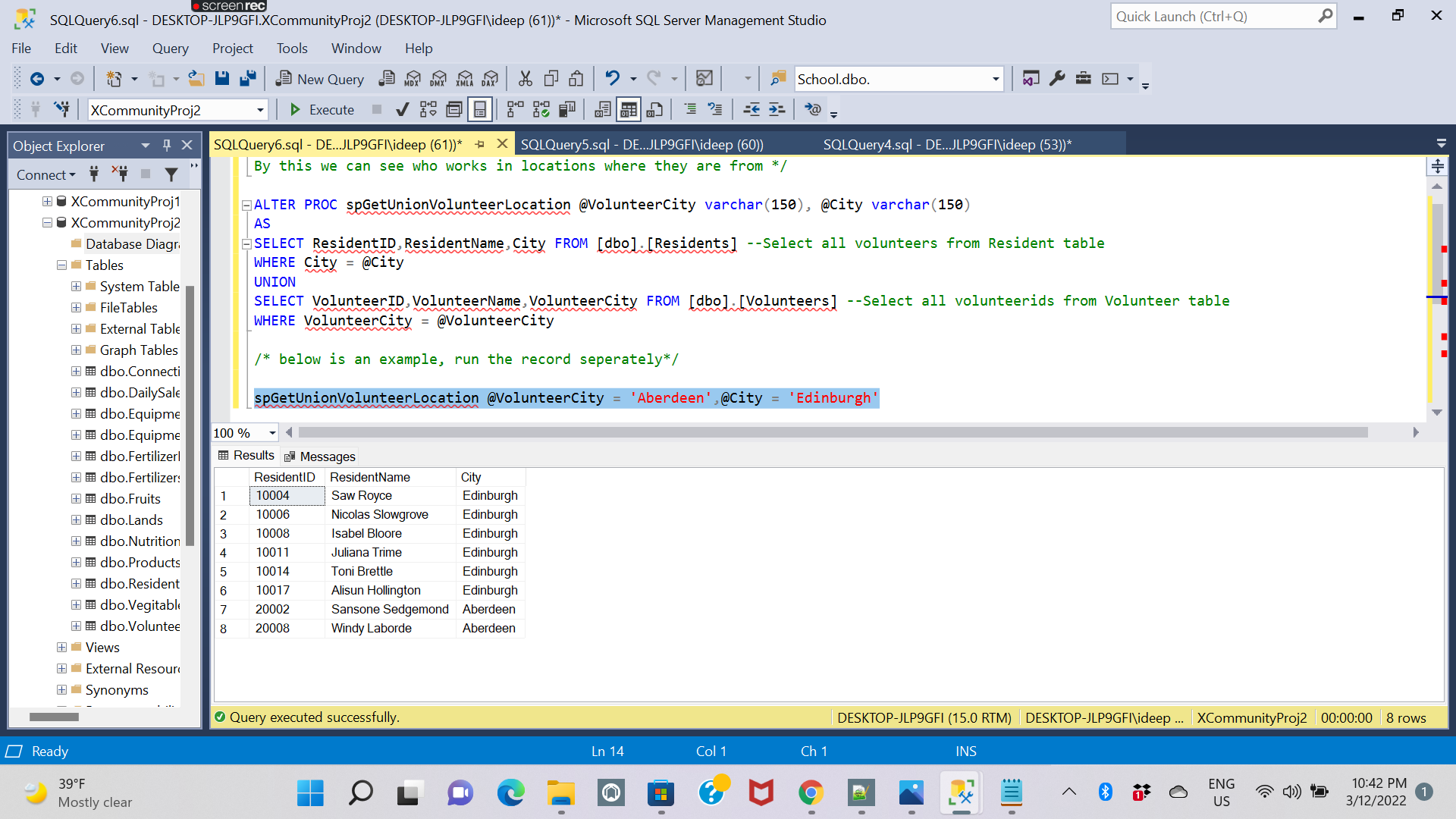
spGetSubQueryResidents @City = 'Aberdeen'

1. After the code is executed the database will announce the data of residents based on the city they worked for as shown in the below image



## 3. Stored Procedure to combine volunteers in both resident table and volunteer table:

**3.1 Run code to Stored Procedure to combine volunteers in both resident table and volunteer table**



/\*combine volunteers in both the resident table and volunteer table.

By this we can see who works in locations where they are from \*/

ALTER PROC spGetUnionVolunteerLocation @VolunteerCity varchar(150), @City varchar(150)

AS

SELECT ResidentID,ResidentName,City FROM [dbo].[Residents] --Select all volunteers from Resident table

WHERE City = @City

UNION

SELECT VolunteerID,VolunteerName,VolunteerCity FROM [dbo].[Volunteers] --Select all volunteerids from Volunteer table

WHERE VolunteerCity = @VolunteerCity

/\* below is an example, run the record seperately\*/

spGetUnionVolunteerLocation @VolunteerCity = 'Aberdeen',@City = 'Edinburgh'

**3.2 Step by step process for the combine volunteers in both resident table and volunteer table**

**Step 1 : Step by Step process for the code executed for resident table and volunteer table**

1. The user should select and click on execute for the below code

ALTER PROC spGetUnionVolunteerLocation @VolunteerCity varchar(150), @City varchar(150)

AS

SELECT ResidentID,ResidentName,City FROM [dbo].[Residents] --Select all volunteers from Resident table

WHERE City = @City

UNION

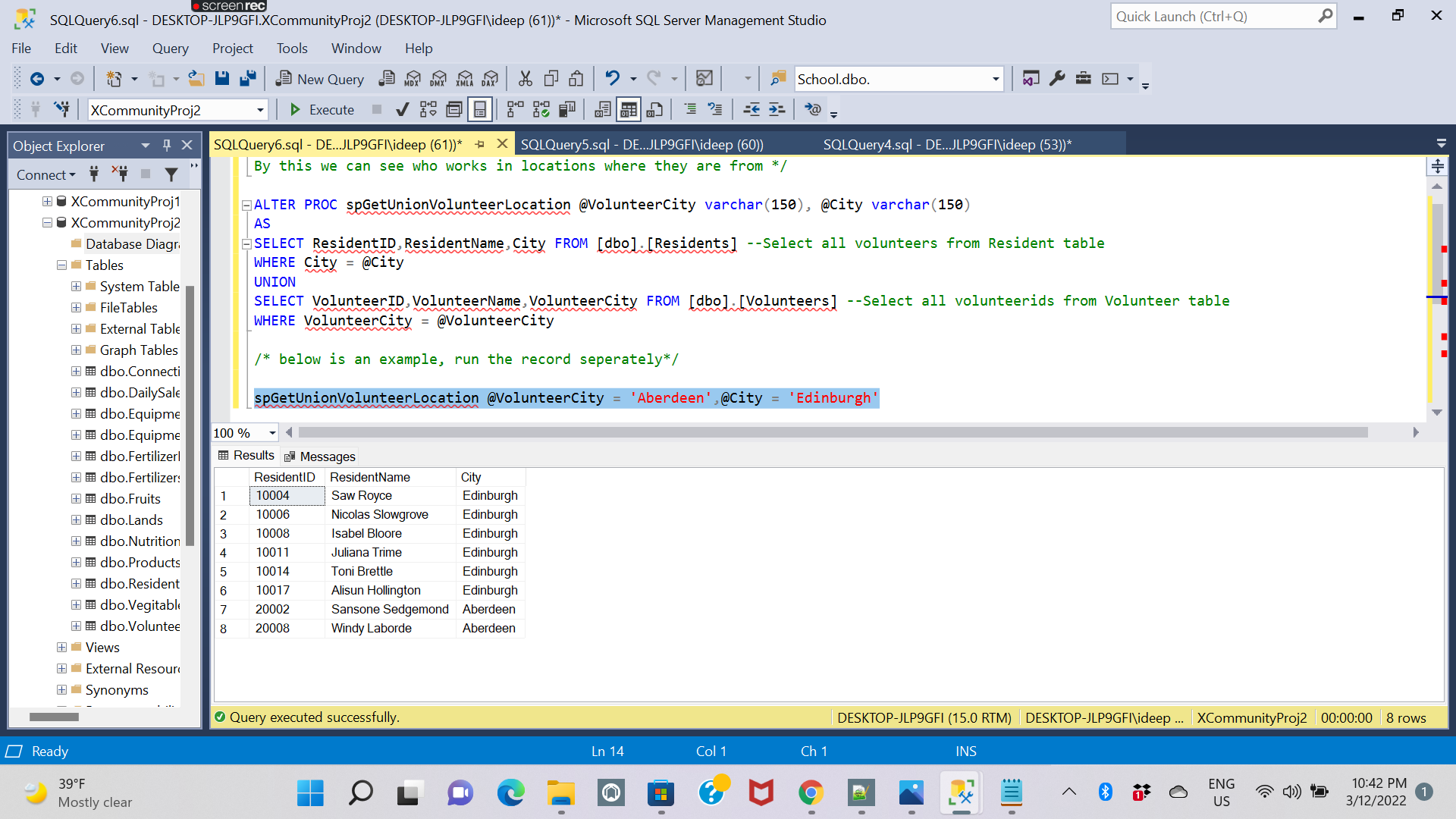
SELECT VolunteerID,VolunteerName,VolunteerCity FROM [dbo].[Volunteers] --Select all volunteerids from Volunteer table

WHERE VolunteerCity = @VolunteerCity

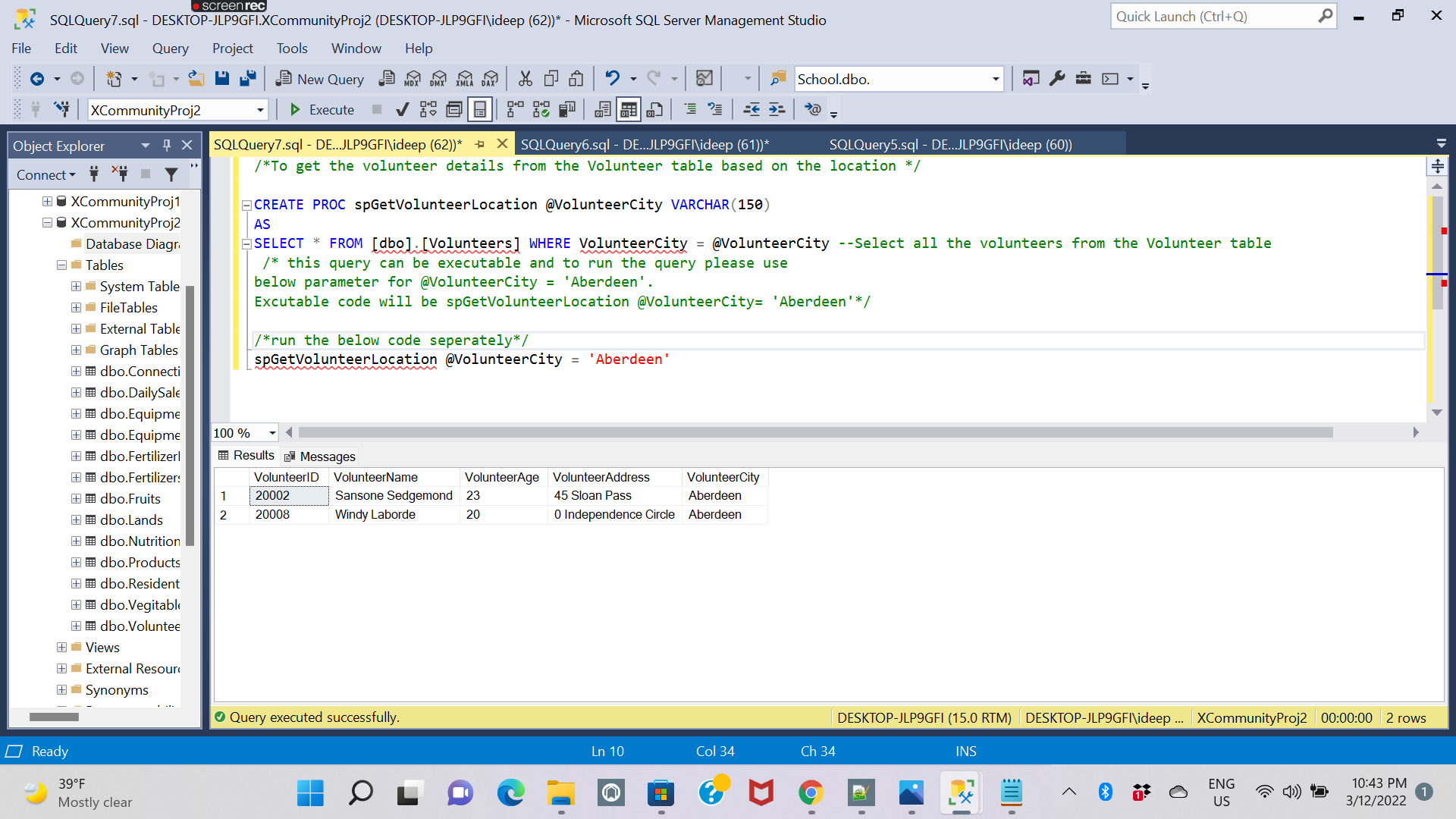
1. The user should again select and click on execute for the below code

spGetUnionVolunteerLocation @VolunteerCity = 'Aberdeen',@City = 'Edinburgh'

1. After clicking on execute the database will retrieve the data of resident table and volunteer table



## 4. Stored Procedure to get volunteer details from the volunteer table based on the location:



**4.1 Run code to stored procedure to get volunteer details from the volunteer table based on the location:**

/\*To get the volunteer details from the Volunteer table based on the location \*/

CREATE PROC spGetVolunteerLocation @VolunteerCity VARCHAR(150)

AS

SELECT \* FROM [dbo].[Volunteers] WHERE VolunteerCity = @VolunteerCity --Select all the volunteers from the Volunteer table

/\* this query can be executable and to run the query please use

below parameter for @VolunteerCity = 'Aberdeen'.

Excutable code will be spGetVolunteerLocation @VolunteerCity= 'Aberdeen'\*/

/\*run the below code seperately\*/

spGetVolunteerLocation @VolunteerCity = 'Aberdeen'

**4.2 Step by Step process for the code executed to get volunteer details from the volunteer table based on the location:**

1. The user should select the below code and click on execute .

CREATE PROC spGetVolunteerLocation @VolunteerCity VARCHAR(150)

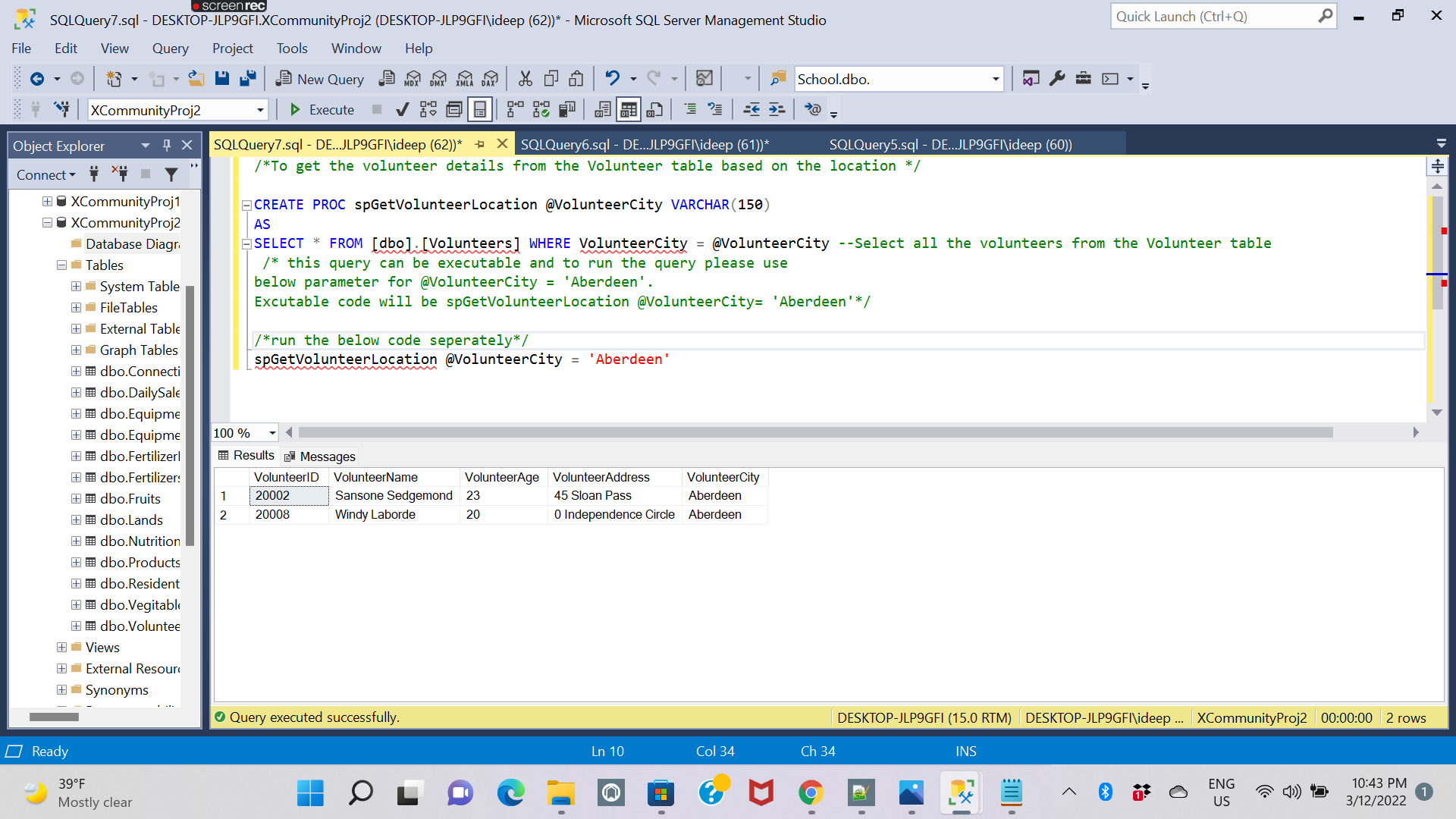
AS

SELECT \* FROM [dbo].[Volunteers] WHERE VolunteerCity = @VolunteerCity

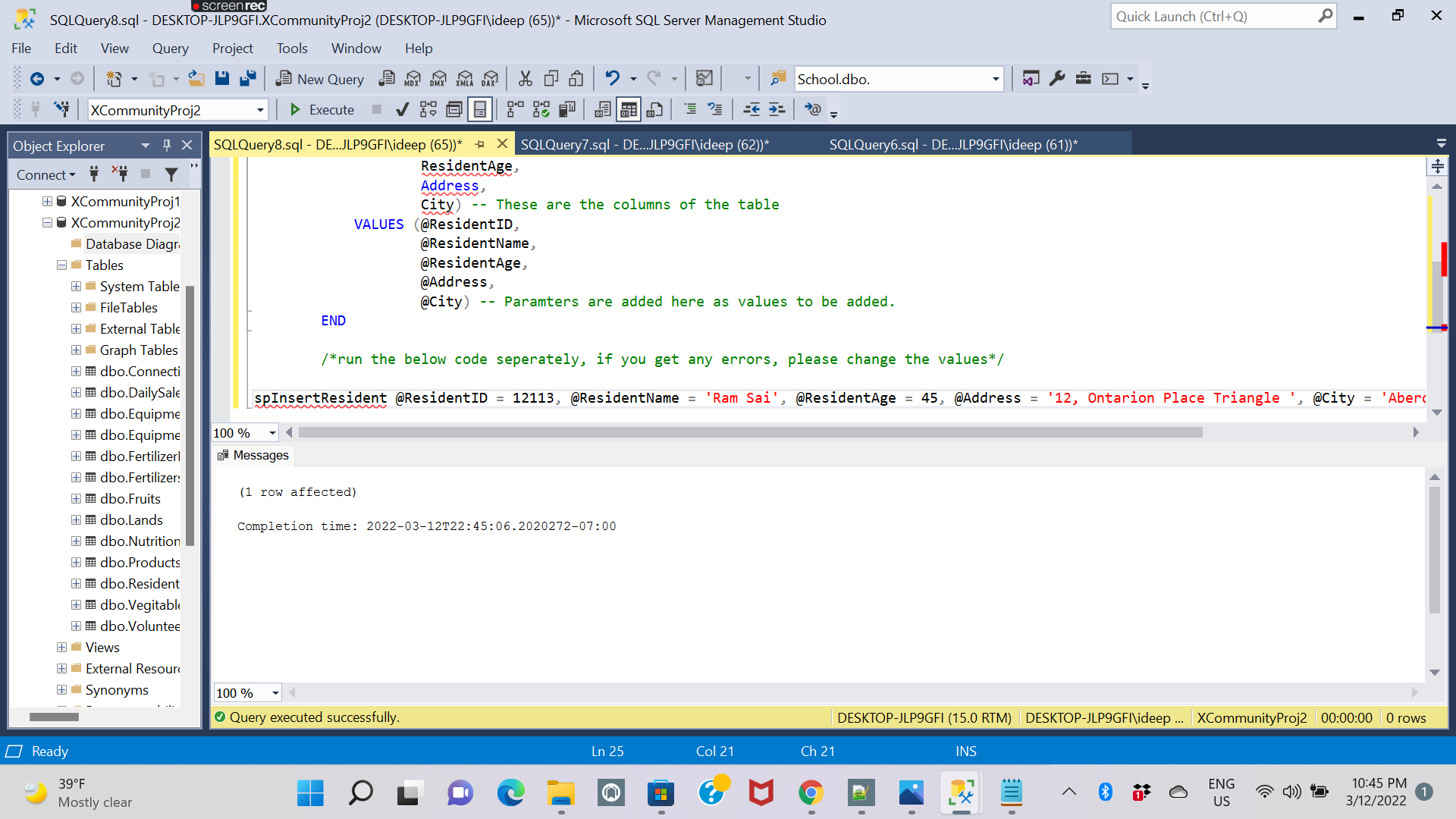
1. The user should select the below code again and click on execute .

spGetVolunteerLocation @VolunteerCity = 'Aberdeen'

1. After the code is executed the database will retrieve the data of volunteer details from the volunteer table based on the location



## 5. Stored procedure to insert a new resident to the table



**5.1 Run code to stored procedure to insert a new resident to the table**

/\*to insert a new resident to the table\*/

CREATE PROC spInsertResident (@ResidentID INT,

@ResidentName varchar(150),

@ResidentAge int,

@Address varchar(150),

@City varchar(150)) --These are parameters for user has to insert

AS

BEGIN

INSERT INTO [dbo].[Residents]

(ResidentID,

ResidentName,

ResidentAge,

Address,

City) -- These are the columns of the table

VALUES (@ResidentID,

@ResidentName,

@ResidentAge,

@Address,

@City) -- Paramters are added here as values to be added.

END

/\*run the below code seperately, if you get any errors, please change the values\*/

spInsertResident @ResidentID = 12112, @ResidentName = 'Alfred Wiley', @ResidentAge = 45, @Address = '12, Ontarion Place Triangle ', @City = 'Aberdeen'

**5.2 Step by step process to stored procedure to insert a new resident to the table**

1. The user has select the below code and click on execute

CREATE PROC spInsertResident (@ResidentID INT,

@ResidentName varchar(150),

@ResidentAge int,

@Address varchar(150),

@City varchar(150)) -These are parameters for user has to insert

AS

BEGIN

INSERT INTO [dbo].[Residents]

(ResidentID,

ResidentName,

ResidentAge,

Address,

City) -- These are the columns of the table

VALUES (@ResidentID,

@ResidentName,

@ResidentAge,

@Address,

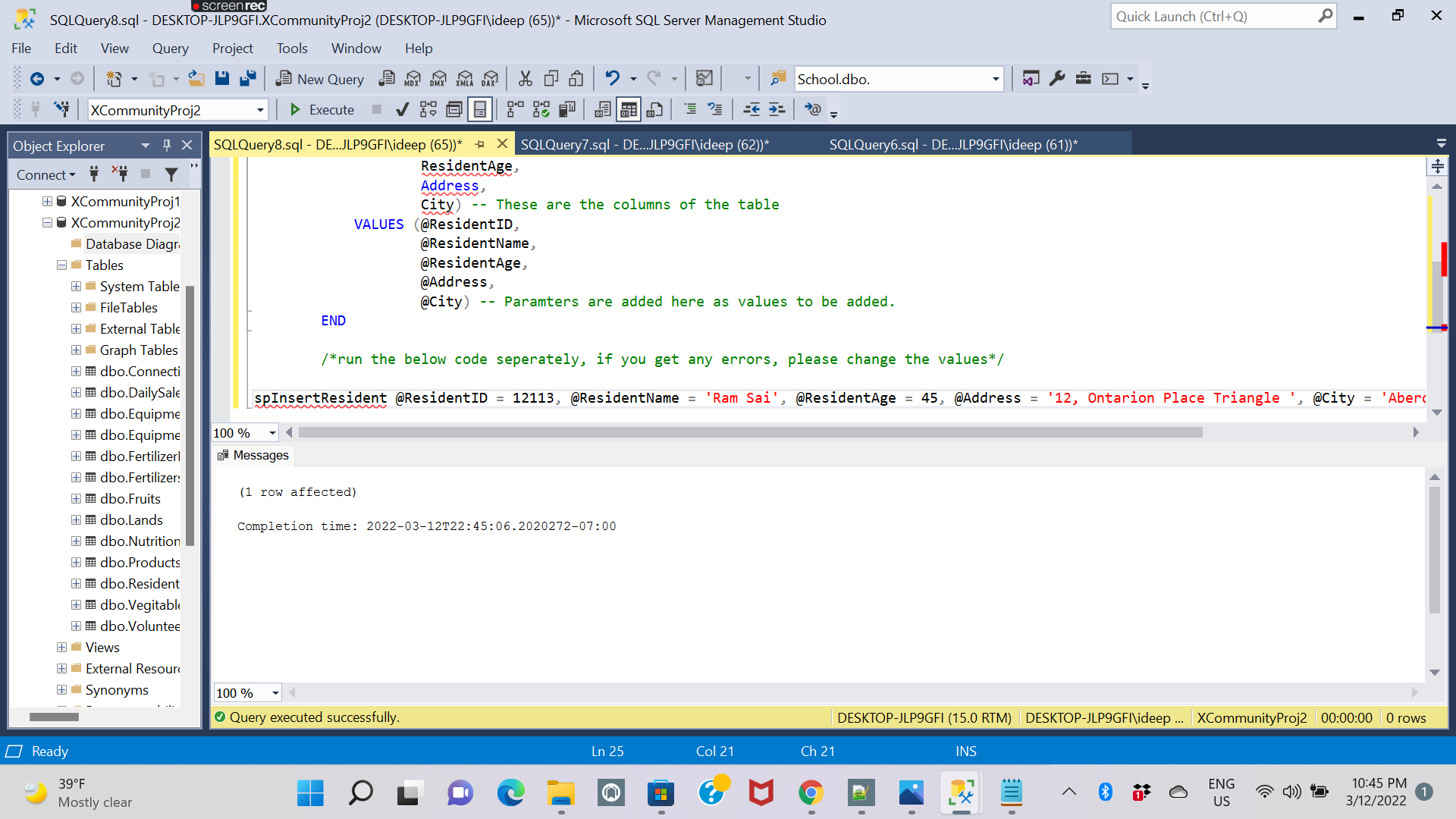
@City) -- Paramters are added here as values to be added.

END

1. User should execute the below code separately

spInsertResident @ResidentID = 12112, @ResidentName = 'Alfred Wiley', @ResidentAge = 45, @Address = '12, Ontarion Place Triangle ', @City = 'Aberdeen'

1. The database will issue the below result with output of new resident to the table.



## 6. Stored Procedure to find out profits per products:

/\*the purpose of this Proc is to find out profits per products\*/

CREATE PROC ProductSalesProfit AS

SELECT

D.ProductID,B.ProductDescription,

D.Quantity,D.SellingUnitPrice,

B.AVGCost,

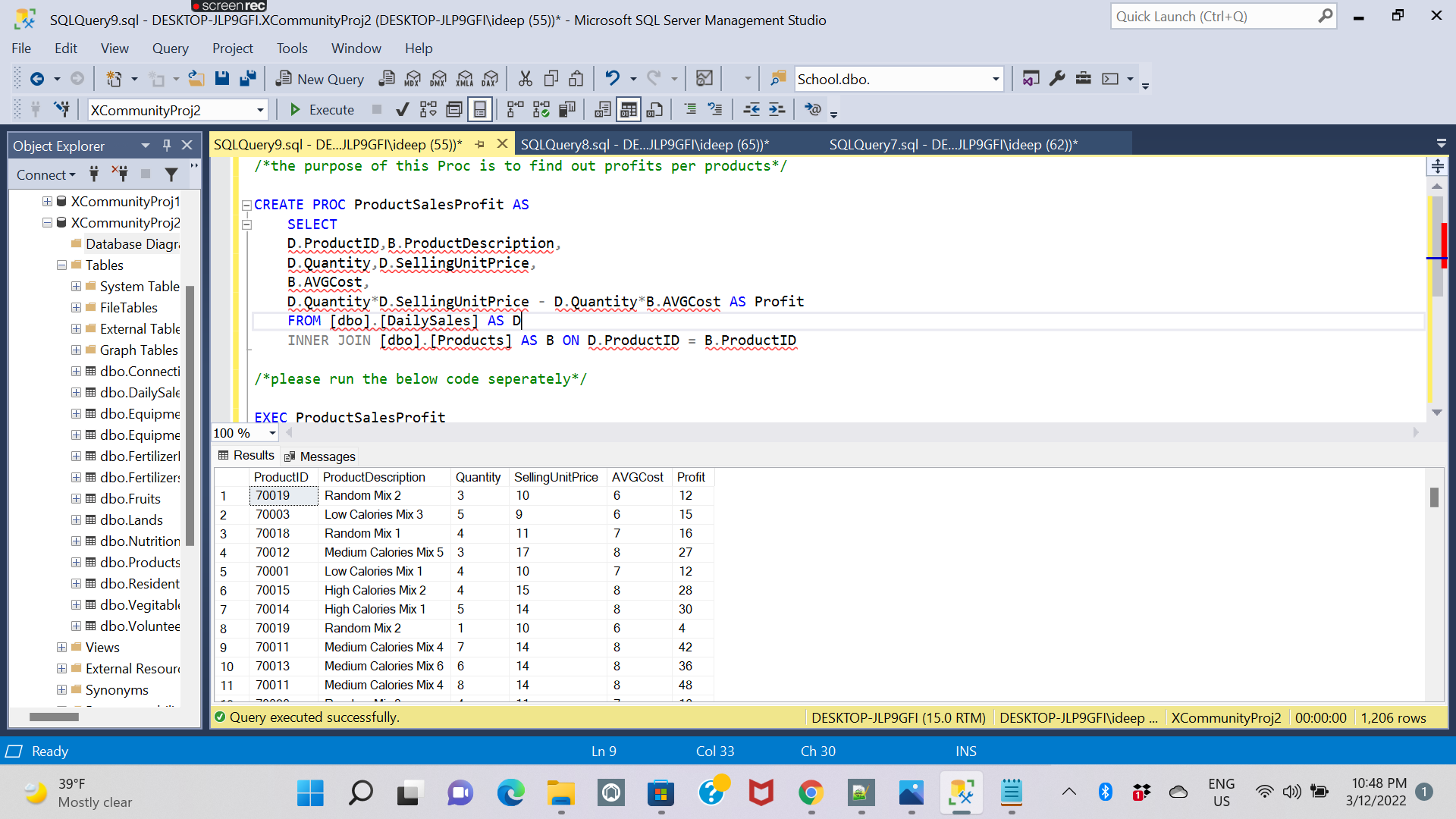
D.Quantity\*D.SellingUnitPrice - D.Quantity\*B.AVGCost AS Profit

FROM [dbo].[DailySales] AS D

INNER JOIN [dbo].[Products] AS B ON D.ProductID = B.ProductID

/\*please run the below code seperately\*/

EXEC ProductSalesProfit



## 7.Stored Procedure get the location of resident based on the location

/\*Get the location of resident based on the location\*/

CREATE PROC ResidentLocations @City varchar(150)

AS

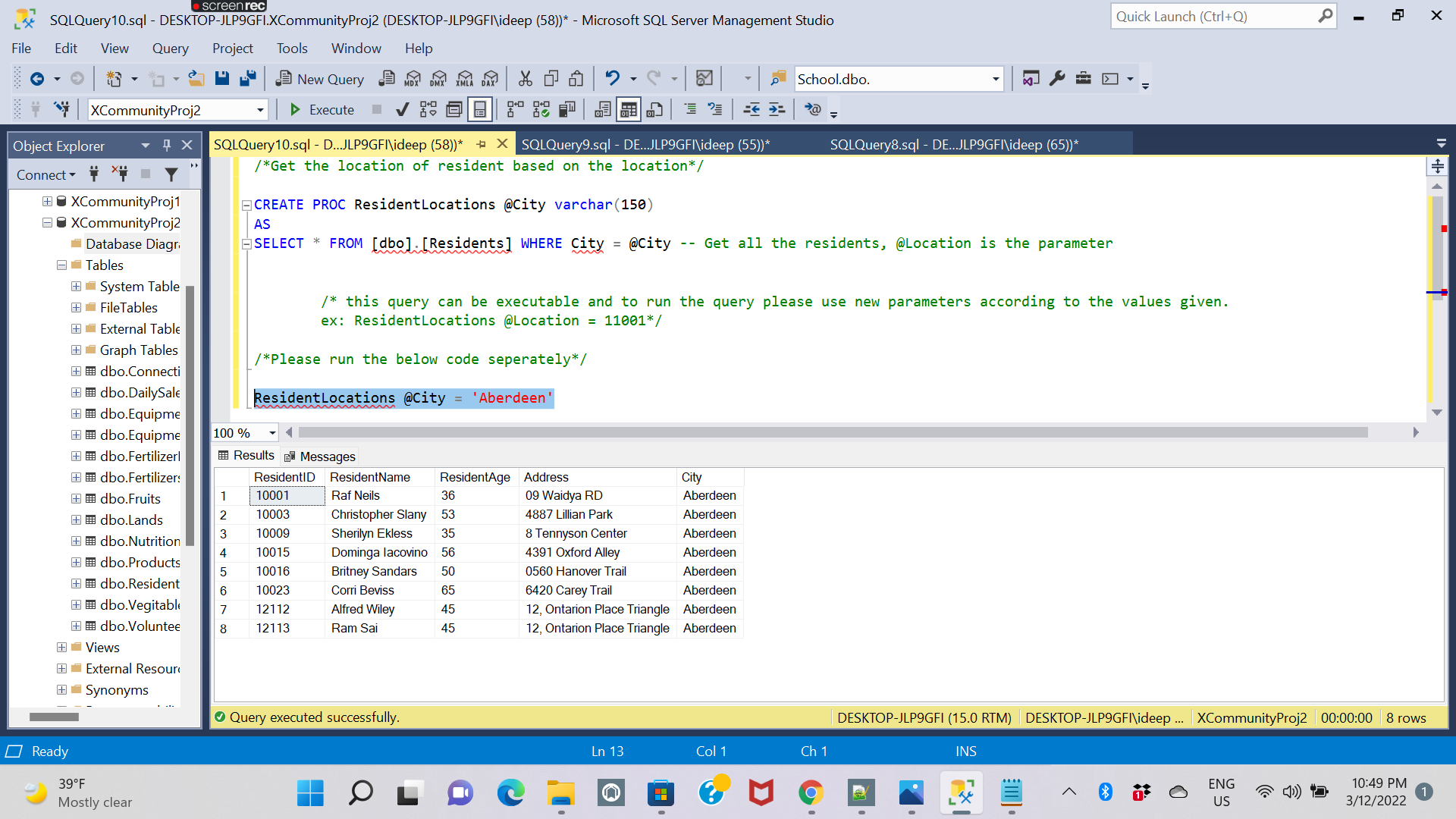
SELECT \* FROM [dbo].[Residents] WHERE City = @City -- Get all the residents, @Location is the parameter

/\* this query can be executable and to run the query please use new parameters according to the values given.

ex: ResidentLocations @Location = 11001\*/

/\*Please run the below code seperately\*/

ResidentLocations @City = 'Aberdeen'



# 

# Views:

## View 1- Profit By Location and Product:

**/\*this view created for PowerBI report: Profit by Location and Product\*/**

CREATE VIEW ProfitProductPerLand AS

SELECT

A.SalesDate,A.ProductID,

C.ProductDescription, B.LandID,

D.LandLocation,

A.Quantity\*A.SellingUnitPrice - A.Quantity\*C.AVGCost AS Profit

FROM [dbo].[DailySales] AS A

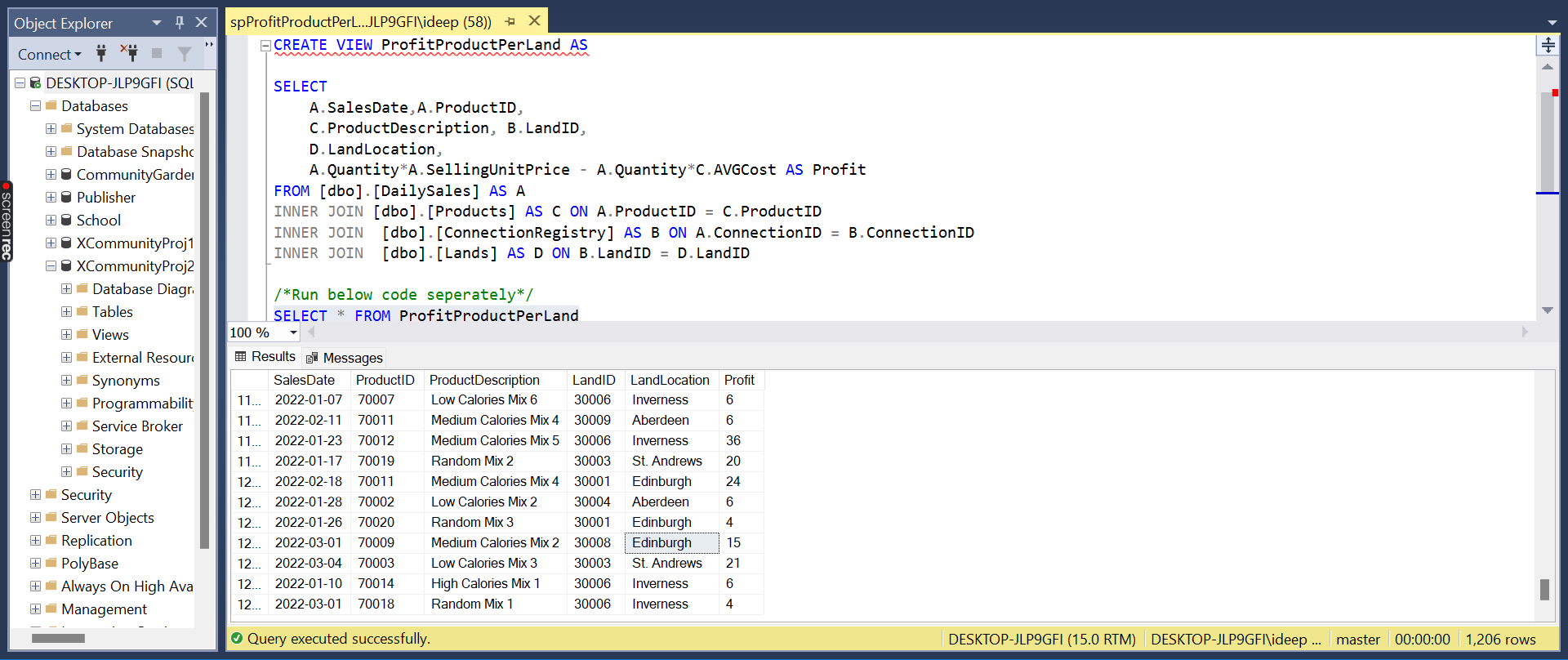
INNER JOIN [dbo].[Products] AS C ON A.ProductID = C.ProductID

INNER JOIN [dbo].[ConnectionRegistry] AS B ON A.ConnectionID = B.ConnectionID

INNER JOIN [dbo].[Lands] AS D ON B.LandID = D.LandID

/\*Run below code seperately\*/

SELECT \* FROM ProfitProductPerLand



## View 2- To identify the Sales in Descending Order

**/\*the purpose of this view is to identify sales in descending order\*/**

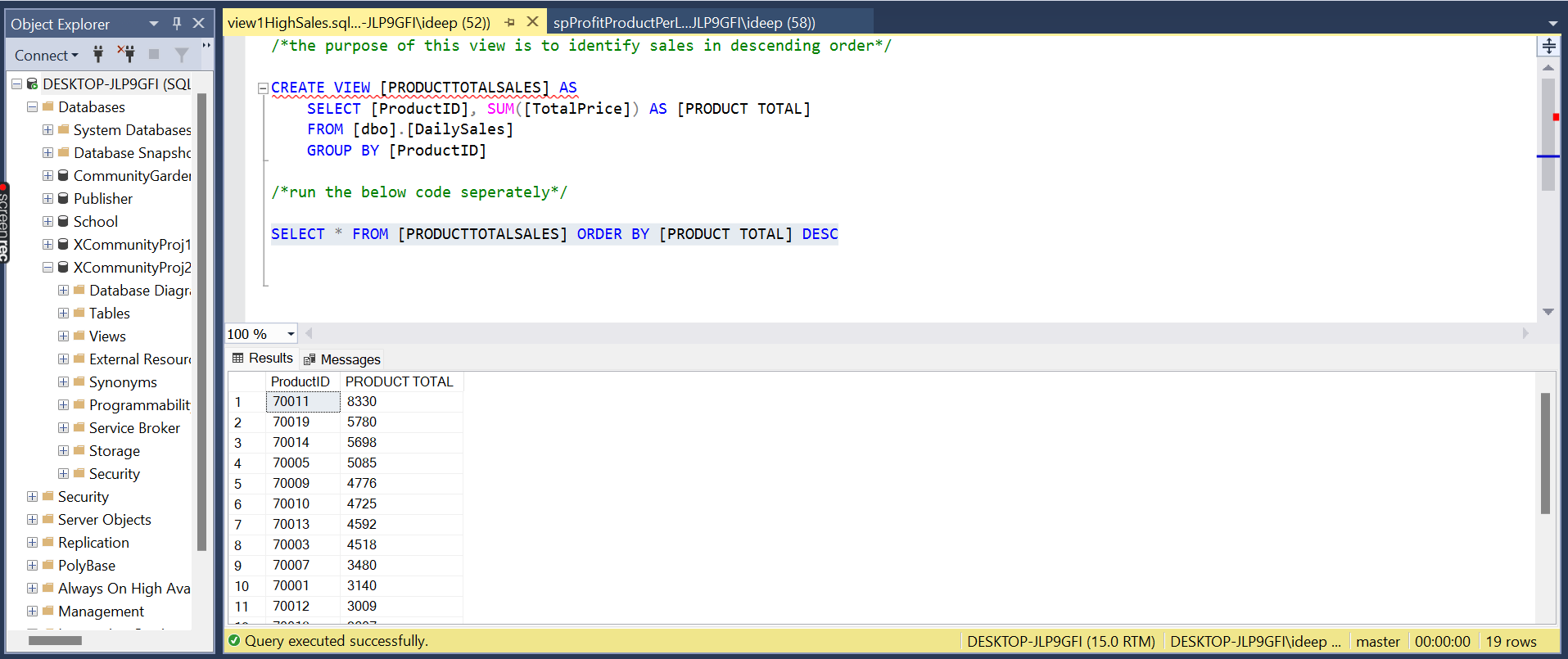
CREATE VIEW [PRODUCTTOTALSALES] AS

SELECT [ProductID], SUM([TotalPrice]) AS [PRODUCT TOTAL]

FROM [dbo].[DailySales]

GROUP BY [ProductID]

/\*run the below code seperately\*/



## View 3- To View Product Total Sales in Descending Order

**SELECT \* FROM [PRODUCTTOTALSALES] ORDER BY [PRODUCT TOTAL] DESC**

/\*the purpose of this view is to identify profit with product description\*/

CREATE VIEW ProfitCalculation AS

SELECT

D.ProductID,B.ProductDescription,

D.Quantity,D.SellingUnitPrice,

B.AVGCost,

D.Quantity\*D.SellingUnitPrice - D.Quantity\*B.AVGCost AS Profit

FROM [dbo].[DailySales] AS D

INNER JOIN [dbo].[Products] AS B ON D.ProductID = B.ProductID

/\*run the below code seperately\*/

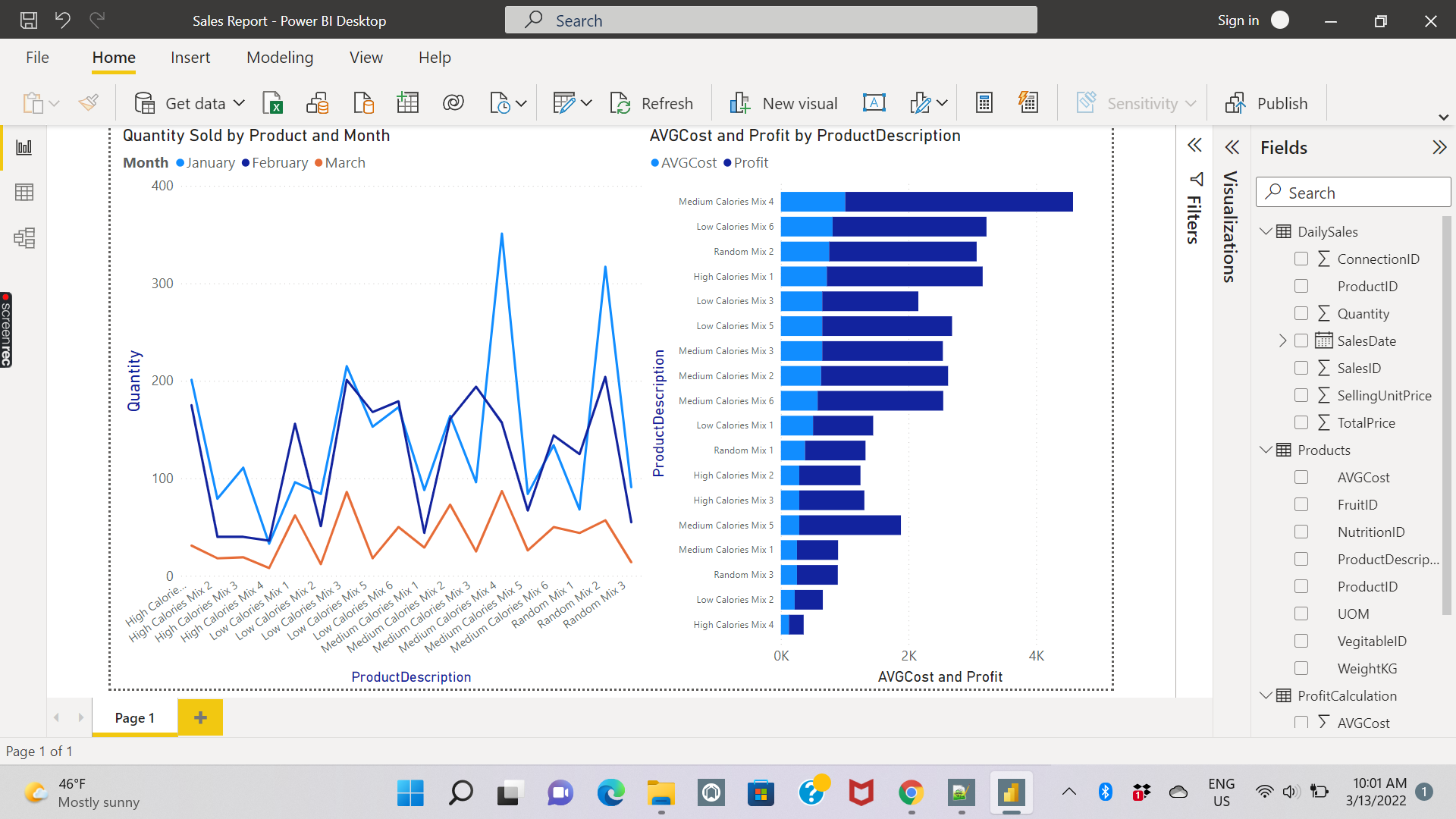
SELECT \* FROM ProfitCalculation ORDER BY ProductID

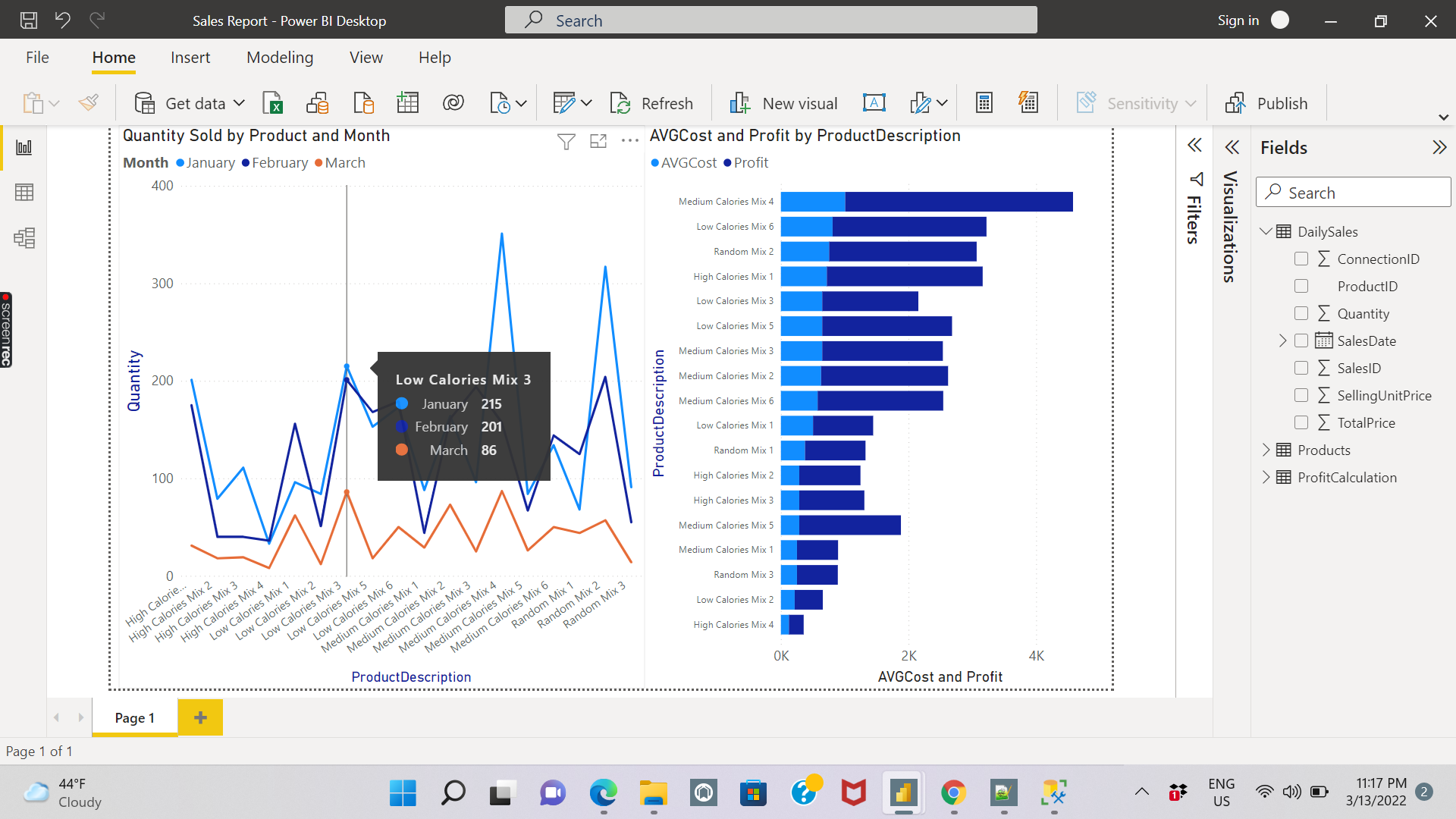


# Reports

## Report1: Sales Report

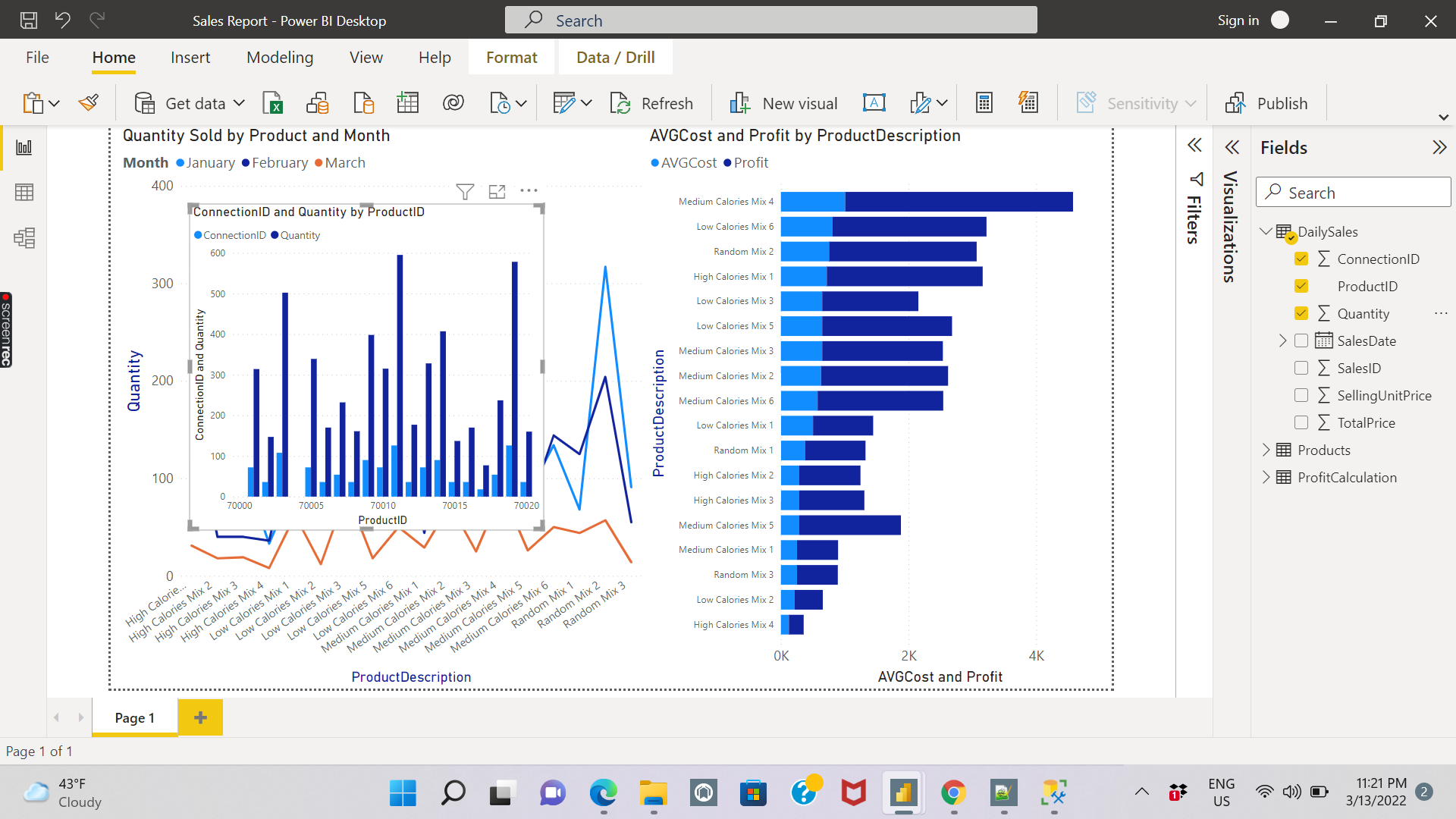
The user can analyse the patterns of relationships among several entities by making use of reports. Here, Power BI and SSRS have been used to make and analyse reports.



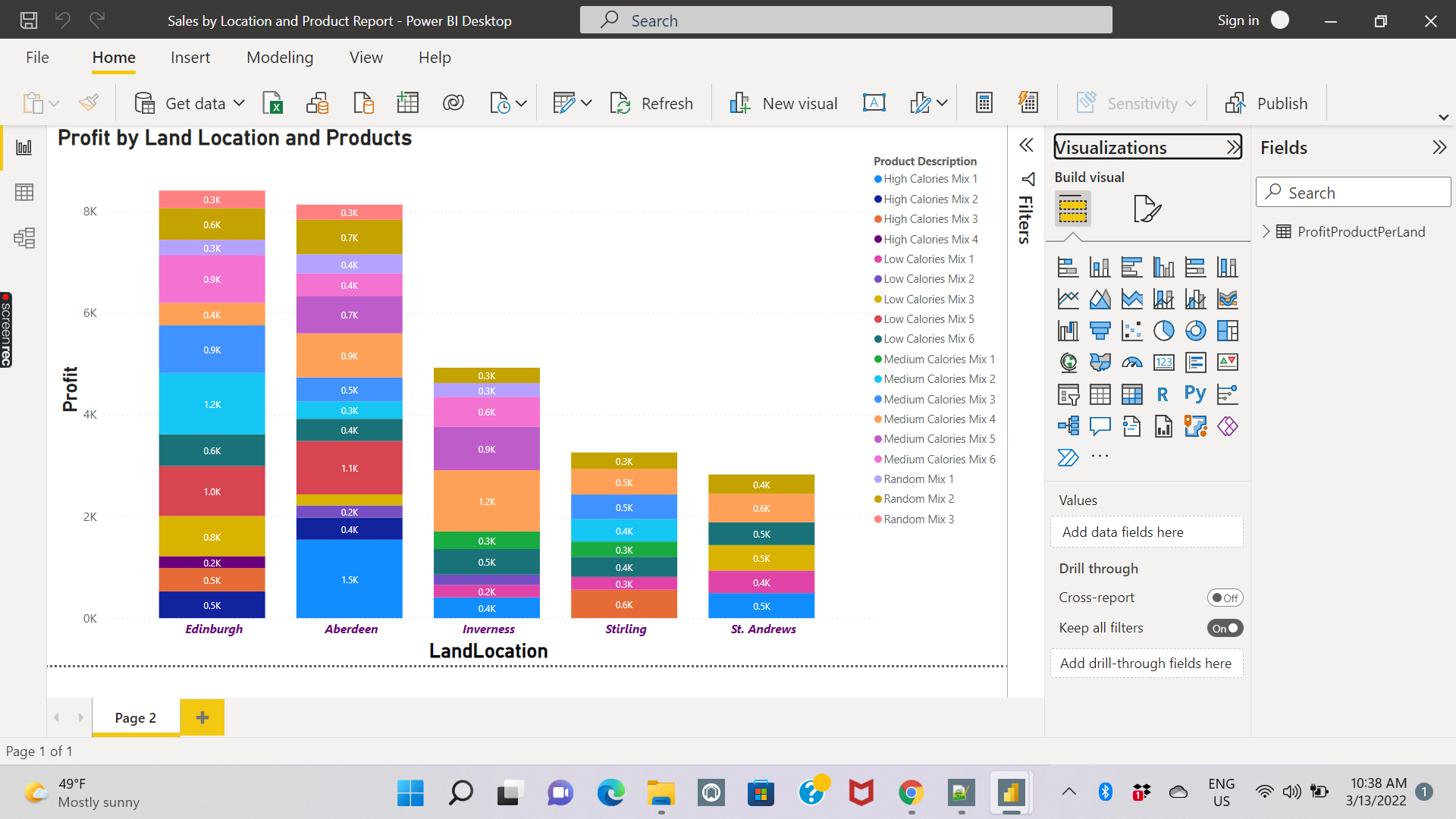


We can know about the sales of the products according to various months.

Further, by clicking on the fields such as Daily sales, connectionID and ProductID, we can draw several insights, such as below.

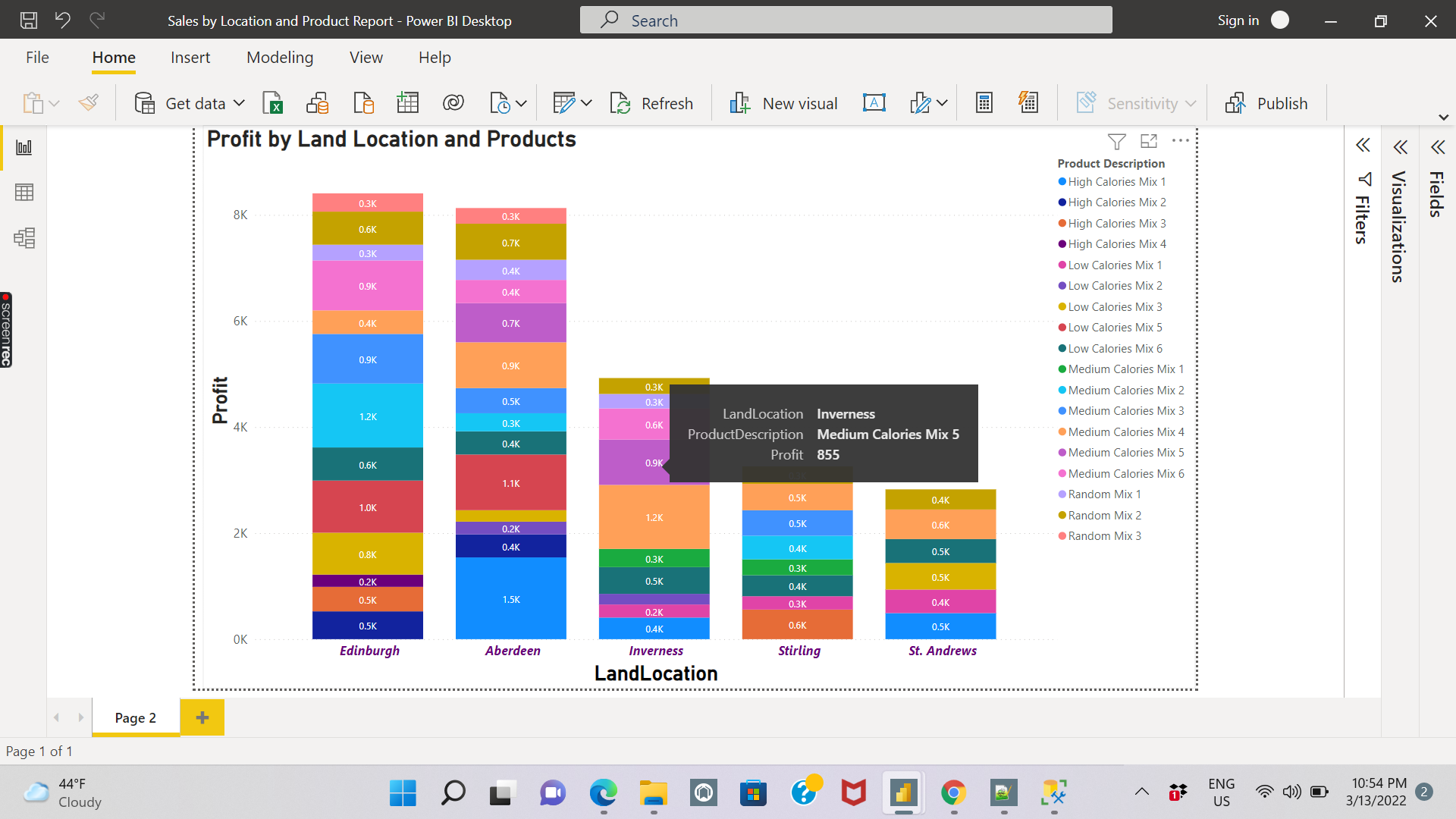


## Report2: Sales by Location and Products



You can analyse the data in the report and look for the details associated with each field. For instance, by hovering your mouse over Inverness as shown below, we can get the information associated with it that the profit associated with the product is 855 and the product description is Medium Calories Mix 5.

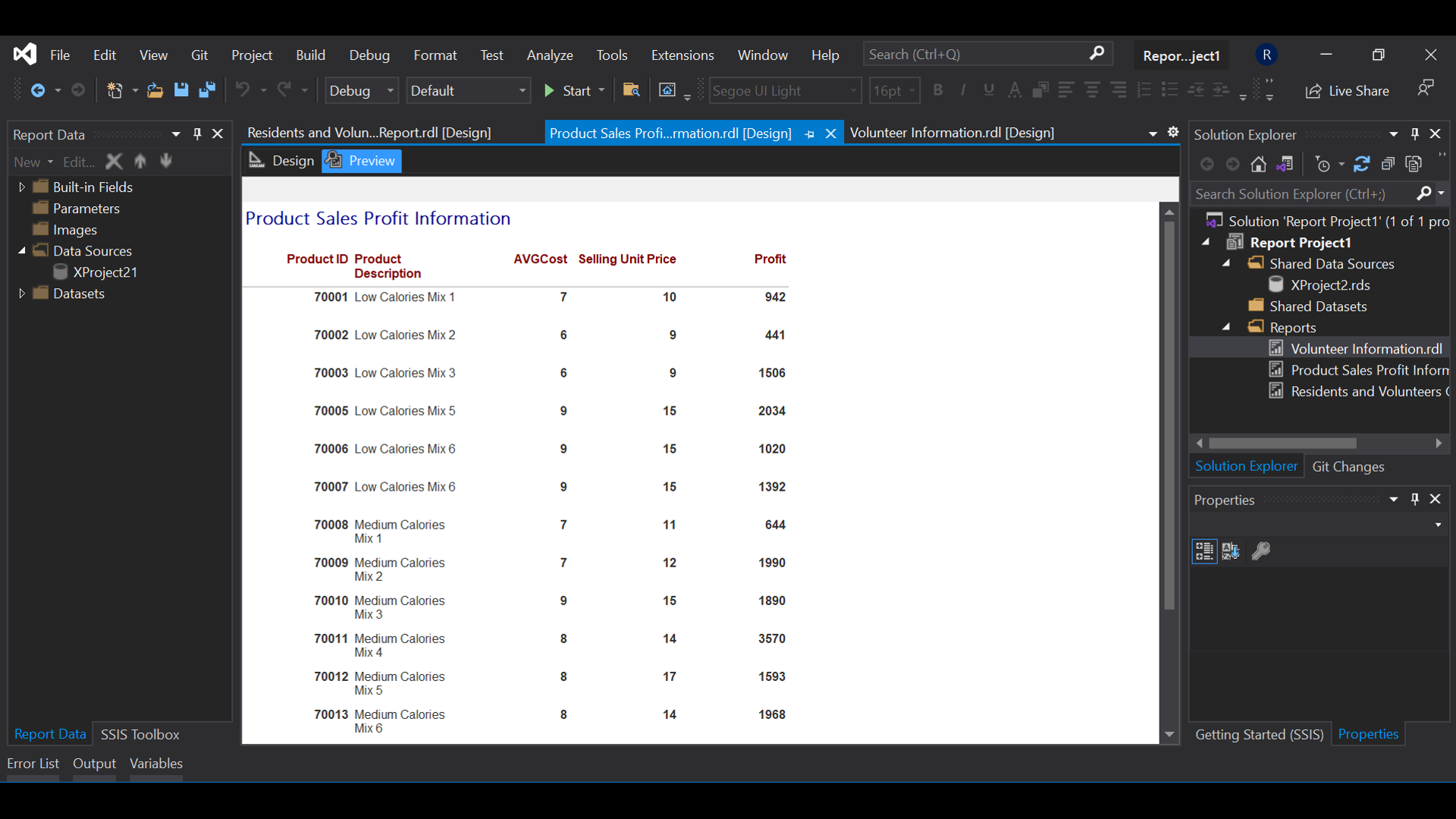
In the same way, we can analyse for the rest of the report.



## Report3: Product Sales Profit Information

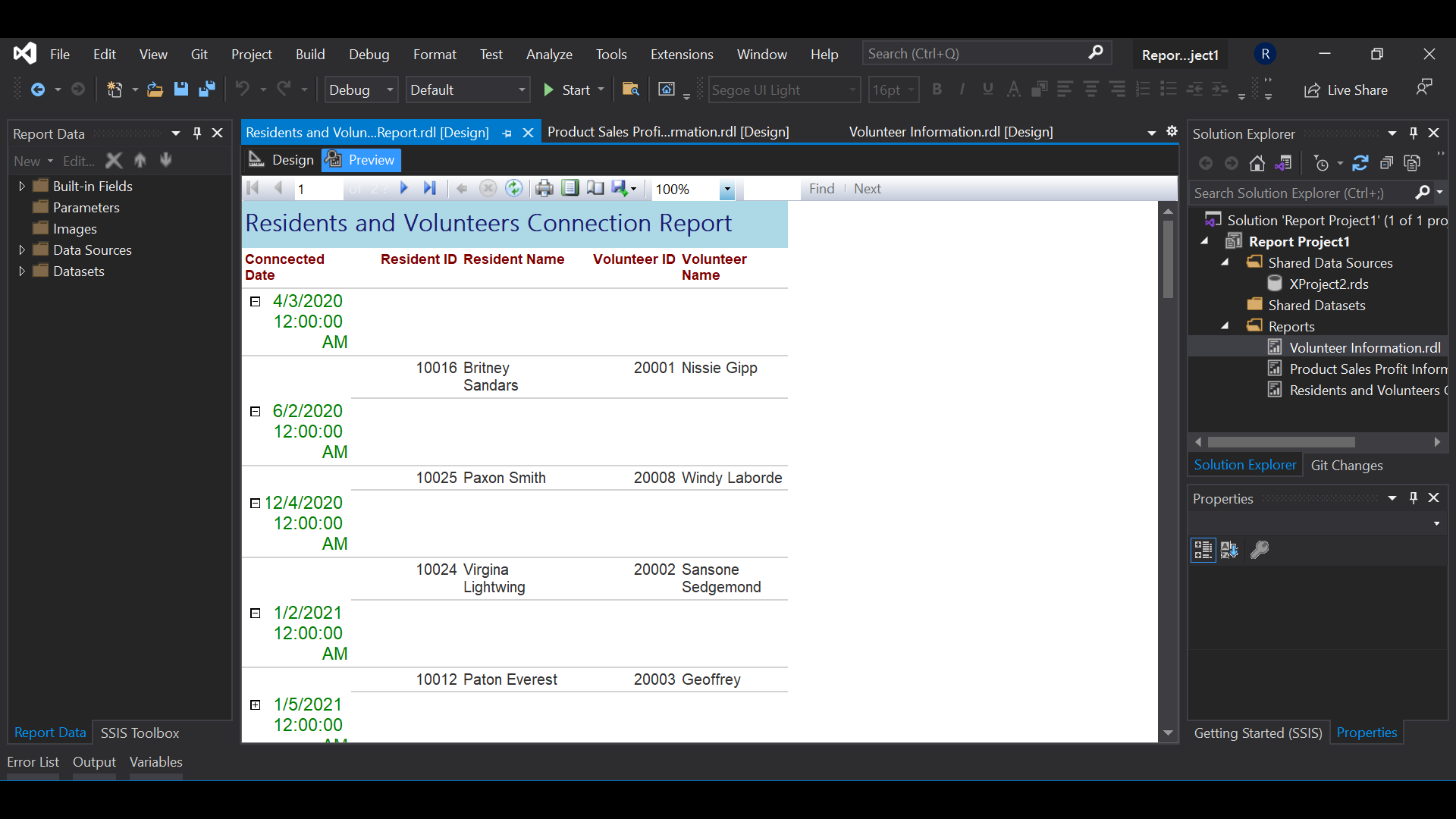
In SSRS, we can analyse the product sales profit information by running its files or connecting the SSRS engine to the database.

Here, the product sales and profit information has been shown

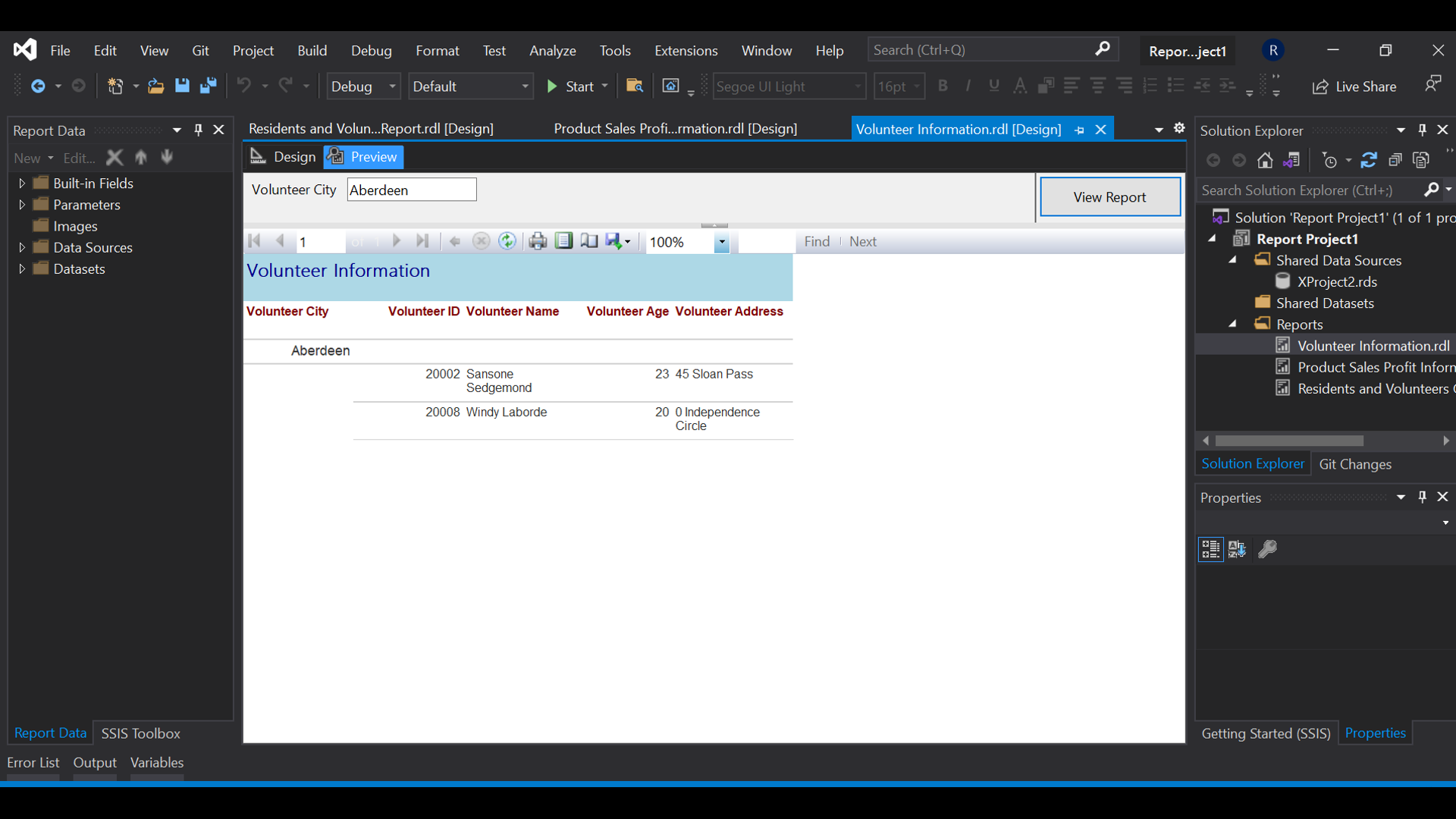


## Report4: Residents and Volunteers Connection Report

In this report, a connection report of residents and volunteers has been given. By looking at this report, we can know the details of residents and volunteers connected to each other and their connection date.



## Report5: Volunteer Information



# User Defined Function

Follow the steps to create a user defined function

/\* This UDF is to understand the impact from inflation on the selling price, this also

can be used for totalprice or quantity\*/

CREATE FUNCTION dbo.Function1 (@ColumnName int, @Percent int)

RETURNS INT AS

BEGIN

RETURN @ColumnName + (@ColumnName\*@Percent)/100

END

GO

/\*change the variable values as required\*/

SELECT \*, dbo.Function1(SellingUnitPrice,10) AS AddingInflation FROM [dbo].[DailySales]

