Working with SQL Server Data

The data you use for Power BI can come from several different sources, but, more often than not, SQL Server will be used for hosting it. We’re going to look at how you can work with SQL Server data and the relationships that exist between tables.

Power BI Desktop is far more robust than the Power BI service for business intelligence reports because it offers support for many more data sources, and it has more tools for you to use in transforming your data.

SQL Server is one of those data sources that can only be used in Power BI Desktop, not in Power BI service. Although the service does give us the connectors for SQL Data Warehouse and Azure SQL Database, it doesn’t offer the ones we need for SQL Server. The Power BI Desktop lets us access SQL Server data from whole tables or allows us to run queries that will return just a subset of the data from several different tables. Stored procedures can be called, including the one that allows you to run both R and Python scripts – sp_execute_external_script.

We’re going to delve into how we retrieve the data from an instance of the SQL Server and look at some different ways the data can be returned. As with the last chapter, we are using the AdventureWorks2017 database, and we are running it on a local SQL Server instance.

Most of what we discuss here can easily be used on any other RDBMS – Relational Database Management System – and on multiple data source types. All data in Power BI is organized into queries, which are structures similar to tables and used when we work with datasets. Because of that, once the data is in the Desktop, most of the actions you do are likely to be similar, regardless of where the data came from. You will come across features that are specific to a relational database, though, especially when you are looking at the relationships between the tables.

Retrieving an SQL Server Table

Power BI Desktop lets you retrieve entire SQL Server tables or views. It will preserve the structure and, where it can, it will identify what, if any, relationship exists between them. The example we're going to work on now demonstrates how to import tables and, if you intend to follow this practically, the first thing to do is retrieve the tables we want from our AdventureWorks2017 database.

To do that, open Power BI and go to the Home ribbon. Click on Get Data, and the relevant dialog box will open. Go to the Database section and double-click on SQL Server Database. A new dialog box opens, type in the SQL Server instance name of SqlSrvInstance, and the target database name of SqlSrvDatabase.

These are the two connections created for the purpose of this example. SqlSrvInstance has the name of the Server instance, while SqlSrvDatabase has the target database name. When you provide information about the connections, you have a choice – use the name of the connection or create parameters of your own.

Once you have input the required details, click on OK, and the Navigator dialog box will open. Choose the following three tables:

- Person.Address
- Person.BusinessEntityAddress
- Person.Person

In the right pane, you can see the selected table's data. If you wanted to include the tables related to the chosen ones, you could click on Select Related Tables. If, for example, you choose Person and then click on Select Related Tables, you would get nine tables, way more than we need for our purpose. The project you are working on will determine if you use this option or not but, if you do opt for it and only want some of the related tables, use the checkboxes to include or exclude tables.

When you want to import your tables, click on Load, and every table or view that you chose gets loaded to the Power BI Desktop, and each is listed in the Data view as an individual dataset. The datasets can be modified in Query Editor, the same as you do with any other data.

Probably, one of the first things you will do in Query Editor is to rename the datasets and get rid of any extra columns:

- **Rename a Dataset –** in Queries, right-click on the dataset and choose Rename; type the new name and press the Enter key
- **Remove a Column –** in the main grid, pick the column and go to the Home ribbon; click on Remove Columns
- **Remove Multiple Columns in One Go –** choose the first column, press/hold CTRL and choose each column you want to remove; on the Home ribbon, click on Remove Columns

Following these instructions, do the following for your three imported tables:

- **Person.Address table –** change the name to Address and then remove six columns – AddressLine1, AddressLIne2, PostalCode, SpatialLocation, rowguid, ModifiedDate
- **Person.BusinessEntityAddress table –** change the name to BusEntAddress and then remove three columns – AddressTypeID, rowguid, ModifiedDate
- **Person.Person table** – change the name to Person and then remove ten columns – PersonType, NameStyle, Title, MiddleName, Suffix, EmailPromotion, AdditionalContactInfo, Demographics, rowguid, ModifiedDate.

For the time being, we can ignore any of the columns that have relationship data in them; we'll come back to them later.

Once you have updated your queries, make sure the changes are applied and come out of Query Editor. You will see that every dataset now has just two or three fields in it. Filters can also be added in Query Editor if you want the data refined further, or filters can be added to the visualizations when you do your reports.

Work with Relationships

Did you notice, when the datasets were updated in Query Editor, that the queries had several pseudo-columns in them; these were indicated by overlong names and values in gold, usually Value or Table.

These columns are used to represent a relationship between tables in the dataset. If a column has Table values, these represent foreign keys in a different table that is referencing the primary table on which we based the query. If they have Value values, these are representing foreign keys in our primary table, referencing a different table.

As an example, if you are in Query Editor and you choose the Address query, four of those relationship columns will appear:

- Person.BusinessEntityAddress
- Person.StateProvince
- Sales.SalesOrderHeader
- Sales.SalesOrderHeader(AddressID)

These columns are indicating that the Address table, as it is in the original database, has a foreign key in it, referenced by three other foreign keys:

- In the Address query, the column called Person.BusinessEntityAddress is representing a foreign key that is in the table called Person.BusinessEntityAddress; the key is referencing the table called Person.Address. The foreign key gets created in the Person.BusinessEntityAddress table, in the AddressID column.
- In the Address query, the column called Person.StateProvince is representing a foreign key that is in the table called Person.Address; the key is referencing the table called Person.StateProvince. The foreign key gets created in the Person.Address table in the StateProvinceID column.
- In the Address query, the column called Sales.SalesOrderHeader(AddressID) is representing a foreign key that is in the table called Sales.SalesOrderHeader; the key is referencing the table called Person.Address. The foreign key gets created in the Sales.SalesOrderHeader table in the BillToAddress column.
- In the Address query, the column called

> Sales.SalesOrderHeader(AddressID) 2 is representing a foreign key that is in the table called Sales.SalesOrderHeader; the key is referencing the table called Person.Address. The foreign key gets created on the Sales.SalesOrderHeader table in the ShipToAddress column.

Click on any one of the Table or Value values, and a new step is added by Query Editor to Applied Steps (in the right pane); the dataset is also modified, so it includes information that relates specifically to the chosen value. For example, take the first row from the Address query; you can see it has a value of 1 for AddressID and a value of 79 for StateProvinceID; if you were to click on the value for Value in that row's Person.StateProvince column, a new step, would be added to applied steps. The step is named 1, and the dataset is modified, so it only has information out of the StateProvince table that is specific to the value of 79 for StateProvinceID.

The data can be seen for any Value instance in Person.StateProvince simply by clicking on it. The step associated with it will be given the name of the selected row's AddressID value and, when you are done looking at the results, you can delete the newly added steps and put the query back to how it was.

If you go to the Person.BusinessEntityAddress column and click on the Table value in the first row, a new set would be added, named as 1, but this time containing information from the table called BusinessEntityAddress. The dataset would contain just one row after it was modified, and there would be a foreign key reference in the row to the value of 1 for AddressID in the Address table.

Play about with the Table and the Value values to see all the different data related to them; just make sure that the newly added steps are deleted from Applied steps, so the query is back to its original form. You won't be doing this in the examples here, but it is good to learn for other SQL Server-related projects.

As well as finding the relationships with foreign keys, Power BI Desktop will also try to identify any relationships that exist between the datasets we imported. These relationships can be seen by looking at Relationships in the main Desktop window. You can see each dataset represented graphically, and you can move these around as you wish. If a relationship is found between any datasets, a connector is used to join them, indicating the relationship type.

In our case, Desktop has determined that there is a one-to-many relationship between two datasets – Person and BusEntAddress. It has also detected a one-to-one relationship between two datasets – BusEntAddress and Address. Double-clicking on the connector will show you the relationship details. For example, to see the details of the one-to-many relationship between BusEntAddress and Person, double-click on the connector between the two – the Edit Relationship box opens, and you can see the details relating to the relationship. If needed, you can also modify the details, although you will find that Power BI Desktop already gets the relationships right, most of the time, based on the data currently held in the dataset. One thing you might spot is that this is not shown as a one-to-many relationship; it is a many-to-one relationship instead. This comes down to the order the datasets are listed in; in our case, BusEntAddress is first.

With Power BI Desktop doing all this work for you, it is much easier to come up with visualizations that are based on several datasets. Later in the book, we'll be looking at creating those visualizations, so, for now, just understand that the process is made much easier because of the relationships – the data has all been mapped for you.

How to Merge Datasets

There may be occasions when you want several datasets combined into one. You might, for example, decide that the three datasets we imported earlier should be combined into one. To do this, go to Query Editor and click on Person. Click Merge Queries on the Home ribbon and then select Merge Queries as New.

When the Merge dialog box opens, go to the first drop-down menu and make sure that Person is selected. Next, go to the middle drop-down menu and choose BusEntAddress and, in the drop-down menu for Join Kind, choose Inner (Only Matching Rows). Query Editor now knows that an inner join needs to be created between the queries. Lastly, for each of the referenced datasets, choose the column titled BusinessEntityID and click on OK.

A new query now gets created, and you can rename this as CityData. The query has the first three columns from Person, together with some relationship columns. Delete every column except for the one called BusEntAddress, as this one represents the current query.

Next, go to the BusEntAddress query and choose at least one column for the merged query. To do this, go to the icon shown in the top right of the column called BusEntAddress and clear every column, leaving just the AddressID column. There is no need for you to add BusinessEntityID as it is already in the Person dataset; the rest of the columns are just the relationship columns.

You should also clear the checkbox next to Use Original Column Name as Prefix, so the column name is kept simple; click on OK, and the AddressID column will be added to the CityData query.

Now we want to merge our Address and CityData queries. Keeping CityData selected in the Query Editor, go to the Home ribbon and click on Merge Queries. The Merge dialog box opens, go to the second drop-down menu and choose Address. In the Join Kind menu, choose Inner (Only Matching Rows). And, for both datasets, the AddressID column should be selected; then click OK.

Now go the top right of the reference column called Address and click on the icon; clean the columns, leaving just two – City, and StateProvinceID. Also, clear the checkbox beside Use Original Column Name as Prefix, click on OK, and your datasets will merge.

You now have just one dataset you can use for other operations. As an example, you could apply transformations, like row filtering, column splitting, pivot the data, and so on, while retaining the two original datasets. And with just one dataset to work on, visualizations are easier to create because you don’t have to fight your way through several sources. However, as with any Power BI feature, whether you use it will depend on whether your project will benefit from it.

Retrieving SQL Server Data Using T-SQL Queries

Power BI Desktop gives you several ways to transform your data but, without a doubt, the ability to return data from SQL Server databases using T-SQL queries is one of the most powerful methods. It saves you from having to specify each individual view or table and, by using a query, you can get the data you want without the need to filter rows, remove columns, or do anything else once the data is imported. This means that you work with just one dataset that has all the data you could want instead of having to go through several datasets to extract a bit of information here and there.

For this example, we will use the SELECT statement below to join six of the AdventureWorks2017 tables together, returning the data to Desktop:

```
SELECT cr.Name AS Country, st.Name AS Territory,
 p.LastName + ', ' + p.FirstName AS FullName
FROM Person.Person p
  INNER JOIN Person.BusinessEntityAddress bea
    ON p.BusinessEntityID = bea.BusinessEntityID
  INNER JOIN Person.Address a
    ON bea.AddressID = a.AddressID
  INNER JOIN Person.StateProvince sp
    ON a.StateProvinceID = sp.StateProvinceID
  INNER JOIN Person.CountryRegion cr
    ON sp.CountryRegionCode = cr.CountryRegionCode
  INNER JOIN Sales.SalesTerritory st
    ON sp.TerritoryID = st.TerritoryID
WHERE PersonType = 'IN' AND st.[Group] = 'North America'
ORDER BY cr.Name, st.Name, p.LastName, p.FirstName;
```

Now we want to run the query so, in the Desktop window, go to the Home ribbon and click on Get Data. In the dialog box, go to Database, find SQL

Server Database, and double-click on it.

A new dialog box opens, type in the SQL Server instance name as SqlSrvInstance, and the target database name as SqlSrvDatabase. Click the arrow for Advanced Options, and the dialog box expands. Go to the SQL Statement box and type in the SELECT statement from above.

Click OK, and a preview window is displayed, showing you some of the data that is being imported. Click on Load, and the data will load into Power BI Desktop. When it has loaded, click the Data view and find the new dataset. Right-click it and click on Rename; input the new name as CountryData and press the Enter key.

The dataset is now just as you want it, and you haven't needed to go through all those extra steps you have to do when you import tables one at a time. Extra transformations can still be applied if you want them but, most of the time, that won't be necessary.

Power BI Desktop lets you run all sorts of T-SQL queries when you want data retrieved from a Server database, such as the EXECUTE statement used for called stored procedures. As an example, the stored procedure called sp_execute_external_script can be called if you want an R script to run or, as you can see in the example below, a Python script:

```
DECLARE @pscript NVARCHAR(MAX);

SET @pscript = N'

# import Python modules

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from microsoftml import rx_oneclass_svm, rx_predict

# create data frame

iris = load_iris()

df = pd.DataFrame(iris.data)
```

```
df.reset_index(inplace=True)
df.columns = ["Plant", "SepalLength", "SepalWidth", "PetalLength", "PetalWidth"]
# split data frame into training and test data
train, test = train_test_split(df, test_size=.06)
# generate model
svm = rx_oneclass_svm(
    formula="~ SepalLength + SepalWidth + PetalLength + PetalWidth", data=train)
# add anomalies to test data
test2 = pd.DataFrame(data=dict(
Plant = [175, 200],
SepalLength = [2.9, 3.1],
SepalWidth = [.85, 1.1],
PetalLength = [2.6, 2.5],
PetalWidth = [2.7, 3.2]))
test = test.append(test2)
# score test data
scores = rx_predict(svm, data=test, extra_vars_to_write="Plant")
OutputDataSet = scores';
EXEC sp_execute_external_script
  @language = N'Python',
  @script = @pscript
  WITH RESULT SETS(
    (SpeciesID INT, Score FLOAT(25)));\
```

Be aware that, to run this script, Machine Learning Services (In-Database) must be installed, and the property called sp_execute_external_script enabled.

In our case, the stored procedure runs a Python script used for looking at sample databases with iris flower data and identifying any anomalies. Run the script, and you should get a dataset called IrisData. Each time the query is run, the values will be different because the data is random. In the dataset, you can see the IDs for the iris species and scores that indicate anomalies.

The example doesn't use SQL Server data, but it does show that Power BI Desktop can run Python scripts. And you can create scripts that include data from multiple sources or SQL Server database data.

Perhaps the best part about is this is that you get to use the analytical capabilities offered by Python while still being able to import the data in the format that suits you. That lets you create your visualizations easily, providing critical insights into the data you have available.

Make the Most of SQL Server Data

Being able to run the T-SQL queries in Desktop is a powerful way of accessing the SQL Server data. As well as gaining huge benefits from using the SQL Server query engine, you also get to reduce how many datasets are imported to Power BI Desktop and reduce the number of transformations you need to make.

You do need to have an understanding of T-SQL, but once you have that, you have way more flexibility when you want to work with SQL Server data. However, even if you still import your data one table at a time, you can still make use of some flexible tools for working with Server data and preparing the data so you can create useful visualizations for posting on Power BI service.

As I said earlier, Power BI service does not provide an SQL Server connector. The only way around it is to export the data form the SQL server into a file that would then need to be imported into the Power BI service – you cannot make a direct connection between the service and SQL Server. The more you understand how to use SQL Server data in Power BI Desktop, the better your data can be visualized in many different ways.