Analyzing and Visualizing Data with Microsoft Power BI

Exam Ref 70-778

Daniil Maslyuk

Microsoft

Exam Ref 70-778 Analyzing and Visualizing Data with Microsoft Power BI®

Daniil Maslyuk

Exam Ref 70-778 Analyzing and Visualizing Data with Microsoft Power BI

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

ISBN-13: 978-1-5093-0702-9
ISBN-10: 1-5093-0702-8

Library of Congress Control Number: 2018938487
1 18

Editor-in-Chief	Brett Bartow
Senior Editor	Trina MacDonald
Development Editor	Rick Kughen
Managing Editor	Sandra Schroeder
Senior Project Editor	Tracey Croom
Editorial Production	Backstop Media
Copy Editor	Liv Bainbridge
Indexer	Julie Grady
Proofreader	Katje Richstatter
Technical Editor	Chris Sorensen
Cover Designer	Twist Creative, Seattle

To my wife, Dasha, who was very patient and supported me during the writing process in every way she could.

—Daniil Maslyuk

Contents at a glance

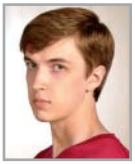
Contents

Acknowledgements

I would like to thank Trina MacDonald for handling the project and giving me the opportunity to write my first book, which turned out to be a very rewarding experience. Also, I would like to thank all the people who helped making the book more readable and contain fewer errors: Chris Sorensen, Rick Kughen, Liv Bainbridge, Troy Mott, and everyone else at Pearson who worked on this book but I haven't worked directly with.

A few people have contributed to my becoming a fan of Power BI. Gabriel Polo Reyes was instrumental in my being introduced to the world of Microsoft BI. Thomas van Vliet, my first client, hired me despite my having no prior commercial experience with Power BI and fed me many problems that led to my mastering Power BI.

About the author



DANIIL MASLYUK (MCSA: BI Reporting; MCSE: Data Management and Analytics) is a Microsoft business intelligence consultant who specializes in Power BI, Power Query, and Power Pivot; the DAX and M languages; and SQL Server and Azure Analysis Services tabular models. Daniil blogs at *xxlbi.com* and tweets as @DMaslyuk.

Introduction

The 70-778 exam focuses on using Microsoft Power BI for data analysis and visualization. About one fourth of the exam covers data acquisition and transformation, which includes connecting to various data sources by using Power Query, applying basic and advanced transformations, and making sure that data adheres to business requirements. Approximately half the questions are related to data modeling and visualization. Power BI is based on the same engine that is used in Analysis Services, and the exam covers a wide range of data modeling topics: managing relationships and hierarchies, optimizing data models, using What-if parameters, and using DAX to create calculated tables, calculated columns, and measures. The exam also covers selecting, creating and formatting visualizations, as well as bookmarks and themes. The remainder of the exam covers sharing data by using dashboards, reports, and apps in Power BI service. Furthermore, the exam tests your knowledge on managing custom reporting solutions, using Power BI Report Server, configuring security, and keeping your reports up to date.

This exam is intended for business intelligence professionals, data analysts, and report creators who are seeking to validate their skills and knowledge in analyzing and visualizing data with Power BI. Candidates should be familiar with how to get, model, and visualize data in Power BI Desktop, as well as share reports with other people.

This book covers every major topic area found on the exam, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the "Need more review?" links you'll find in the text to find more information and take the time to research and study the topic. Great information is available in blogs and forums.

Organization of this book

This book is organized by the "Skills measured" list published for the exam. The "Skills measured" list is available for each exam on the Microsoft Learning website: http://aka.ms/examlist. Each chapter in this book corresponds to a major topic area in the list, and the technical tasks in each topic area determine a chapter's organization. If an exam covers six major topic areas, for example, the book will contain six chapters.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to *http://www.microsoft.com/learning*.

Check back often to see what is new!

Microsoft Virtual Academy

Build your knowledge of Microsoft technologies with free expert-led online training from Microsoft Virtual Academy (MVA). MVA offers a comprehensive library of videos, live events, and more to help you learn the latest technologies and prepare for certification exams. You'll find what you need here:

http://www.microsoftvirtualacademy.com

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

https://aka.ms/examref778/errata

If you discover an error that is not already listed, please submit it to us at the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

Important: How to use this book to study for the exam

Certification exams validate your on-the-job experience and product knowledge. To gauge your readiness to take an exam, use this Exam Ref to help you check your understanding of the skills tested by the exam. Determine the topics you know well and the areas in which you need more experience. To help you refresh your skills in specific areas, we have also provided "Need more review?" pointers, which direct you to more in-depth information outside the book.

The Exam Ref is not a substitute for hands-on experience. This book is not designed to teach you new skills.

We recommend that you round out your exam preparation by using a combination of available study materials and courses. Learn more about available classroom training at *http://www.microsoft.com/learning*. Microsoft Official Practice Tests are available for many exams at *http://aka.ms/practicetests*. You can also find free online courses and live events from Microsoft Virtual Academy at *http://www.microsoftvirtualacademy.com*.

This book is organized by the "Skills measured" list published for the exam. The "Skills measured" list for each exam is available on the Microsoft Learning website: *http://aka.ms/examlist*.

Note that this Exam Ref is based on this publicly available information and the author's experience. To safeguard the integrity of the exam, authors do not have access to the exam questions.

CHAPTER 1

Consuming and transforming data by using Power BI Desktop

The Power BI development cycle is divided into four parts: data discovery, data modeling, data visualization, and distribution of reports. Each stage requires its own skill set. We cover data modeling and visualization skills in Chapter 2, "Modeling and visualizing data," and report distribution in Chapter 3, "Configure dashboards, reports, and apps in the Power BI Service." In this chapter, we review the skills you need to consume data in Power BI Desktop. Power BI has a rich set of features available for data shaping, which enables the creation of sophisticated data models. We start with the steps required to connect to various data sources. We then review the basic and advanced transformations available in Power BI Desktop, as well as ways to combine data from distinct data sources. Finally, we review some data cleansing techniques.

> ***IMPORTANT***
>
> ***Have you read page xxi?***
>
> **It contains valuable information regarding the skills you need to pass the exam.**

Skills in this chapter:

- Skill 1.1: Connect to data sources
- Skill 1.2: Perform transformations
- Skill 1.3: Cleanse data

Skill 1.1: Connect to data sources

Before you model or visualize any data, you need to prepare and load it into Power BI, creating one or more connections to data sources. Power BI can connect to a wide variety of data sources, and the number of supported data sources grows every month. Furthermore, Power BI allows you to create your own connectors, making it possible to connect to virtually any data source.

MORE INFO **DATA CONNECTORS IN POWER BI**

You can keep up with all the new Power BI features, including new connectors, on the official blog at *https://powerbi.microsoft.com/en-us/blog/*. For more details on how you can create your own data connectors, see "Getting Started with Data Connectors" at *https://github.com/Microsoft/DataConnectors*.

The data consumption process begins with an understanding of business requirements and data sources available to you. For instance, if your users need near real-time data, your data consumption process is going to be drastically different compared to working with data that is going to be periodically refreshed. Not all data sources support the near real-time experience, which is called DirectQuery, and comes with its own limitations.

This section covers how to:

- Connect to databases, files, and folders
- Import from Excel
- Connect to SQL Azure, Big Data, SQL Server Analysis Services (SSAS)

Connect to databases, files, and folders

Databases, files, and folders are some of the most common data sources used when connecting to data in Power BI. Power BI can connect to the following databases:

- SQL Server database
- Access database
- SQL Server Analysis Services database
- Oracle database
- IBM DB2 database
- IBM Informix database (Beta)
- IBM Netezza (Beta)
- MySQL database
- PostgreSQL database
- Sybase database
- Teradata database
- SAP HANA database
- SAP Business Warehouse database
- Amazon Redshift
- Impala
- Snowflake

- ODBC
- OLE DB

Power BI can also connect to the following file types:

- Excel
- Text/CSV
- XML
- JSON

Files can also be connected to in bulk mode through the following folder connectors:

- Folder
- SharePoint folder
- Azure Blob Storage
- Azure Data Lake Store

To connect to a data source, you need to click the Home tab and select Get Data in the External Data group. Clicking the text portion of the button opens a drop-down list with the most common data sources. When you click More in the drop-down list, the full Get Data window opens.

The window, shown in Figure 1-1, is divided into two parts: on the left, you can select data source types, which includes File, Database, Azure, Online Services, and Other. On the right, there is a list of data sources. Above the left pane, there is a search bar with which you can search for data sources.

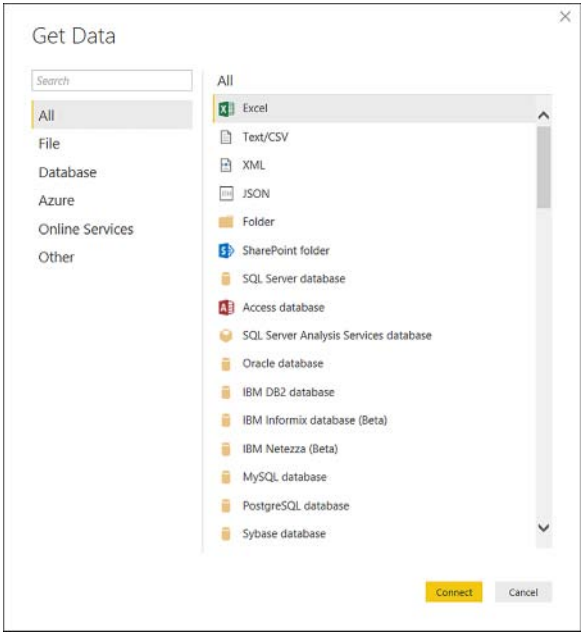


FIGURE 1-1 Get Data window

Before going any further, let's discuss the various data connection options that are available, because choosing one may prevent you from switching to the other after you start developing your data model.

Data connectivity modes

The most common way to consume data in Power BI is by importing it to the data model. When you import data in Power BI, you create a copy of it that is kept static until you refresh your dataset. Currently, data from files and folders can only be imported in Power BI. When it comes to databases, there are two ways in which you can make data connections. The two data connectivity options are shown in Figure 1-2.

FIGURE 1-2 An example data connection window with the option to choose between Import and DirectQuery

First, you can import your data into Power BI, which copies data into the Power BI data model. This method offers you the greatest flexibility when you model your data because you can use all available features in Power BI.

Second, you can connect to your data directly in its original source. This method is known as DirectQuery. With DirectQuery, data is not kept in Power BI. Instead, the original data source is queried every time you interact with Power BI visuals. Not all data sources support DirectQuery.

A special case of DirectQuery called Live Connection exists for SQL Server Analysis Services (both Tabular and Multidimensional), as well as the Power BI Service. We will cover LiveConnection in more detail later in this chapter.

Importing data

When you import data, you load a copy of it into Power BI. Because Power BI is based on an in-memory engine called VertiPaq (also known as xVelocity), the imported data consumes both the RAM and disk space, because data is stored in files. During the development phase, the imported data consumes the disk space and RAM of your development machine. Once you publish your report to a server, the imported data consumes the disk space and RAM of the server to which you publish your report. The implication of this is that you can't load more data into Power BI than your hardware allows.

You have an option to transform data when you import it in Power BI, limited only by the functionality of Power BI. If you only load a subset of tables from your database, and you apply filters to some of the tables, only the filtered data gets loaded into Power BI.

Once data is loaded into the Power BI cache, it is kept in a compressed state, thanks to the VertiPaq engine. The compression depends on many factors, including data type, values, and cardinality of the columns. In most cases, however, data will take much less space once it is loaded into Power BI compared to its original size.

One of the advantages of this data connection method is that you can use all of the functionality of Power BI without restrictions, including all transformations available in Power Query Editor, as well as all DAX functions when you model your data.

Additionally, you can use data from more than one source in the same data model. For example, you can load some data from a database and some data from an Excel file. You can then either combine them in the same table in Power Query Editor or relate the tables in the data model.

Another advantage of this method is the speed of calculations. Because the VertiPaq engine stores data in-memory in a compressed state, there is little to no latency when accessing the data. Additionally, the engine is optimized for calculations, resulting in the best computing speed.

DirectQuery

When you use the DirectQuery method, you are not loading any data into Power BI. All the data remains in the data source, except for metadata, which Power BI keeps. Metadata includes column and table names, data types, and relationships. For most data sources supporting DirectQuery, when connecting to a data source, you select the structures you want to connect to, such as tables or views. Each structure becomes a table in your data model. With some sources, such as SAP Business Warehouse, you only select a database, not specific tables or other structures.

With this method, Power BI only serves as a visualization tool. As a result, the Power BI file size will be negligible compared to a file with imported data.

At the time of this writing, only the following databases support DirectQuery connectivity.

- Amazon Redshift
- Azure HDInsight Spark (Beta)
- Azure SQL Database
- Azure SQL Data Warehouse
- Google BigQuery (Beta)
- IBM Netezza (Beta)
- Impala (version 2.x)
- Oracle Database (versions 12 and above)
- SAP Business Warehouse (Beta)
- SAP HANA
- Snowflake
- Spark (Beta) (versions 0.9 and above)
- SQL Server
- Teradata Database
- Vertica (Beta)

The main advantage of this method is that you are not limited by the hardware of your development machine or of the server to which you will publish your report. All data is kept in the data source, and all the calculations are done in the source as well. Using DirectQuery entails some implications to the available functionality.

Implications of using DirectQuery

There are a number of implications that occur when using DirectQuery.

Report performance varies

When using DirectQuery, the report performance depends on the underlying source hardware. If it can return queries in fewer than five seconds, then the experience is bearable, yet still might feel slow to users who are accustomed to the speed of the native VertiPaq engine. If the data source is not fast enough, the queries might even time out, making the report unusable. Whether the data source can handle the additional load from querying should also be considered. With DirectQuery, each visual a user interacts with sends a query to the data source, and this happens to every user who is working with a report at the same time.

Only one data source may be used at a time

DirectQuery can only use one data source at a time. Unlike importing data, it is not possible to combine data from multiple sources. For example, if you need to use a table from an Excel file in your report, you need to load it into the same data source that you are using.

Range of data transformations is limited

The range of data transformations that can be applied to data is limited with DirectQuery. For OLAP sources, such as SAP Business Warehouse, no transformations can be applied, and the entire model is used as a data source. For relational data sources, such as SQL Server, some transformations can still be applied, although they are quite limited due to performance considerations when compared to transformations available with imported data. The transformations need to be applied every time there is an interaction with a visual, not once per data refresh, as in the case of importing data. Only those transformations that can be efficiently translated to the data source query language are allowed. In case you try to apply transformations that are not allowed, you will get an error (Figure 1-3) and be prompted to either cancel the operation or import data.

FIGURE 1-3 Unsupported by DirectQuery transformation error

Not every query type is usable

Not every kind of query can be used in DirectQuery mode. When a user interacts with a visual in a report that uses DirectQuery, all of the necessary queries to retrieve the data are combined and sent to the data source. For this reason, it is not possible to use native queries with Common Table Expressions or Stored Procedures.

Data modeling is limited

The data modeling experience has its limitations in DirectQuery as well. Data modeling includes the creation of measures, calculated columns, hierarchies, and relationships; renaming and hiding columns; formatting measures and columns; defining default summarization and sort order of columns.

- By default, measures are limited only to those that are not likely to cause any performance issues. If you author a potentially slow measure, you will get an error like the following: "Function 'SUMX' is not supported in this context in DirectQuery mode." If you want to lift the restriction, click **File** > **Options and settings** > **Options** > **DirectQuery** > **Allow Unrestricted Measures In DirectQuery Mode**. This allows you to write any measure, given that it has a valid expression.
- With DirectQuery, there are no built-in date tables that are created for every date/datetime column like in Import mode by default. Date tables are required for Time Intelligence calculations, and if the data source has a date table, it can instead be used for Time Intelligence purposes.
- Calculated columns are limited in two ways. First, they can only use the current row of the table or a related row in a many-to-one relationship, which rules out all aggregation

functions. Second, calculated columns can use only some of the functions that return scalar values. More specifically, only functions that can be easily translated into a data source's native language are supported. For example, you can create a "Month Name" column in a Sales table with RELATED function, but you cannot count the number of rows in the Sales table for each row in the Date table in a calculated column because that would require an aggregation function COUNTROWS. Usually, IntelliSense, Microsoft's autocomplete feature, will list only the supported functions.

- Parent-child functions, such as PATH, are not supported in DirectQuery. If you need to create a hierarchy of employees or chart of accounts, consider building it in the data source.
- Calculated tables are not supported in DirectQuery mode. Consider creating a view in the data source in case you need a dynamic table.

Security limitations

There are security limitations to DirectQuery. Currently, when you publish a report that is using DirectQuery, it will have the same fixed credentials that you specify in Power BI service. This means that all users will see the same data unless the report is using the Row Level Security feature of Power BI.

Underlying data changes frequently

You should keep in mind that if the underlying data is changing frequently, there is no guarantee of visuals displaying the same data due to the nature of DirectQuery. To display the latest data, visuals need to be refreshed. Metadata, if changed in the source, is only updated after a refresh in the Power BI Desktop.

When to use DirectQuery

To get the best user experience, you should import data if you can. There are two situations in which you may consider DirectQuery over importing data.

First, if the size of the data model is too large to fit into memory, DirectQuery may be a viable option. You should keep in mind that performance will depend on the data source's hardware.

Second, if the underlying data changes frequently, and reports must always show the most recent data, then DirectQuery could be the solution. Again, the data source must be able to return the results in a reasonable amount of time. Otherwise there might not be a point in querying the latest data.

Both issues could potentially be addressed by Live Connection.

Live Connection

A special case of DirectQuery for SQL Server Analysis Services and Power BI service is called Live Connection. It differs from DirectQuery in some ways:

- It is not possible to define relationships in Live Connection.
- You cannot apply any transformations to data.
- Data modeling is limited to only creating measures for SQL Server Analysis Services Tabular and Power BI service. The measures are not restricted in any way.

You may consider using Live Connection over importing data because of the enhanced data modeling capabilities and improved security features in the data source. More specifically, unlike DirectQuery, Live Connection considers the username of the user that is viewing a report, which means security can be set up dynamically. Additionally, SQL Server Analysis Services can be configured to refresh as frequently as needed, unlike a Schedule Refresh in Power BI service that is limited to eight times a day on a Pro license and 48 times a day with Power BI Premium.

Table 1-1 summarizes the similarities and differences between three data connectivity modes.

TABLE 1-1 Data connectivity modes compared

Category	Import data	DirectQuery	Live Connection
Data model size limitation	Tied to license; Power BI Pro: 1 GB limit per dataset; Power BI Premium: capacity based	Limited only by underlying data source hardware	SQL Server Analysis Services: Limited only by underlying data source hardware; Power BI service: same dataset size limits as Import Data
Number of data sources	Unlimited	Only one	Only one
Data refresh	Tied to license; Power BI Pro: up to 8 times a day at 30 min intervals; Power BI Premium: up to 48 times a day at 1 min intervals	Report shows the latest data available in the source	Report shows the latest data available in the source
Performance	Best	Slowest	Best
Data transformation	Fully featured	Limited to what can be translated to data source language	None
Data modeling	Fully featured	Highly restricted	SSAS Tabular and Power BI Service: measures can be created without restrictions
Security	Row-level security can be applied based on current user login	Cannot use row-level security defined at data source; Row-level security must be done in Power BI Desktop	Can leverage data source security rules based on current user's login

NOTE **DIRECTQUERY IN POWER BI**

For more details on the advantages and limitations of DirectQuery in Power BI, see "Power BI and DirectQuery" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-directquery-about/*.

EXAM TIP

Be prepared to answer when DirectQuery or Live Connection data connectivity modes are appropriate based on a client's business requirements.

Connecting to Microsoft SQL Server

To connect to Microsoft SQL Server, click **Get Data** > **SQL Server Database**. You will then see the window in Figure 1-2. In it, you must specify a server name, and you have an option to specify a database name. You must then choose between Import and DirectQuery data connectivity modes.

If you expand Advanced options, you can specify a custom timeout period in minutes and a SQL statement to run. If you write a SQL statement, you must specify a database.

Below the SQL statement input area, there are three check boxes.

- **Include relationship columns** First, you can include or exclude the relationship columns. This option checks if a table has any relationships with other tables and includes expandable relationship columns in Power Query Editor. This might be useful if you want to denormalize your data and save an extra step of merging tables in Power Query Editor. The default selection is **include**.
- **Navigate using full hierarchy** Second, you can enable or disable navigation with a full hierarchy. With the option enabled, you can navigate from the server down to databases, then schemas, and finally objects within schemas. With the option disabled, you navigate from the server to the databases, and then all objects from all schemas. The default selection is **disable**.
- **Enable SQL Server Failover support** Third, you can enable or disable SQL Server Failover support. With this option enabled, you can benefit from local high availability through redundancy at the server-instance level by leveraging Windows Server Failover Clustering. The default selection is **disable**.

Once you click **OK** to connect, the credentials window opens, and you have two options for authentication: **Windows** and **Database**. If you choose **Windows**, you can either select to use the current user credentials or specify alternate credentials. After you specify credentials and click **Connect**, you might get a prompt on Encryption Support; you can click **OK** to connect without encryption.

The Navigator window then opens, where you can choose objects to add to the data model. The window, which can be seen in Figure 1-4, is divided into two parts. On the left side, you

see a list of all the objects you can choose. For SQL Server, you can choose tables, views, scalar functions, and table functions. Note that you cannot select stored procedures, even if they return tables. When you select an item, you can click **Select Related Tables** if you want to select all tables that are related to the selected table.

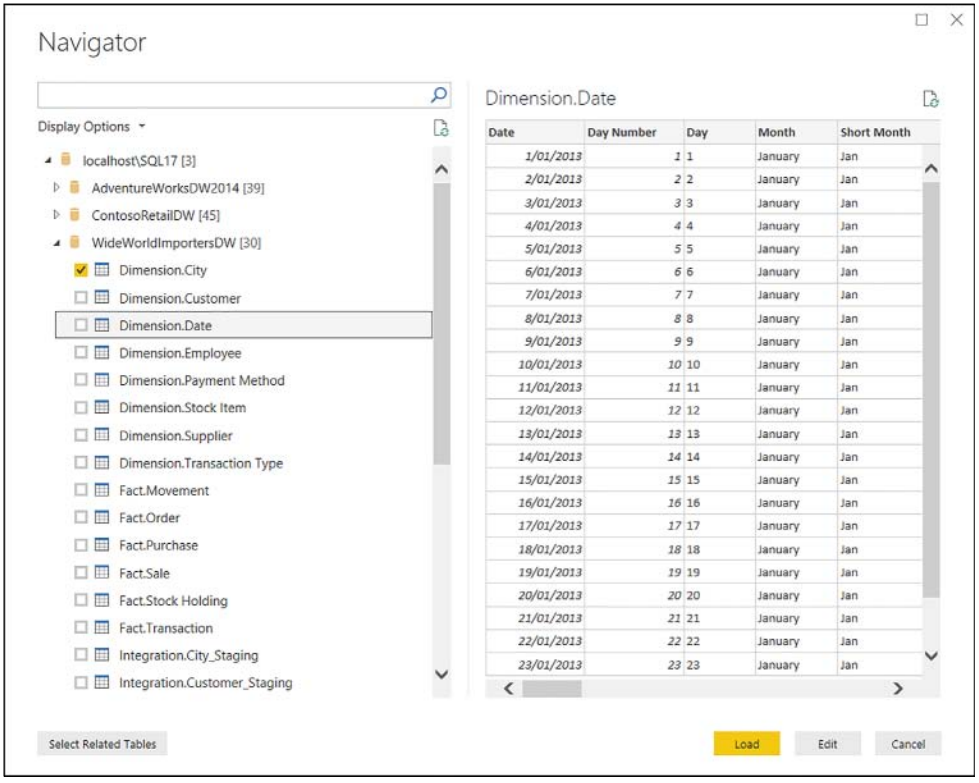


FIGURE 1-4 The Navigator window

Selecting an object brings up a preview of data inside the object. If you select a function for preview, you will need to specify one or more parameters to see a data preview. Note how you are not limited to choosing objects from one database only (unless you specified a database in the initial connection settings).

After selecting the desired objects, you can either load data directly to the data model without any transformations by clicking **Load**, or you can apply transformations in Power Query Editor by clicking **Edit**. If you choose the latter option, you will then need to click **Close & Apply** to load data into Power BI data model.

If you select objects from more than one database, Power BI will create a connection string for each database. You can access the list of connections either from the **Home** tab; **External Data** group in the main Power BI window by clicking on the text portion of

Edit Queries > **Data Source Settings;** or from the Power Query Editor by clicking the **Home** tab and selecting **Data Source Settings** in the **Data Sources** group.

When you load data, Power BI shows the activities associated with each query, such as:

- Evaluating
- Waiting for other queries
- Creating connection in model
- Loading data to model
- Detecting relationships

If one of the queries fails, other queries will not load. After data loading is finished, each query appears as a table with columns in the Fields pane.

Power BI supports connections to SQL Server starting with SQL Server 2005.

Connecting to Access database

To connect to the Access database, select **Get Data** > **Access Database**. You will then be prompted to specify the database file in the Open window. Note that you can open the file in read-only mode if necessary. After you select the file and click **Open**, a Navigator window comes up with the list of available objects.

You can then select the objects you want to include in your data model. Afterward, you can either load the objects directly into the data model by clicking **Load** or apply transformations by clicking **Edit**. If you click **Edit**, you will be able to click on the cog wheel next to the **Source** step in **Query Settings**. Doing so opens a window where you can specify advanced settings (Figure 1-5).

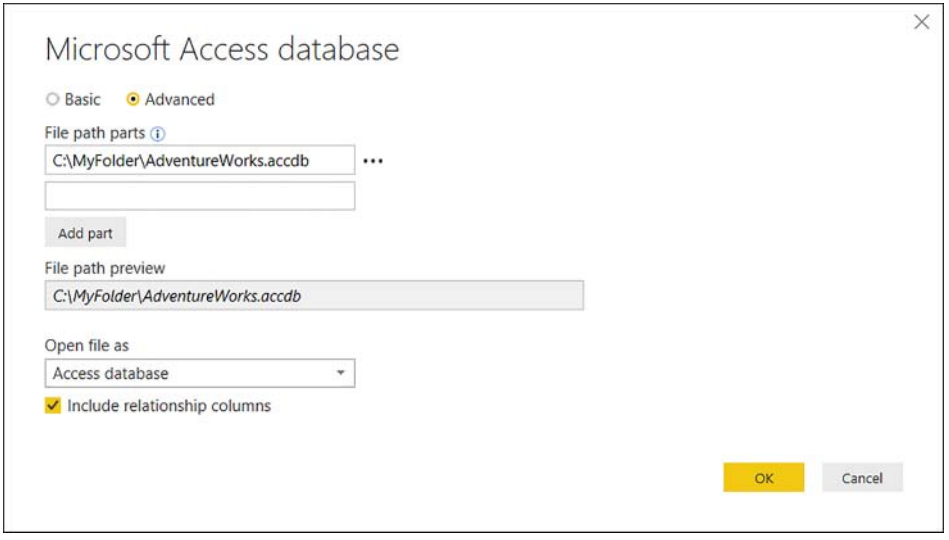


FIGURE 1-5 Advanced Access database connection settings

In advanced settings, you can choose whether you want to include relationship columns in tables, which are included by default. You can also compose the file path from parts. Each part can contain a fixed subset of the file path or reference to a parameter.

Power BI supports connections to all versions of Access database files, except password-protected ones. The provider version, however, needs to be at least ACE 2010 SP1.

> ***NOTE*** **INSTALLING THE NECESSARY DRIVERS**
>
> **If you lack the necessary drivers, you may see an error message similar to the following: "DataSource.NotFound: Microsoft Access: The 'Microsoft.ACE.OLEDB.12.0' provider is not registered on the local machine. The 64-bit version of the Access Database Engine 2010 Access Database Engine OLEDB provider may be required to read 'AdventureWorks.accdb.'"**
>
> **The required software can be downloaded from Microsoft at *https://www.microsoft.com/en-us/download/confirmation.aspx?id=13255*.**

Connecting to an Oracle database

To connect to an Oracle database, select **Get Data** > **Oracle Database**. If you are connecting to an Oracle database for the first time, you might see a message indicating your provider is out of date, and you might want to consider upgrading it. For the connection to be successful, you need to have the correct Oracle client software installed, depending on the version of Power BI Desktop you are running—32-bit or 64-bit. To find out which version you have, select **File** > **Help** > **About**, then look at the Version line.

> ***NOTE*** **INSTALLING THE CORRECT ORACLE CLIENT SOFTWARE**
>
> **If you need 32-bit Oracle client software, you can download it from Oracle at *http://www.oracle.com/technetwork/topics/dotnet/utilsoft-086879.html*. In case you need 64-bit software, you can also download it from Oracle at *http://www.oracle.com/technetwork/database/windows/downloads/index-090165.html*.**

Once you open the initial connection settings window (Figure 1-6), the experience is very similar to SQL Server connection settings. There are only two differences at this stage: first, you cannot specify a database to connect to. And second, there is no option to enable SQL Server Failover support. If you need to specify SID in addition to the server name, you can specify it with a forward slash after the server name. For example, **ServerName/SID**.

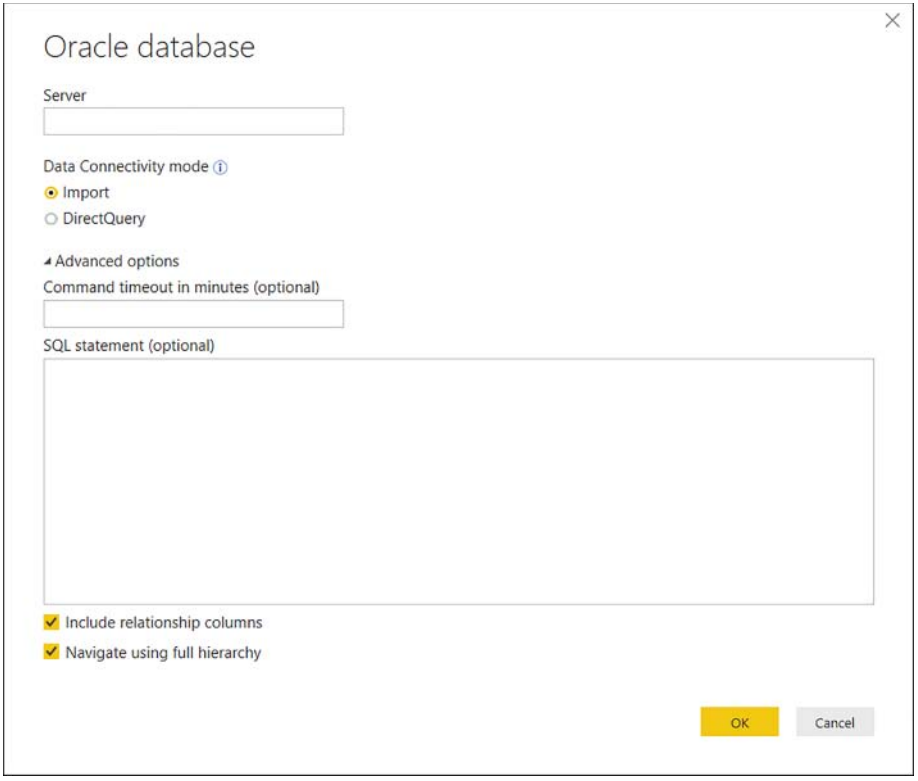


FIGURE 1-6 Oracle database connection settings window

Once you specify the required parameters and click **OK**, you are taken to the credentials window. You have the same options as with SQL Server: either **Windows** or **Database**; for the former, you can either use the current user's credentials or specify alternate credentials.

After you specify the credentials, the Navigator window opens, where you can choose the objects for inclusion in the data model. If you chose to navigate using full hierarchy, the schemas appear with folder icons in the Navigator window. In an Oracle database connection, only tables and views can be selected.

Finally, you have an option of loading the database objects right away by clicking **Load**; if you wish to apply transformations before loading, you will need to click **Edit**, which will take you to the Power Query Editor.

Power BI supports connections to Oracle databases starting with Oracle 9; the provider needs to be running at least version ODAC 11.2 Release 5.

Connecting to a MySQL database

To connect to a MySQL database, select **Get Data** > **MySQL Database**. If it's the first time you are connecting to a MySQL database, you will likely need to install the latest data provider for MySQL, called Connector/Net. After installing it, you should restart Power BI Desktop for the update to take effect.

> ***NOTE*** **DOWNLOADING MYSQL DATA PROVIDER**
>
> **You can download the latest Connector/Net data provider for MySQL from the official MySQL website at *https://dev.mysql.com/downloads/connector/net/*.**

Once you open the initial connection settings window, you will need to specify both the server and database names. There is no option to choose DirectQuery when connecting to MySQL, because the latter only supports the Import data connectivity mode. The advanced options are the same as SQL Server's name, sans the option to enable SQL Server Failover support.

After you click **OK**, you are taken to the credentials window. MySQL supports Windows authentication, and you can either use the current user's credentials or specify alternate ones. You also have an option to use Database authentication mode. Clicking **Connect** might prompt a note saying the connection will be unencrypted. If you click **OK**, you will be taken to the standard Navigator window. If you enabled full hierarchy navigation, the schemas would appear with folder icons. With MySQL connections, you can choose tables, views, and scalar functions to include in your data model. You can then proceed with loading the data, with an option of applying transformations to it in Power Query Editor by clicking **Edit**.

Power BI supports connections to MySQL databases starting with MySQL 5.1; the data provider needs to be running version 6.6.5 at a minimum.

Connecting to PostgreSQL database

To connect to a PostgreSQL database, select **Get Data** > **PostgreSQL Database**. If you are connecting to a PostgreSQL database for the first time, you might get an error message prompting you to install "one or more additional components."

> ***NOTE*** **DOWNLOADING POSTGRESQL DATA PROVIDER**
>
> **You can download the latest Npgsql, the .NET data provider for PostgreSQL, from its official GitHub repository at *https://github.com/npgsql/Npgsql/releases*.**

When installing Npgsql, make sure to select **Npgsql GAC Installation** (Figure 1-7). Otherwise, the data provider might not function correctly.

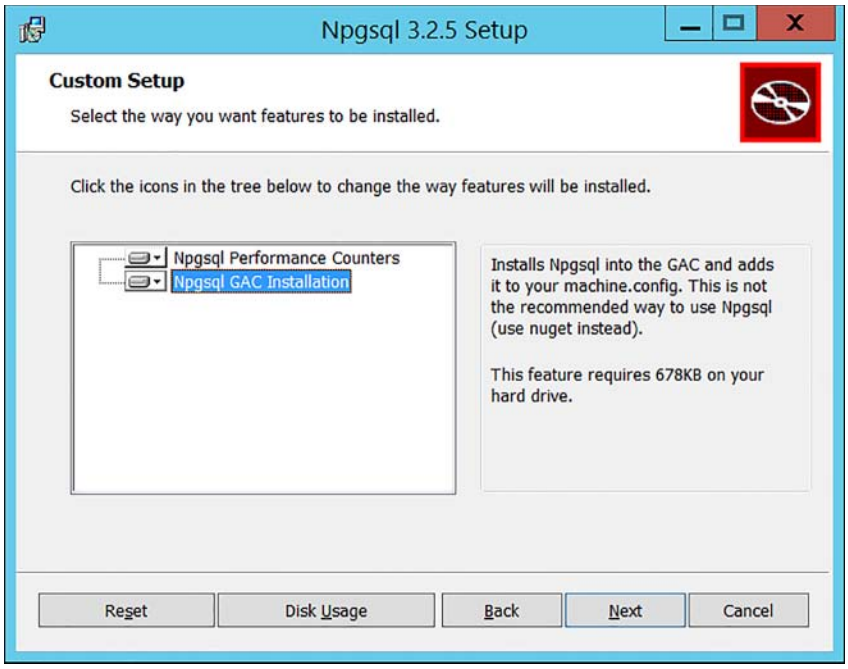


FIGURE 1-7 Npgsql setup features selection window.

Once you have the data provider installed correctly, you will need to restart Power BI, and then you can start the connection setup. In the initial PostgreSQL database connection window, you will need to specify both the server and database names. There is no option to choose between Import and DirectQuery, as PostgreSQL currently does not support DirectQuery. The advanced settings are the same as the ones for the MySQL connection: you can specify a custom connection timeout in minutes and a native database query; you can also elect to include the relationship columns and navigate using the full hierarchy.

Once you specify the connection settings, you will be prompted to enter connection credentials. For PostgreSQL connections, you can only use database credentials. After you enter the credentials, you will see the standard Navigator window. If you chose to navigate using full hierarchy, the database schemas will appear with folder icons. In PostgreSQL, you can only select tables and views to include in your data model. Once you choose the desired objects, you can either load the data by clicking **Load** or transform it before loading by clicking **Edit**.

Power BI supports connections to PostgreSQL starting with PostgreSQL 7.4; the Npgsql.NET provider needs to be at least version 2.0.12.

Connecting to data using generic interfaces

Apart from using built-in connectors that are specific to their data sources, Power BI allows you to connect to other data sources with generic interfaces. These methods can also be useful in cases where built-in connectors do not work properly. Currently, Power BI supports the following generics interfaces:

- ODBC
- OLE DB
- OData
- REST APIs
- R Scripts

For these connectors, you need to specify your own connection strings. You might also need to install additional software for the connectors to work. For example, to run R scripts, you need to install R locally on your machine. To connect to a PostgreSQL database through ODBC, you will need to download the ODBC driver for PostgreSQL. The exact details on how to connect to any data source are specific to each data source and are outside of the scope of this book.

> ***MORE INFO*** **CONNECTING TO DATA WITH GENERIC INTERFACES IN POWER BI DESKTOP**
>
> **By using generic interfaces Power BI, you can greatly increase the list of data sources to which you can connect. For more details on working with generic interfaces, see "Connect to data using generic interfaces in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-connect-using-generic-interfaces/*.**

Connecting to Text/CSV files

To connect to a Text or CSV file, select **Get Data**, **Text/CSV**. You will then need to select your file in the standard Open window. Choosing the file and clicking **Open** takes you to the next screen (Figure 1-8), where you see a preview of your data, along with the settings.

Power BI automatically determines the file encoding, delimiter type, and how many rows should be used to detect the data types in the file. You can change these settings using the drop-down options if need be.

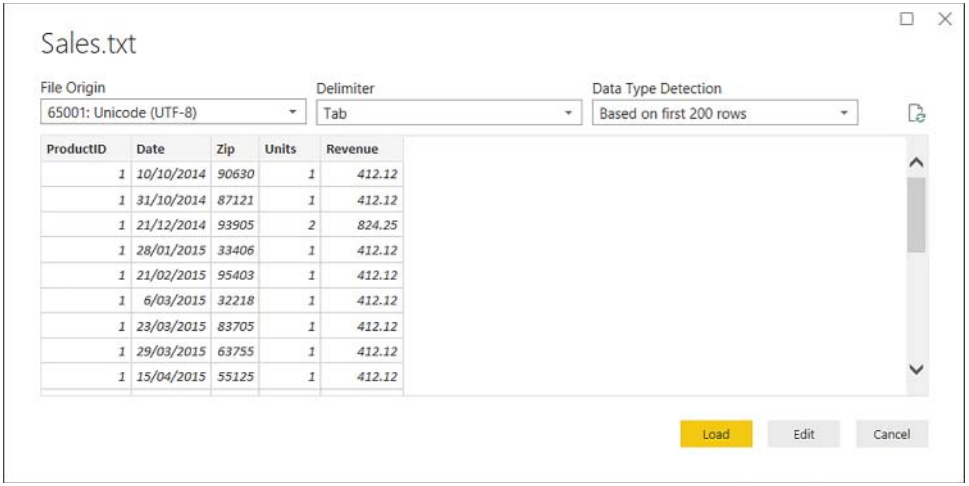


FIGURE 1-8 The Text/CSV file settings and data preview window

After you make sure the settings look correct to you, you can either click **Load** to load data directly into Power BI, or you can click **Edit** and apply further transformations to data in Power Query Editor.

Connecting to JSON files

To connect to a JSON file, you need to select **Get Data** > **JSON**. After selecting the file and clicking **Open**, you will be taken directly to Power Query Editor. To extract the data from your JSON file, you will likely need to perform various transformations, depending on the structure of your file. Figure 1-9 shows an imported JSON file that contains two tables within. The tables have different structures.

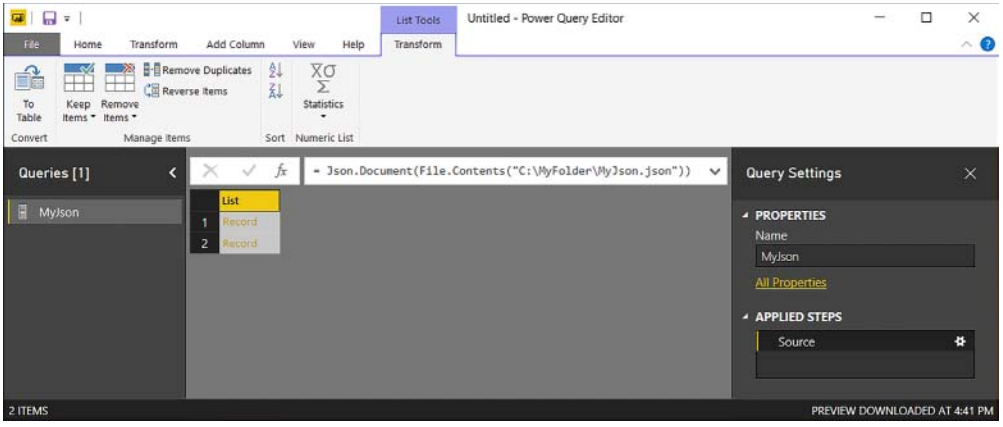


FIGURE 1-9 Power Query Editor after opening a JSON file

If you want to extract the data from your JSON file, you can either transform the starting list to a table by clicking the **Transform** tab and selecting **To Table** in the **Convert** group, or you can drill down into a specific record by clicking on a specific **Record** link. If you would like to see a preview of data in a record, you can click on its cell without clicking on the link, which will open a data preview pane at the bottom of Power Query Editor.

Clicking on the cog wheel next to the **Source** step in Query Settings opens a window where you can specify advanced settings. Among other things, you can specify file encoding in the **File Origin** drop-down list. Once you are done with transformations, you can click **Close & Apply** to load data into Power BI data model.

Connecting to XML files

To connect to an XML file, select **Get Data** > **XML**. Unlike JSON files, XML files have a structure that can be parsed by Power BI Desktop. Once you select the file you want opened in the Open window, you are taken to a Navigator window (Figure 1-10), where you see the structure of the file.

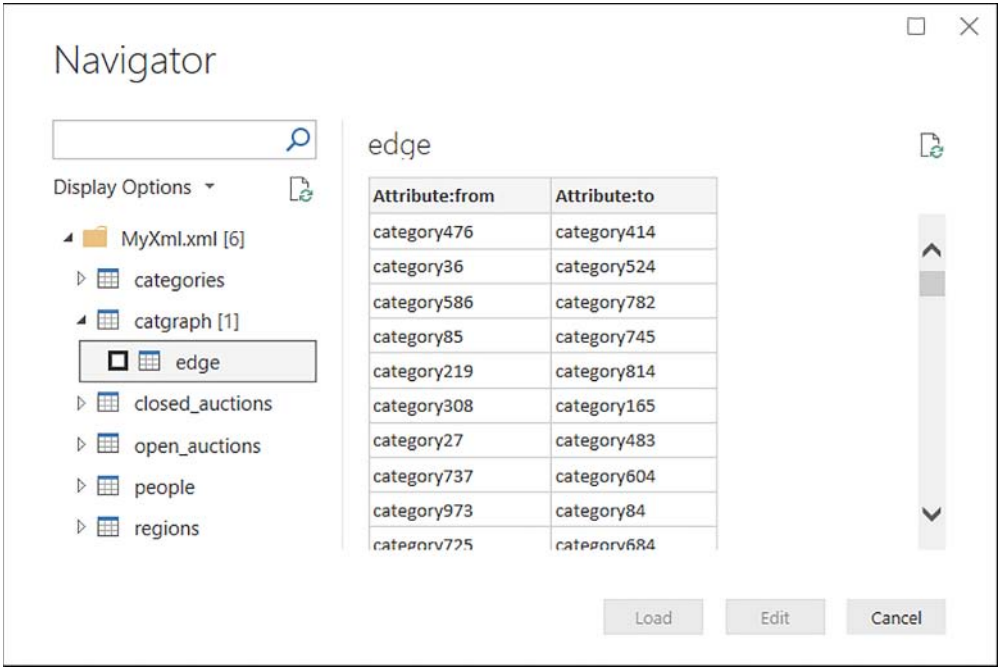


FIGURE 1-10 Contents of a sample XML file

After selecting the items that you want to import to your data model, you can click **Load**, which will load the data to Power BI cache as-is. Alternatively, you can click **Edit**, and it will open the Power Query Editor window for you to apply transformations to your data. In Power Query Editor, you can click on the cog wheel next to the **Source** step to open the advanced file

settings, where you can specify file encoding if need be. Clicking the **Home** tab and selecting **Close & Apply** in the **Close** group will load the data to the data model.

Connecting to a Folder

If you have several files that share the same structure, you can import them one by one, applying the same transformations, and then append them together in Power Query Editor. There is one significant problem with this approach: it is time-consuming. There is a more efficient way: instead of importing the files individually, you can connect to the folder that contains them.

To connect to a folder, select **Get Data** > **Folder**. You will be prompted to specify the folder path, which you can do either by clicking **Browse** and navigating to the folder in the Browse For Folder window, or you can paste the folder path. Once you click **OK**, a new window (Figure 1-11) opens where you see a list of files in the folder in binary format in the **Content** column, along with their attributes. These attributes include:

- Name
- Extension
- Date accessed
- Date modified
- Date created
- Attributes
- Folder Path

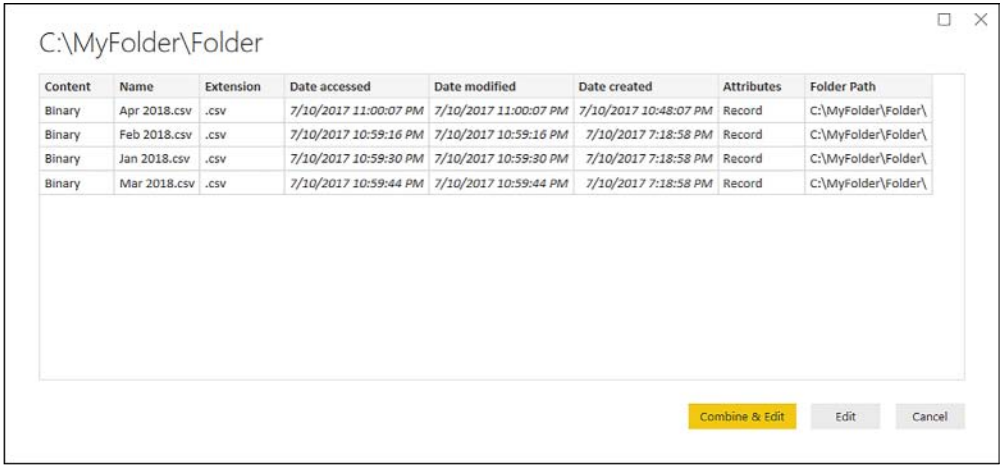


FIGURE 1-11 Folder preview window

At this stage, you have two options to continue: you can either select **Combine & Edit** or just **Edit**. Clicking the latter brings you to Power Query Editor with the starting point that is the same as the data preview.

If you click **Combine & Edit**, however, the Combine Files window opens, where you can specify settings under which files should be combined.

The first thing you can choose is an example file. By default, the first file is the example file. Alternatively, you can choose a specific file. The implication of choosing a certain file is that the query might break if this file is later renamed, moved, or deleted.

The other settings that you can specify depend upon the type of the files you are combining. For Text/CSV files, for example, you can choose the same options as for an individual CSV file—file origin (encoding), delimiter type, and the number of rows used for data type detection. You also have an option to skip files with errors. For Excel files, a Navigator window opens, where you can choose one object to consolidate from each file. The selected object needs to be of the same name and type across files.

After specifying the relevant settings, clicking **OK** creates several objects in Power Query Editor, which you can see in the Queries pane in Power Query Editor (Figure 1-12).

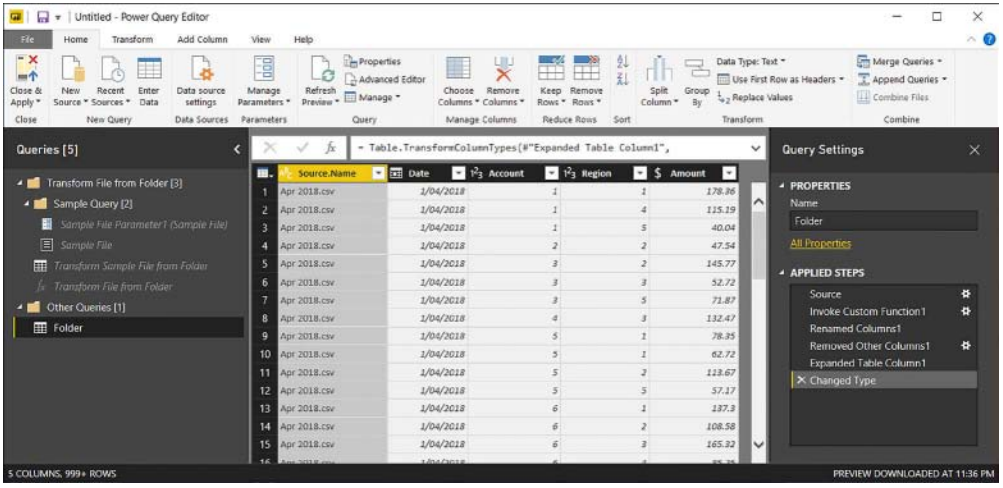


FIGURE 1-12 Objects created after combining files

Based on the sample file, Power BI decides which transformations should be applied to each file. For example, if you are combining text files with headers, then the latter should be used as column names. The transformations are combined into a custom function, which is then applied to each file from the folder to which you are connecting. Auxiliary objects, the parameter, and the binary files are created as well.

If you add or remove files in the folder later, you can click **Refresh**, and all the data will be reloaded without any manual intervention. For the folder connector to work correctly, however, it is very important that all of the files share the same structure.

NOTE **COMBINING BINARIES IN POWER BI DESKTOP**

With the Folder data source, you are not limited to combining CSV or text files; you can also combine Excel, JSON, and other types of files. For more details on the functionality, see "Combine binaries in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-combine-binaries/*.

Connecting to a SharePoint folder

The process of connecting to a folder in SharePoint is similar to connecting to a local folder, except for the initial connection window. Once you click **Get Data** > **SharePoint Folder**, you need to specify site URL, which is the root SharePoint site URL path, excluding any subfolders.

After you click **OK**, you are then taken to the credentials window. You have three options to choose from: **Anonymous**, **Windows**, and **Microsoft Account**. As usual, if you choose to use Windows credentials, you can either use the current user's credentials or alternate credentials. Choosing **Microsoft Account** prompts you to sign into your account in the window.

The experience that follows the credentials window is identical to the process of connecting to a local folder.

Connecting to web pages and files

With Power BI Desktop, you can get data from web pages. To review the functionality, you can connect to a Wikipedia article called "List of states and territories of the United States" at *https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States*. Note that because Wikipedia's nature, the information on the page might change without notice, and what you see may differ slightly from figures in this chapter. But the overall process will be the same.

Let's start by clicking **Get Data** > **Web**. The only required parameter is a URL. The advanced options allow you to compose a URL from parts, specify a custom timeout period in minutes, as well as add one or more HTTP request header parameters. When you connect to a web page for the first time, you can select from five authentication methods:

- Anonymous
- Windows
- Basic
- Web API
- Organizational account

Because Wikipedia is a publicly available website, you can choose **Anonymous**. After clicking **Connect**, you are taken to the Navigator window. The window is split into two parts as usual: a list of objects on the left and data preview on the right. In the list of objects, the first

object is **Document**; the rest are tables that Power BI found on the page. Handling web page objects that are not tables is less straightforward: for this, you need to navigate through HTML tags, and this is outside of scope of this book.

> ***MORE INFO*** **WEB SCRAPING IN POWER BI**
>
> **For information on how you can navigate through HTML tags with Power Query, see the article by Gil Raviv, "Web Scraping in Power BI and Excel Power Query" at *https://datachant.com/2017/03/30/web-scraping-power-bi-excel-power-query/*. The data preview has two tabs: Table View and Web View. When you select an object on the left, you can see the way it will appear in the Power Query Editor once you click Edit; if you switch to Web View, you will see the object the way it appears on the web. You can also select tables by ticking check boxes in Web View. You can see the Navigator window in Figure 1-13.**

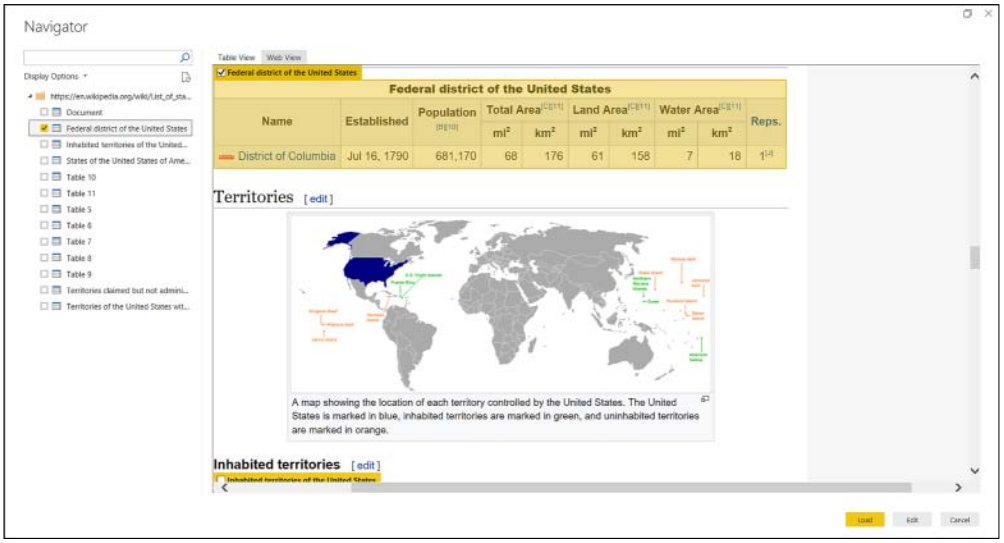


FIGURE 1-13 Navigator window when connecting to a web page

If one or more cells in a table are merged, the content is repeated for every cell once you bring the data to Power Query Editor. After selecting one or more objects, you can load the data directly to the data model, or edit it and then load.

The same connector, Web, can also be used to connect to files, such as Excel, text, JSON, XML, and others located on the Internet by specifying a URL.

NOTE **CONNECTING TO FILES IN ONEDRIVE FOR BUSINESS**

It is possible to connect to files from OneDrive for Business, and you can use either an individual or group account for this. To connect to a file from OneDrive, you will need its link, which you can generate in OneDrive. You will then need to click Get Data > Web, and paste the link. Note that you need to remove the "?web=1" porting of the URL so that Power BI can access your file directly. For more details on how to use OneDrive as a data source, including scheduling refresh, see "Use OneDrive for Business links in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-use-onedrive-business-links/*.

Connecting to Azure Data Lake Store and Azure Blob Storage

You can also connect to a folder located in Azure Data Lake Store, but the process is slightly different compared to a local folder and SharePoint folder.

Once you click **Get Data** > **Azure Data Lake Store**, you will need to specify the folder path starting with ad://. Note that the path does not need to be the root path; it can be a specific folder as well as a file.

With Azure Data Lake Store, the only authentication option is an organizational account. After you specify the credentials, a folder preview opens, but there is no **Combine & Edit** button; the only options are **OK** and **Cancel**. Clicking **OK** is the same as clicking **Edit**; doing so takes you to the Power Query Editor. You can still combine your files automatically, leveraging the same mechanism that works for local and SharePoint folders. To do this, you need to click the double arrow next to the **Content** column, which is the left-most column (Figure 1-14).

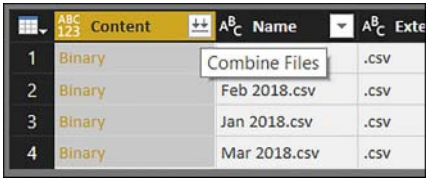


FIGURE 1-14 Combine Files button

Clicking the **Combine Files** button opens the Combine Files window with settings relevant to the type of the files. Clicking **OK** creates the following auxiliary objects to combine files: a parameter, a binary file reference, a processed sample file, a custom function, and the final combining query.

The process of connecting to Azure Blob Storage is identical to an Azure Data Lake Store connection, except you need to select **Get Data** > **Azure Blob Storage** first, and then specify account name or URL. Even if your containers have folders inside, the structure will be flattened so you can see all the files inside your containers.

Import from Excel

Power BI can work with Excel files in two distinct ways:

- Import data
- Import workbook contents

Importing data only gives you the raw data from Excel while importing workbook contents imports Power Query queries, Power Pivot data model, and Power View worksheets.

Import data from Excel

To connect to an Excel file, select **Get Data** > **Excel**. In the following **Open** dialog window, navigate to your file and click **Open**.

Power BI then opens the Navigator window (Figure 1-15), which presents Excel sheets, tables, and named ranges in the left pane. Every item type has its own icon. If you select an item in the left pane, a preview of its data will appear on the right.

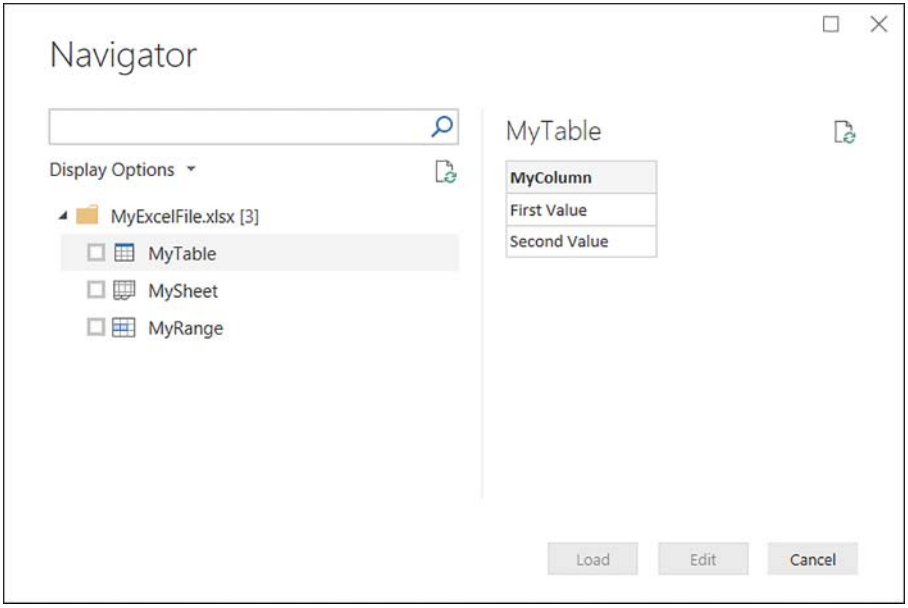


FIGURE 1-15 Navigator window when connecting to Excel

Once you select items to import, you can either load them into the data model right away by clicking **Load**, or you can edit them before loading by clicking **Edit**. The latter option opens Power Query Editor, where you can apply transformation to your data. To load the data after editing it, click **Close & Apply** in the Power Query Editor window.

Import Excel workbook contents

To import Excel workbook contents, select **File** > **Import** > **Excel Workbook Contents**. Select your file in the **Open** dialog window that follows.

You will then see a message stating that a new Power BI Desktop file will be made for you, which will retain as much useful content as possible. This means that Power BI Desktop imports Power Query queries, Power Pivot data models, and Power View worksheets as long as it supports the elements inside them. You can then click **Start** to import the workbook contents. You will then see the Import Excel workbook contents window (Figure 1-16).

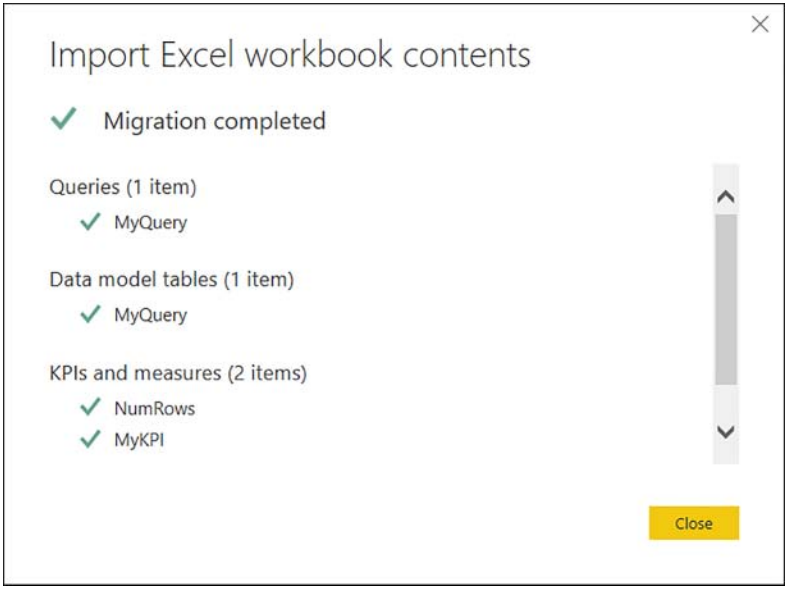


FIGURE 1-16 Import Excel workbook contents status window

If your queries contain links to the data from Excel sheets in the same workbook (obtained by clicking **Data** > from **Table** or **Excel.CurrentWorkbook M Function**, you will have a choice of either copying the data or keeping the connection to the Excel file.

Copying the data creates a copy of it in the query in the form of a compressed JSON document. You can edit it later the Power Query Editor by clicking the cog wheel next to the **Source** step in the Query Settings pane on the right.

The option **Keep Connection**, instead of copying the data, keeps the dependency on the Excel file, meaning the file is referenced with a full file path.

IMPORTANT **IMPORTING POWER VIEW SHEETS**

Not all visuals from Power View can be imported in Power BI because some Power View visuals have no corresponding visuals in Power BI. For example, in Power View, you use horizontal or vertical multiples in a pie chart. In Power BI, there is no such option. When such a visual is imported to Power BI, you will get a placeholder visual with the following error message: "This visual type is not yet supported." You will receive this error message for each unsupported visual imported from an Excel file.

NOTE **IMPORTING EXCEL WORKBOOK CONTENTS**

The best way to migrate a Power Pivot data model to Power BI is by importing Excel workbook contents. For more details on the process, see "Import Excel workbooks into Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-import-excel-workbooks/.*

Connect to SQL Azure, Big Data, SQL Server Analysis Services (SSAS)

In some cases, importing data into Power BI may not be a viable option due to its volume, change frequency, or other reasons. In these cases, you can connect to data sources that already have data models in them that can be easily consumed in Power BI in either DirectQuery or Live Connection mode.

Connecting to Azure SQL Database and Azure SQL Data Warehouse

Both Azure SQL database and Azure SQL Data Warehouse have their own connection options in the Get Data window. The data connection experience, however, is identical to that of SQL Server. Furthermore, the same two functions are used to connect to all three data sources: Sql.Database in case you connect to a specific database or Sql.Databases if you do not specify a database name.

To connect, you need to specify a fully qualified name of your server, which you can find in Azure Portal. Usually, it is in the following format <name>.database.windows.net.

The only limitation to be aware of is even though you are given an option of authenticating using Windows credentials, neither Azure SQL Database nor Azure SQL Data Warehouse currently support this option, leaving only **Database Authentication Mode** available.

Additionally, you should make sure that firewall rules for the database you are connecting to are configured properly.

NOTE **CONFIGURING AZURE SQL DATABASE FIREWALL RULES**

Information on how to configure firewall rules for Azure SQL Database can be found on the Microsoft Docs website at *https://docs.microsoft.com/en-us/azure/sql-database/sql-database-firewall-configure*.

Connecting to Azure HDInsight Spark

To connect to Azure HDInsight Spark, click **Get Data** > **Azure** > **Azure HDInsight Spark (Beta)** and click **Connect**. Because this is a preview connector, you will get a warning message saying that it might not work in the same way in the final version, and future changes may cause your queries to become incompatible.

After clicking **Continue**, you will be prompted to enter the server name. You can get the server name from Azure Portal, and usually, it is in the following format: *https://<name>.azurehdinsight.net*. The only other choice you will need to make is between **Import** and **DirectQuery connectivity** modes.

Clicking **OK** takes you to credentials window, with the only authentication option being username and password. Once you specify the credentials, you will be taken to a standard Navigator window, where you see the tables in your Spark server and a data preview pane.

Connecting to SQL Server Analysis Services (SSAS)

Power BI Desktop supports two data connectivity modes with SQL Server Analysis Services (SSAS): Import and Live Connection. As explained above, Live Connection is a special case of DirectQuery. To connect to SSAS from Power BI Desktop, click **Get Data** > **SQL Server Analysis Services Database** > **Connect**.

You will then see the initial connection settings window, where you need to specify the server name. Optionally, you can also enter a port number with a colon following the server name—for example, **localhost:1234**. You can specify a database name, or you can select it later. By default, **Connect Live** is selected instead of **Import**. If you select **Import**, you will have an option to write a custom MDX or DAX query. Clicking **OK** takes you to the authentication window, where you have three options: **Windows**, **Basic**, and **Microsoft Account**. After you specify your credentials, you are taken to the Navigator window.

If you select **Connect Live**, you are prompted to choose a model or perspective from your database. Clicking **OK** would create a live connection to the database.

If you selected **Import** in the initial connection settings, the Navigator window (Figure 1-17) lets you build a table using attributes and measures from a model that you select. If you have more than one model in your database, you will only be able to use one at a time. Once you are finished building a table, you can either load the data right away by clicking **Load** or apply further transformations to it in Power Query Editor by clicking **Edit**.

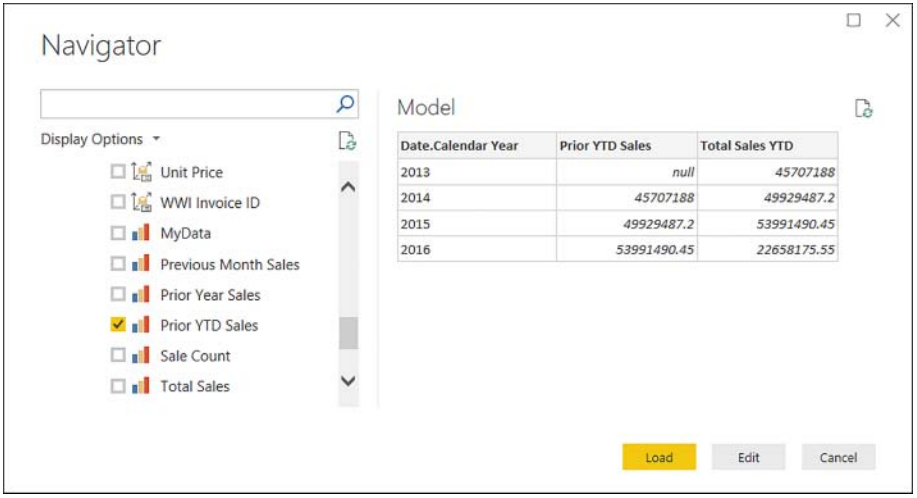


FIGURE 1-17 Navigator window after selecting Import

While Import behaves the same way with SSAS as with other data sources, Live Connection is different. The first notable difference is that there are no **Data** and **Relationships** buttons in the main Power BI Desktop window on the left; you can only use the Report view. It is not possible to view the underlying data or modify it in any way. However, if you are using a Tabular model, you can create report-level measures and Quick Measures in your report. These measures would not be added to the data source. Instead, they will be kept in the report only.

NOTE **POWER BI DESKTOP AND ANALYSIS SERVICES**

While Power BI supports almost all the features of Analysis Services Tabular, not all Multidimensional features are currently supported, such as Actions and Named Sets. Furthermore, working with SSAS Multidimensional requires at least SQL Server 2012 SP1 CU4 for the connector to work properly. For more details on working with SSAS Tabular, see "Using Analysis Services Tabular data in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-analysis-services-tabular-data/*. For an overview of capabilities and features of Power BI and SSAS MD connections, see "Connect to SSAS Multidimensional Models in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-ssas-multidimensional/*.

Connecting to Power BI service

Power BI Desktop allows you to connect to datasets published to a Power BI service. To create a connection, select **Get Data> Power BI** service. At this stage, you need to sign in to your Power BI account, unless you have already done so. Signing in opens a window with the workspaces you have access to, and datasets inside them. You can see the Power BI service window in Figure 1-18.

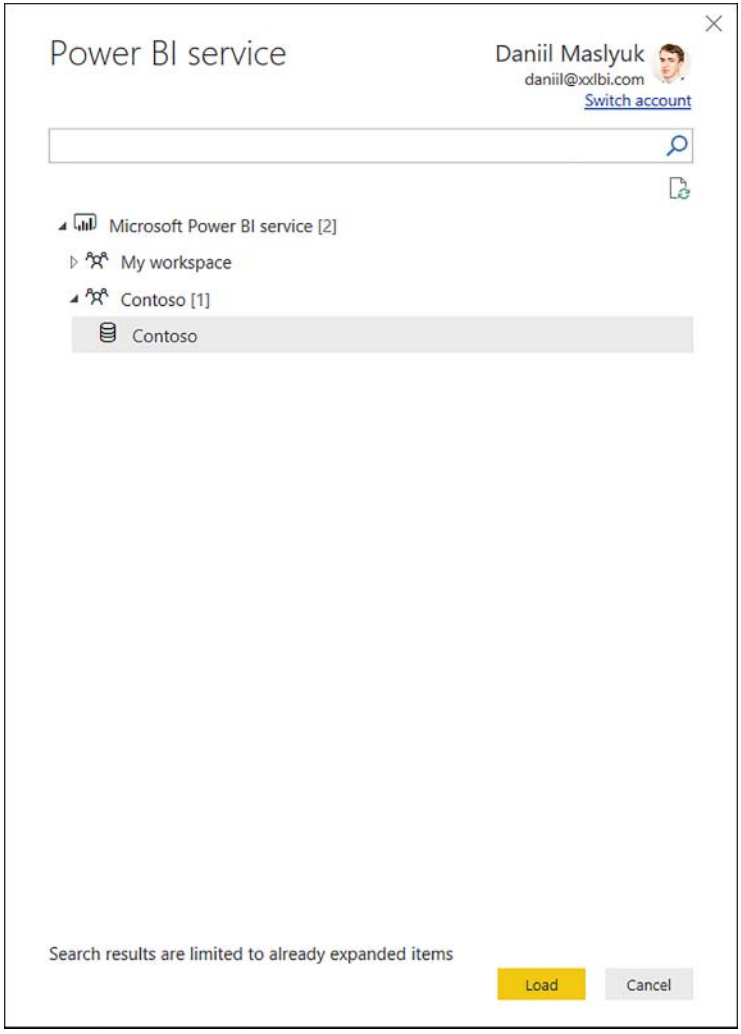


FIGURE 1-18 Power BI service window

Clicking **Load** creates a connection, and it behaves like an SSAS Tabular Live Connection.

> ***MORE INFO*** **CONNECT TO DATA SOURCES IN POWER BI DESKTOP**
>
> For a video overview on how to connect to data sources in Power BI Desktop, see the "Connect to Data Sources in Power BI Desktop" page on Power BI Guided Learning at *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-1-2-connect-to-data-sources-in-power-bi-desktop/*.

MORE INFO **CONNECTING TO SAP BW AND SAP HANA**

The exam does not test your knowledge of Power BI behavior when connecting to SAP Business Warehouse (BW) and SAP HANA, though you should be aware of the significant differences compared to regular relational databases. Both data sources support DirectQuery, but in the case of SAP BW, the experience is closer to Live Connection than DirectQuery because Power Query Editor is not available. With SAP HANA, you can edit your queries in Power Query Editor. For more information, you can review the following articles:

- **"Use the SAP BW Connector in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-sap-bw-connector/*.**
- **"DirectQuery and SAP Business Warehouse (BW)" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-directquery-sap-bw/*.**
- **"Use SAP HANA in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-sap-hana/*.**
- **"DirectQuery and SAP HANA" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-directquery-sap-hana/*.**

Skill 1.2: Perform transformations

Often, once you have created connections, you will need to apply transformations to your data unless you are using a data model that is ready to be used in Power BI Desktop.

Power BI Desktop has a very powerful ETL (extract, transform, load) tool in it: Power Query. Power Query is virtually the same engine that first appeared as an Excel add-in for Excel 2010 and Excel 2013, and it is part of Excel 2016 (Get & Transform Data). This engine is also part of SQL Server Analysis Services 2017, Azure Analysis Services, and Common Data Service.

When you connected to various data sources and worked inside Power Query Editor earlier in this chapter, you have already been using Power Query. Besides connecting to data, Power Query can perform sophisticated transformations to it. In this book, Power Query refers to the engine behind Power Query Editor.

Power Query uses a programming language called M, which is short for "mashup." It is a functional case-sensitive language. The latter point is worth bringing attention to because unlike the other language of Power BI we are going to cover later (DAX), M is case-sensitive. In addition to that, it is a completely new language that, in contrast with DAX, does not resemble Excel formula language in any way.

This section covers how to:

- Design and implement basic and advanced transformations
- Apply business rules
- Change data format to support visualization

Design and implement basic and advanced transformations

Data does not always come in perfect shapes and forms. It is nearly impossible to create a dataset that would be perfect for every analysis because that would create many variations of the same data in one source. Therefore, it is imperative to be able to shape the data in the format that would be the best for your goals.

Power Query overview

We can start by having a closer look at Power Query Editor (Figure 1-19). You can open it from the main Power BI Desktop window by clicking the **Home** tab and selecting **Edit Queries** in the **External Data** group. Let's assume you have connected to Wide World Importers database in Import mode, but you have not loaded any data yet. When connecting, add a check mark to **Fact Sale** in the list of objects and then click **Select Related Tables**; click **Edit** to continue.

> ***NOTE*** **DOWNLOADING WIDE WORLD IMPORTERS DATABASE**
>
> **You can download the WideWorldImportersDW database backup for SQL Server from GitHub at *https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0*. Installation instructions can be found on Microsoft Docs website at *https://docs.microsoft.com/en-us/sql/sample/world-wide-importers/installation-and-configuration-wwi-oltp*.**

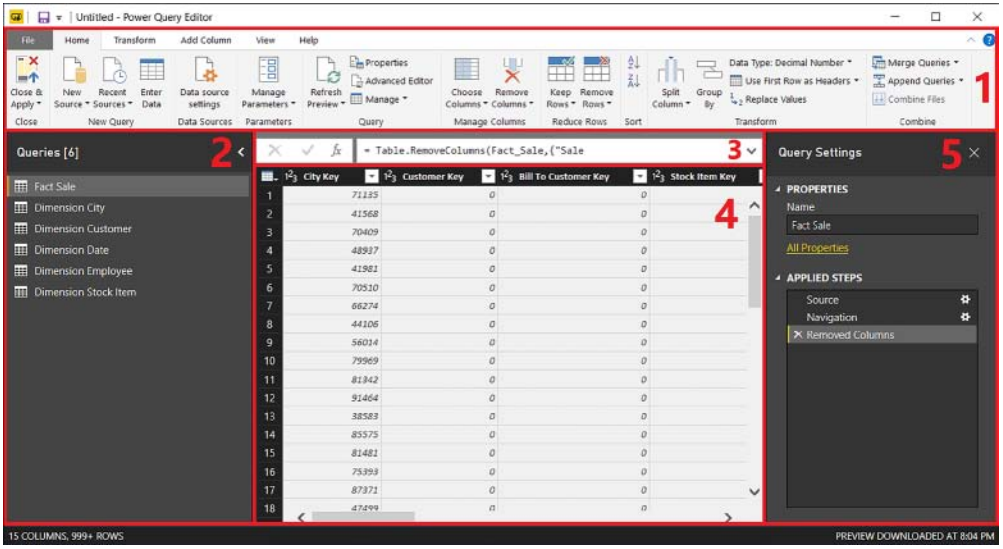


FIGURE 1-19 The Power Query Editor window

Power Query Editor can be divided into five parts, each marked in the figure above:

1. Ribbon
2. Queries pane
3. Formula Bar
4. Data preview
5. Query Settings pane

If you have not yet customized your Power BI settings, you might not see the Formula Bar that is visible in Figure 1-19. Because it is very useful when authoring intermediate-to-complex queries, it is advisable to turn it on. To do that, select the **View** tab, and in the **Layout** group, select **Formula Bar**. **Formula Bar** can be expanded by clicking on the arrow in its right part. If you accidentally close the Query Settings pane, you can turn it back on in the **View** tab as well. Other useful buttons in the **View** tab include **Go to Column** > **Advanced Editor**, and **Query Dependencies**.

Go to Column allows you to select a column from a list of all columns in a table. Once you click the button, a window with a list of all columns opens. Inside, columns are sorted by their natural order (for example, in the order they currently appear in the query). There is an option to sort the list alphabetically, as well as do a search. This can be useful when there are many columns, and you are struggling to locate the column you are trying to find.

In the data preview pane, you can see icons to the left of column names; they signify data types. Figure 1-20 shows a list of data types supported in Power Query, along with their icons.

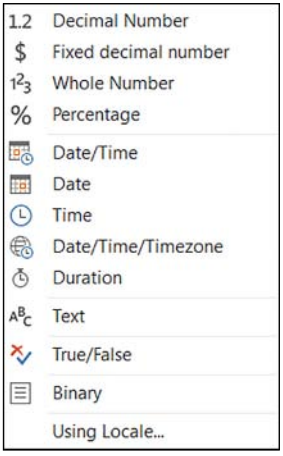


FIGURE 1-20 List of data types supported by Power Query

The last item in the list, **Using Locale**, is not a data type but an option to select a data type considering the locale. For example, 1/4/2018 means 1 April 2018 in Australia, but it means January 4, 2018 in the USA. With Power Query, you can differentiate between the two. If you see ABC123 displayed, it means that there is no data type set for the column.

IMPORTANT **POWER QUERY EDITOR AND DATA MODEL DATA TYPES**

Several data types only exist in Power Query Editor but not once you load the data. For instance, Percentage and Duration values are converted into Decimal Number and Date/Time/Time zone values are converted into Date/Time ones. Currently, Binary columns are not loaded.

As mentioned above, Power Query records all the transformations steps, and you can see them in the Applied Steps area on the right. The last step provides the output for a query. When you click on a step, you can see the code behind it in Formula Bar. In Advanced Editor, you can see all steps at once, and you can edit the code as well. Currently, Advanced Editor only has one feature: it checks for some obvious syntax errors. There is no IntelliSense yet, so if you are coding in Advanced Editor, you are on your own.

NOTE **DATA PREVIEW RECENTNESS**

To make query editing experience more fluid, Power Query caches data previews. Therefore, if your data changes often, you may not see the latest data in Power Query Editor. To refresh a preview, you can select Home > Refresh Preview. To refresh previews of all queries, you should select Home > Refresh Preview > Refresh All.

In the **Query Dependencies** view, you can see all of your data sources and queries tied together when there is a connection between them. In our example, there is one data source: the WideWorldImportersDW database, which has a database icon next to it. From this data source stems six arrows—one to each query, which, in our case, are tables. You can see the **Query Dependencies** view in Figure 1-21.

FIGURE 1-21 Query Dependencies view

In the **Home**, **Transform**, and **Add Column** tabs we see buttons that transform data, and we are going to look at some of them in detail.

> ***MORE INFO*** **THE POWER QUERY EDITOR**
>
> **For a more detailed description of the Power Query Editor interface, including illustrations for each area, see "Query overview in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-query-overview/*.**

Using the Power Query Editor interface

If you followed the example outlined above, you need to take an additional step to make your screen match Figure 1-19. Note that in the Fact Sale query, there is a step called **Removed Columns**. In it, unnecessary columns are excluded. Power Query does not modify any underlying data by default, so if you remove a column in a query, it only removes it from this query and keeps it in the data source.

If you right-click on the left-most column, **Sale Key**, and select **Remove**, you will remove this column from the query. At this stage, a new step—**Removed Columns**—would be added to your query. If you look at Formula bar, it will have the following code:

```
= Table.RemoveColumns(Fact_Sale,{"Sale Key"})
```

This code is written in M. At this stage, we are not going to modify the code, but it is still useful to see that the step consists of a function that takes two arguments: a table and a list of columns to remove. Curly braces denote a list in M.

You can now remove some more columns that you don't need. Hold the Ctrl key and select the following columns:

- Description
- Package
- Total Dry Items
- Total Chiller Items
- Lineage Key
- Dimension.City
- Dimension.Customer(Bill To Customer Key)
- Dimension.Customer(Customer Key)
- Dimension.Date(Delivery Date Key)
- Dimension.Date(Invoice Date Key)
- Dimension.Employee
- Dimension.Stock Item

Right-click any of them and select Remove. Note that there is no extra step generated, and the code in Formula Bar is updated to include more columns:

```
= Table.RemoveColumns(Fact_Sale,{"Sale Key", "Description", "Package", "Total Dry
Items", "Total Chiller Items", "Lineage Key", "Dimension.City", "Dimension.Customer(Bill
To Customer Key)", "Dimension.Customer(Customer Key)", "Dimension.Date(Delivery Date
 Key)", "Dimension.Date(Invoice Date Key)", "Dimension.Employee", "Dimension.Stock
 Item"})
```

Note also how we had to remove the following columns only because we did not uncheck the **Include Relationship Columns** option in advanced settings when we first connected to the database:

- Dimension.City
- Dimension.Customer(Bill To Customer Key)
- Dimension.Customer(Customer Key)
- Dimension.Date(Delivery Date Key)
- Dimension.Date(Invoice Date Key)
- Dimension.Employee
- Dimension.Stock Item

If you click the cog next to the Source step, you can change the setting.

IMPORTANT **UNCHECKING INCLUDE RELATIONSHIP COLUMNS**

Be aware that if you uncheck the Include Relationship Columns option, the Select Related Tables function in the Navigator window will not work correctly. If you exclude the relationship columns, you must pick the following tables manually:

- **Fact Sale**
- **Dimension.City**
- **Dimension.Customer**
- **Dimension.Date**
- **Dimension.Employee**
- **Dimension.Stock Item**

You can safely leave the option enabled because the relationship columns are not loaded into data model; they are only shown in Power Query Editor.

Once you uncheck **Include Relationship Columns** and click **OK**, you can see an error message in place of the **Removed Columns** step. This error message is shown in Figure 1-22.

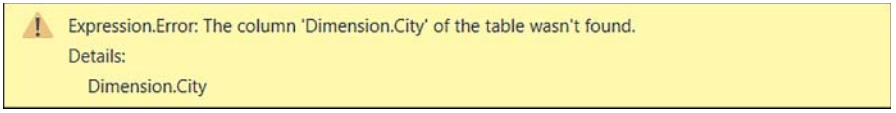


FIGURE 1-22 Error message after we excluded relationship columns

You received this error because you had previously removed several columns, but now that these columns are excluded, the code is trying to remove columns that no longer exist. This error can be fixed in two ways. First, change the settings back and include the relationship columns. Second, remove the last step applied to Fact Sale and remove the unnecessary columns again, but this time without relationship columns.

To remove a step, you can click on the cross icon to the left of its name. In order to remove all steps starting with a certain one, you can right-click on the step and select **Delete Until End**. In our case, it does not matter which option we choose because the **Removed Columns** step is the last one anyway, which means you can select **Delete**.

Now that you've canceled the last step, remove the extra columns again by selecting only the ones that you need. To do that, click **Home** > **Manage Columns** > **Choose Columns**. Note that when you click on the text part of the button, you have an option to select **Go to Column**, which is the same button as in the View tab. In some cases, the same button appears in different ribbons.

Once you click **Choose Columns**, a window with a list of columns appears, and it is identical to the **Go to Column** window, except you can choose multiple columns at once. Uncheck the following columns, keeping the others selected:

- Sale Key
- Description
- Package
- Total Dry Items
- Total Chiller Items
- Lineage Key

Power Query has now created a step called **Removed Other Columns**. If you look at Formula Bar now, you will see the following code:

```
= Table.SelectColumns(Fact_Sale,{"City Key", "Customer Key", "Bill To Customer Key",
 "Stock Item Key", "Invoice Date Key", "Delivery Date Key", "Salesperson Key", "WWI
 Invoice ID", "Quantity", "Unit Price", "Tax Rate", "Total Excluding Tax", "Tax Amount",
 "Profit", "Total Including Tax"})
```

Note that Power Query now uses the Table.SelectColumns function, which, like Table.RemoveColumns, takes two arguments: a table, and a list of columns to keep, instead of columns to remove. Even though the approach is different, the result is the same. You should now have a table with the following columns:

- City Key
- Customer Key
- Bill To Customer Key
- Stock Item Key
- Invoice Date Key
- Delivery Date Key
- Salesperson Key
- WWI Invoice ID
- Quantity
- Unit Price
- Tax Rate
- Total Excluding Tax
- Tax Amount
- Profit
- Total Including Tax

This is a good example that shows that in many cases, there is more than one way to achieve the same goal in Power BI.

Steps in the Applied Steps area of Query Settings can be renamed, which may be useful for code documentation purposes. To do that, you can either select the step you want to rename and hit F2, or you can right-click the step and select **Rename**. In our case, we can keep the names of all steps as-is.

Double-clicking on a step is the same as clicking on the cog wheel next to it; doing so opens step settings, which some, but not all, steps have. Step settings can also be edited by selecting **Edit Settings** after right-clicking on a step.

If you would like to insert a new step to write your own code, you can do it in two ways. First, you can right-click on a step and select **Insert Step After**. This will insert a step after the currently selected step, and the new step will reference the currently selected one. Second, you can click on the **Fx** button in Formula bar, which produces the same result.

Steps can be moved up and down either by dragging them or by right-clicking on a step and selecting either **Move Up** or **Move Down**. Note that in some cases your query might break if you assemble your steps in an incorrect order. For example, if you right-click on the

Removed Other Columns step and select **Move Up**, you will get an error indicating the column City Key was not found, and you will see a fourth step added: **Fact_Sale**. This behavior is explained by the fact that system steps—such as opening a specific database after a connection to a server was made and then locating a specific table—are grouped into a special step called **Navigation**. If you move the **Removed Other Columns** step back down, you will again see only three steps. If you now click **Home**, **Query**, **Advanced Editor**, you will see four steps instead. You can see the full Fact Sale query in Listing 1-1.

LISTING 1-1 Full code of the Fact Sale query

```
let
    Source = Sql.Databases("localhost", [CreateNavigationProperties=false]),
    WideWorldImportersDW = Source{[Name="WideWorldImportersDW"]}[Data],
    Fact_Sale = WideWorldImportersDW{[Schema="Fact",Item="Sale"]}[Data],
    #"Removed Other Columns" = Table.SelectColumns(Fact_Sale,{"City Key", "Customer
Key", "Bill To Customer Key", "Stock Item Key", "Invoice Date Key", "Delivery Date Key",
 "Salesperson Key", "WWI Invoice ID", "Quantity", "Unit Price", "Tax Rate", "Total
 Excluding Tax", "Tax Amount", "Profit", "Total Including Tax"})
in
    #"Removed Other Columns"
```

In the first step, Source, we connect to a server; in the second step, WideWorldImporters-DW, we open the WideWorldImportersDW database; in the third step, Fact_Sale, we open the Fact Sale table. Finally, in the fourth step, #"Removed Other Columns," we remove unnecessary columns. Note that the name of the last step, **Removed Other Columns**, contains spaces, and because of this it must be put into double quotation marks and prefixed with a number sign.

You can now see that when you moved the **Removed Other Columns** step up, you placed it after we opened the WideWorldImportersDW database before we opened the Fact Sale table. This resulted in an error because the columns we were trying to remove could not be located. This example shows that it is important to be careful when you are moving your steps in a query. When you move, add, or delete steps, Power Query only handles the basic dependencies: it updates step references, but it does not make sure that a query will work.

Queries can be split into parts using the **Extract Previous** in the right-click menu. If you right-click on the **Removed Other Columns** step and select **Extract Previous**, you will be prompted to enter the new query name. You can type any name you like. In this example, we are going to name the new query SaleInitial. Once you type the name and click **OK**, a new query with this name is created. This query contains all the steps before the **Removed Other Columns** step. In the Fact Sale query, these steps are replaced with the reference to the Sale-Initial query. Both queries can be seen in Listing 1-2.

LISTING 1-2 SaleInitial and Fact Sale queries

```
// SaleInitial
let
    Source = Sql.Databases("localhost", [CreateNavigationProperties=false]),
    WideWorldImportersDW = Source{[Name="WideWorldImportersDW"]}[Data],
    Fact_Sale = WideWorldImportersDW{[Schema="Fact",Item="Sale"]}[Data]
in
    Fact_Sale
```

```
// Fact Sale
let
    Source = SaleInitial,
    #"Removed Other Columns" = Table.SelectColumns(Source,{"City Key", "Customer Key",
 "Bill To Customer Key", "Stock Item Key", "Invoice Date Key", "Delivery Date Key",
 "Salesperson Key", "WWI Invoice ID", "Quantity", "Unit Price", "Tax Rate", "Total
 Excluding Tax", "Tax Amount", "Profit", "Total Including Tax"})
in
    #"Removed Other Columns"
```

This feature can be useful when you want to separate complex queries into smaller parts for easier maintenance or to reuse a query part.

Some queries support what is called Query Folding. Power Query will try to translate its transformations into the data source's native language where possible. You can see whether Query Folding takes place by right-clicking on a step and selecting **View Native Query**. If the step cannot be selected, it means that Query Folding does not take place. If you click on the **Removed Other Columns** step in the **Fact Sale** step, you will be able to view the native query (Listing 1-3).

LISTING 1-3 Native query of the Removed Other Columns step

```
select [City Key],
    [Customer Key],
    [Bill To Customer Key],
    [Stock Item Key],
    [Invoice Date Key],
    [Delivery Date Key],
    [Salesperson Key],
    [WWI Invoice ID],
    [Quantity],
    [Unit Price],
    [Tax Rate],
    [Total Excluding Tax],
    [Tax Amount],
    [Profit],
    [Total Including Tax]
from [Fact].[Sale] as [$Table]
```

Because we are connected to a SQL Server database, Power Query translated its transformations into SQL. You can notice that instead of importing all the columns and then deleting the unnecessary ones, Power Query is importing only the desired columns.

MORE INFO **QUERY FOLDING**

Query Folding is supported not only by relational databases but by some other data sources as well. For performance reasons, it is best to place the transformations that do not support Query Folding after those that do. For more information about Query Folding, you can read Koen Verbeeck's article, "Query Folding in Power Query to Improve Performance" at *https://www.mssqltips.com/sqlservertip/3635/query-folding-in-power-query-to-improve-performance/*.

The last option in the menu when you right-click on a step is **Properties**. In this window, you can rename the step, as well as add a comment to it. For example, we can include the following comment: "Less is more." This comment will be visible in Advanced Editor. You can see the full query, including the comment, in Listing 1-4.

LISTING 1-4 A comment next to the Removed Other Columns step

```
let
    Source = SaleInitial,
    // Less is more
    #"Removed Other Columns" = Table.SelectColumns(Source,{"City Key", "Customer Key",
"Bill To Customer Key", "Stock Item Key", "Invoice Date Key", "Delivery Date Key",
"Salesperson Key", "WWI Invoice ID", "Quantity", "Unit Price", "Tax Rate", "Total
Excluding Tax", "Tax Amount", "Profit", "Total Including Tax"})
in
    #"Removed Other Columns"
```

It is always a good practice to give your queries friendly names, because they later become tables in your data model. If reports are going to be built by another person, they may be confused by technical terms such as "fact" and "dimension." This also makes the DAX formulas less readable.

To rename a query, you can either right-click on it and select **Rename**, or you can rename it in query properties in the **Query Settings** pane on the right. In our example, queries should be renamed like in Table 1-2.

TABLE 1-2 Old and new query names

Old name	New Name
Fact Sale	Sale
Dimension City	City
Dimension Customer	Customer
Dimension Date	Date
Dimension Employee	Employee
Dimension Stock Item	Stock Item

Let's leave the SaleInitial query named as-is. You can note that it contains the same data as the Sale query, and it has the unnecessary columns. You can either delete the query and replace the code of the Sale query with code from Listing 1-1, or you can disable loading of the

SaleInitial query. Note that if you try to delete SaleInitial now, you will get the error message shown in Figure 1-23.

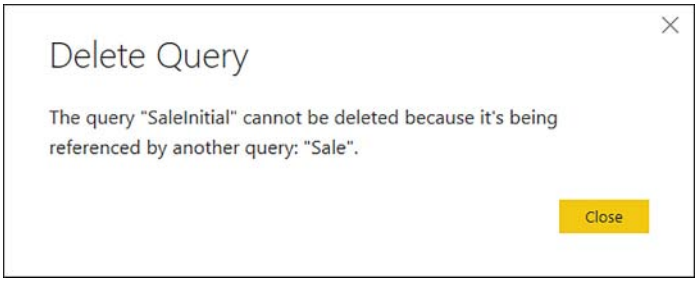


FIGURE 1-23 Error when deleting the SaleInitial query

In this case, we can proceed with replacing the code of the Sale query with code from Listing 1-1 and then delete the SaleInitial query, but disabling loading of SaleInitial is a perfectly valid option, too.

We can disable loading of a query in two ways: first, we can right-click on it and deselect **Enable Load**; second, we can click on the **All Properties** hyperlink in the Query Settings pane. The Query Properties window can also be opened by right-clicking on a query and selecting **Properties**. When you click on the hyperlink, the Query Properties window opens, where you can set the query name and description; you can also enable or disable the load of the query to report and include or exclude it from report refresh. The latter two options are enabled by default. You can uncheck **Enable Load To Report** now. This also automatically excludes the query from report refresh. In the description area, you can enter some text, which will appear in the Query Dependencies view. As an example, enter **Staging Sale** query into the Description field.

At this stage, if we open **Query Dependencies**, we will see that the Sale query comes from the SaleInitial query, and loading of the latter is disabled. You can see the Query Dependencies window in Figure 1-24.

FIGURE 1-24 Query Dependencies view after disabling load of SaleInitial

Disabling the load of a query does not disable the load of queries that reference it, so the Sale query will still be loaded and contain all the data it should. Back in Power Query Editor, in the Queries pane on the left, the SaleInitial's name is now displayed in italics, and the font color is darker compared to other queries.

You can duplicate and reference queries by right-clicking on one of them and selecting **Duplicate** or **Reference**, respectively. Duplicating a query does exactly what the name implies: it creates a copy of the query with the same steps. This way, there is no dependency on the original query, and it can be safely deleted if need be. Referencing, on the other hand, creates a new query with a single step called Source, which references the original query. We have already seen the effects of a query reference with Sale and SaleInitial, where the former referenced the latter. There was a dependency, which prevented SaleInitial from being deleted.

Whether you need to duplicate or reference a query depends on your objectives. In general, it is preferable to reference queries rather than creating copies of them, because that way you follow the "don't repeat yourself" principle.

If you want to duplicate more than one query at once, you can do so by selecting the queries while holding either Ctrl or Shift key and clicking **Copy**, **Paste**. Note that this allows you to paste your queries to other destinations, such as Excel's Power Query or even Notepad for documentation purposes.

Queries can be grouped into folders for easier navigation when you have many queries. Right-click on the **Sale** query and select **Move to Group** > **New Group**. Enter **Facts** in the **Name** field, and click **OK**. This creates two groups, both of which have folder icons next to them: **Fact Tables**, and **Other Queries**. The numbers in square brackets next to the groups display the number of queries in them. Now select the following queries and move them to a new group called **Dimensions**:

- City
- Customer
- Date
- Employee
- Stock Item

This leaves only the SaleInitial query in the Other Queries group. You can move the query to the Facts group by dragging and dropping it. This leaves the Other Queries group empty. Groups can also be reordered as necessary.

Basic transformations

To continue with our example, we need to add sales targets to Power Query Editor. Start by creating a connection to the Target.txt file from this book's companion files. From Power Query Editor, select **New Source** > **Text/CSV** and navigate to the file. Once you click **OK**, and accept default settings, your Power Query Editor window will look similar to Figure 1-25.

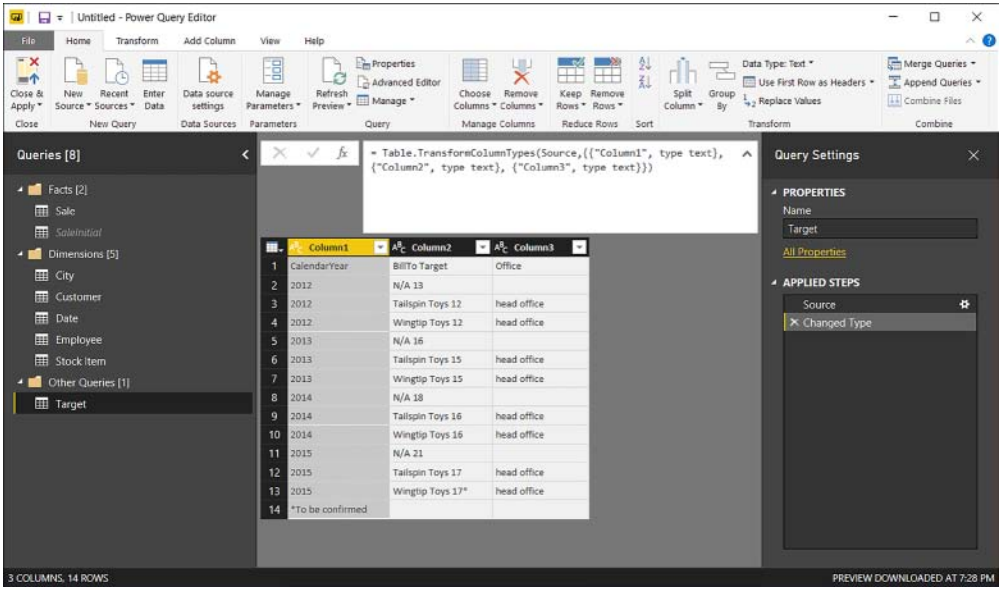


FIGURE 1-25 Power Query Editor after connecting to Target.txt

By default, Power Query tries to detect if there are headers in your text file, and if it decides there are headers, it promotes them from the first row to header names. Also, Power Query automatically detects data types and sets them to what it thinks is appropriate. In our case, Power Query detected no headers and set all columns to type text.

The automatic detection of headers and data types does not always happen correctly, and we should review the steps necessary to apply the transformations manually. We can right-click on the **Changed Type** step and select **Delete**. Only one step is left now: **Source**. We see headers in the first row, and all columns are of type text, given ABC displayed to the left of its names.

To promote the first row to headers, select **Home** > **Transform** > **Use First Row As Headers**. Note that there is an option to demote headers by selecting **Use Headers As First Row**. Alternatively, you can select **Transform** > **Table** > **Use First Row As Headers**. Once we promote the headers, Power Query again detects the data types automatically as text. We can remove this step again by clicking on the cross icon to the left of the **Changed Type** step's name. To set a column data type, click on the data type icon and select the desired data type. Select **Whole Number For The Year** column. Data types can also be selected by clicking **Home** > **Transform**, **Data Type, As Well As Transform** > **Any Column** > **Data Type**.

You will notice that in row 13, column CalendarYear, we have an error. You can click on the cell without clicking on the **Error** hyperlink to see the error message: "DataFormat.Error: We couldn't convert to Number. Details: *To be confirmed." This means that Power Query tried to convert a text string, "*To be confirmed," to number and failed. As it often happens in Power Query, there is more than one way to fix the error.

In case you want to filter the data you are importing, you have two options: either by keeping the specific rows or removing rows. Both options can be found by clicking **Home**, **Reduce Rows**.

Under **Keep Rows**, you have the following options:

- **Keep Top Rows**, where you specify the number of top rows to keep.
- **Keep Bottom Rows**, for which you pick the number of bottom rows to keep.
- **Keep Range of Rows**, which skips a specified number of top rows and then keeps the chosen number of rows.

In addition to the first three options, which work on whole tables, you have **Keep Duplicates** and **Keep Errors**, both of which can work on either the whole table or the selected columns only. For example, if you select the whole table and choose **Keep Duplicates**, you will only see the rows that are complete duplicates of each other. However, if you choose only one column and click **Keep Duplicates**, you will get the rows where the values in the selected column are duplicates, regardless of other columns' values.

Under **Remove Rows**, you have six options:

- **Remove Top Rows** Removes a specified number of top rows. Works on the whole table only.

- **Remove Bottom Rows** Removes a specified number of bottom rows. Works on the whole table only.
- **Remove Alternate Rows** Removes rows following a user-supplied pattern: it starts with a specified row, then alternates between removing the selected number of rows and keeping the chosen number of rows. Works on the whole table only.
- **Remove Duplicates** Removes rows that are duplicates of other rows. Works on either the whole table or the selected columns only.
- **Remove Blank Rows** Removes rows that completely consist of either empty strings or nulls; if you need to remove blank values from one column, you can click on the arrow to the right of a column's name and click Remove Empty. Works on the whole table only.
- **Remove Errors** Removes rows that contain errors. Works on either the whole table or the selected columns only.

In case of **Remove Duplicates** and **Remove Errors**, there is a difference between applying these options to all selected columns or the whole table. In the first case, if you have new columns added to your query, the functions will not work on the new columns, because selecting all columns keeps their names in the code. To remove duplicates or errors from the whole table, select the table icon above row numbers and choose either **Remove Duplicates** or **Remove Errors**.

> ***MORE INFO*** **WORKING WITH ERRORS IN POWER QUERY**
>
> **The topic of error handling is reviewed in more detail later in the chapter, in Skill 1.3: "Cleanse data."**

In this case, we can remove the bottom row. Furthermore, we do not need the 2012 targets, as there is no sales data for the year. Therefore, we can remove the top three rows as well, which should leave us with nine rows. To achieve this result, select **Home** > **Reduce Rows** > **Keep Rows** > **Keep Range of Rows**. Type **4** for the First Row and **9** for the **Number of Rows**, and then click **OK**. If you open **Advanced Editor**, you should see a script like the one shown in Listing 1-5.

LISTING 1-5 M query after removing unnecessary rows

```
let
    Source = Csv.Document(File.Contents("C:\Companion\Target.txt"),[Delimiter=";",
 Columns=3, Encoding=1252, QuoteStyle=QuoteStyle.None]),
    #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),
    #"Changed Type" = Table.TransformColumnTypes(#"Promoted Headers",{{"CalendarYear",
 Int64.Type}}),
    #"Kept Range of Rows" = Table.Range(#"Changed Type",3,9)
in
    #"Kept Range of Rows"
```

Note that the last step's formula is Table.Range(#"Changed Type",3,9). Even though we specified **4** and **9** as parameters in **Keep Range of Rows** settings, we see 3 and 9 in the formula. This is because Power Query has 0-based index system, meaning that the first row is row number 0, the second row is row number 1, and so on.

The next thing we need to do is turn this dataset into the appropriate format. We are looking to get the following three columns: Calendar Year, Bill To Customer, and Target. The last column should be in dollars, not millions of dollars.

To get the first column, Calendar Year, rename CalendarYear and insert a space between the two words. There are four ways to rename a column:

- Double-click its name and enter a new name.
- Right-click its name and select **Rename**.
- Select a column and press F2.
- Select **Transform**, **Any Column**, **Rename**.

The second and third columns require some more work. First, we need to separate the target values from Bill To Customer. Second, where appropriate, we need to append the office name in brackets to Bill To Customer. Finally, we should set the correct data types and names.

We can start by splitting the Bill To Target column. To split a column, right-click on its name and select **Split Column**. The same button can be found in **Home** > **Transform**. You will see two options: either split by the delimiter or by the number of characters. In our case, we should select By Delimiter because our Target values are separated from Bill To Customer values by a space, we should select **Space** in the delimiter drop-down list. Below the delimiter selection, we have three **Split At** options: Left-most delimiter, Right-most delimiter, and Each occurrence of the delimiter. The first two options split a column in two, while the number of columns the third option splits in depends on the number of delimiters in column values. This number of columns can be specified manually in Advanced Options below. In Advanced Options, you also can specify the quote character, as well as whether you want to split your values into columns or rows. Also, if you chose to split by a custom delimiter, you can split using special characters, such as a carriage return or line feed. In our case, we should change the **Split at** option from **Each Occurrence** of the delimiter to **Right-most delimiter** and leave the other settings at their defaults.

Once you click **OK**, the column will be split into two: Bill To Target.1 and Bill To Target.2. Note that Power Query has once again detected data types automatically. If this feature is undesirable, it can be turned off by clicking **File** > **Options**, **And Settings** > **Options** > **Current File** > **Data Load** > **Type Detection**. If you need Power Query to detect a column's data type, you can select **Transform** > **Any Column** > **Detect Data Type**.

Before merging Bill To Target.1 and Office, we need to apply some transformations to the Office column. The column's values should be in brackets in case they are not blank, and each word should be capitalized.

IMPORTANT **BLANK AND NULL VALUES IN POWER QUERY**

In Power Query, blanks and nulls are different. Blank values are zero-length text strings, while nulls are empty values. The implication of this is that you can combine a text string with a blank value, but a text string combined with a null value results in a null value.

To replace a blank value by null value, right-click on the Office column and select **Replace Values**. The same button can be found by clicking **Home** > **Transform**, under **Use First Row As Headers**; as well as in **Transform** > **Any Column** grouping. We should leave the first field, **Value to Find**, blank. In the second field, **Replace With**, we should type **null**. In this case, we should leave the Advanced Options as-is, but if we needed, we could opt to match entire cell contents, as well as replace using special characters. Your Replace Values window should look like Figure 1-26.

FIGURE 1-26 Replace Values window

When you click **OK**, in the Office column instead of blank values you should see null written in italic and aligned to the right. To capitalize each word in the Office column values, right-click on the column name and select **Transform** > **Capitalize Each Word**. There are a few other options in Transform:

- Lowercase transforms all symbols into lowercase
- Uppercase transforms all symbols into uppercase
- Trim removes extra spaces, including at beginning and end of text strings
- Clean removes non-printable characters
- Length replaces a text string with the number of characters in it
- JSON parses JSON contents in a string
- XML parses XML contents in a string

One of the ways to append a text string to a column value is by clicking **Transform** > **Text Column** > **Format** > **Add Prefix** or **Add Suffix**. We should add (as a prefix and) as a suffix. Note that if we didn't replace blank values with nulls a few steps back; we would see **()** instead of nulls.

We can now merge Bill To Target.1 and Office in one column. Start by clicking on the **Bill To Target.1** column header, then hold the **Ctrl** key and click on the **Office** column header. Then right-click either of the two selected columns and select **Merge Columns**. Alternatively, you can select **Transform** > **Text Column** > **Merge Columns**. In the Merge Columns settings window, we should select **Space** as a separator, and we should call the new column **Bill To Customer**. Note that if you selected the **Office** column first, then **Bill To Target.1**, the merge would be done in this order instead, so the order in which you click on column headers matters..

Next, we should rename the column **Bill To Target.2 to Target**. Because the figures are in millions of dollars and we want them to be in dollars, we should multiply the values by 1,000,000. Before we can do that, we need to make sure that all column values are numbers. Note that the last value contains an asterisk. If we multiply it by one million, we will get an error. To remove the asterisk right-click on the **Target** column and select **Replace Values**. **Specify * as Value to Find**, and leave the **Replace With** value empty. We should then change the column's data type to a whole number. Once we've done that, we can select the Target column, then click **Transform** tab, > **Number Column** > **Standard** > **Multiply** and enter **1000000**.

Columns can be reordered by dragging and dropping. Alternatively, we can select the columns we want to move, then do one of the following:

- Select **Transform** > **Any Column** > **Move**.
- Right-click on the header of one of the columns and select **Move**, then choose where to move.

Either method gives you these options: **Left**, **Right**, **To Beginning**, and **To End**. If you are moving more than one column using this method, the order in which you select the columns matters. In our case, we just want the Target column to be moved to the end.

Finally, we can sort rows in tables. One way to do it is to select the drop-down arrow next to a column name, then select either **Sort Ascending** or **Sort Descending**. Alternatively, we can select a column, then click **Home** > **Sort** > **Sort Ascending** or **Sort Descending**. Let's sort the **Bill To Customer** in descending order. We can then sort the **Calendar Year** column in ascending order. Note that there is a small 1 next to the drop-down arrow button in the Bill To Customer column's header, and there's a small 2 in the Calendar Year header. These numbers mean that the table is first sorted by **Bill To Customer**, then by **Calendar Year**.

After all the transformations, the full code of the Target query should be as shown in Listing 1-6.

LISTING 1-6 The complete Target query script

```
let
    Source = Csv.Document(File.Contents("C:\Companion\Target.txt"),[Delimiter=";",
 Columns=3, Encoding=1252, QuoteStyle=QuoteStyle.None]),
    #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),
    #"Changed Type" = Table.TransformColumnTypes(#"Promoted Headers",{{"CalendarYear",
 Int64.Type}}),
    #"Kept Range of Rows" = Table.Range(#"Changed Type",3,9),
    #"Renamed Columns" = Table.RenameColumns(#"Kept Range of Rows",{{"CalendarYear",
 "Calendar Year"}}),
    #"Split Column by Delimiter" = Table.SplitColumn(#"Renamed Columns", "Bill To
 Target", Splitter.SplitTextByEachDelimiter({" "}, QuoteStyle.Csv, true), {"Bill To
 Target.1", "Bill To Target.2"}),
    #"Changed Type1" = Table.TransformColumnTypes(#"Split Column by Delimiter",{{"Bill
To
 Target.1", type text}, {"Bill To Target.2", type text}, {"Office", type text}}),
    #"Replaced Value" = Table.ReplaceValue(#"Changed
 Type1","",null,Replacer.ReplaceValue,{"Office"}),
    #"Capitalized Each Word" = Table.TransformColumns(#"Replaced Value",{{"Office",
 Text.Proper, type text}}),
    #"Added Prefix" = Table.TransformColumns(#"Capitalized Each Word", {{"Office", each
 "(" & _, type text}}),
    #"Added Suffix" = Table.TransformColumns(#"Added Prefix", {{"Office", each _ & ")",
 type text}}),
    #"Merged Columns" = Table.CombineColumns(#"Added Suffix",{"Bill To Target.1",
 "Office"},Combiner.CombineTextByDelimiter(" ", QuoteStyle.None),"Bill To Customer"),
    #"Renamed Columns1" = Table.RenameColumns(#"Merged Columns",{{"Bill To Target.2",
 "Target"}}),
    #"Replaced Value1" = Table.ReplaceValue(#"Renamed
 Columns1","*","",Replacer.ReplaceText,{"Target"}),
    #"Changed Type2" = Table.TransformColumnTypes(#"Replaced Value1",{{"Target",
 Int64.Type}}),
    #"Multiplied Column" = Table.TransformColumns(#"Changed Type2", {{"Target", each _ *
 1000000, type number}}),
    #"Reordered Columns" = Table.ReorderColumns(#"Multiplied Column",{"Calendar Year",
 "Bill To Customer", "Target"}),
    #"Sorted Rows" = Table.Sort(#"Reordered Columns",{{"Bill To Customer",
 Order.Descending}, {"Calendar Year", Order.Ascending}})
in
    #"Sorted Rows"
```

At this stage, your Power Query Editor should look like Figure 1-27.

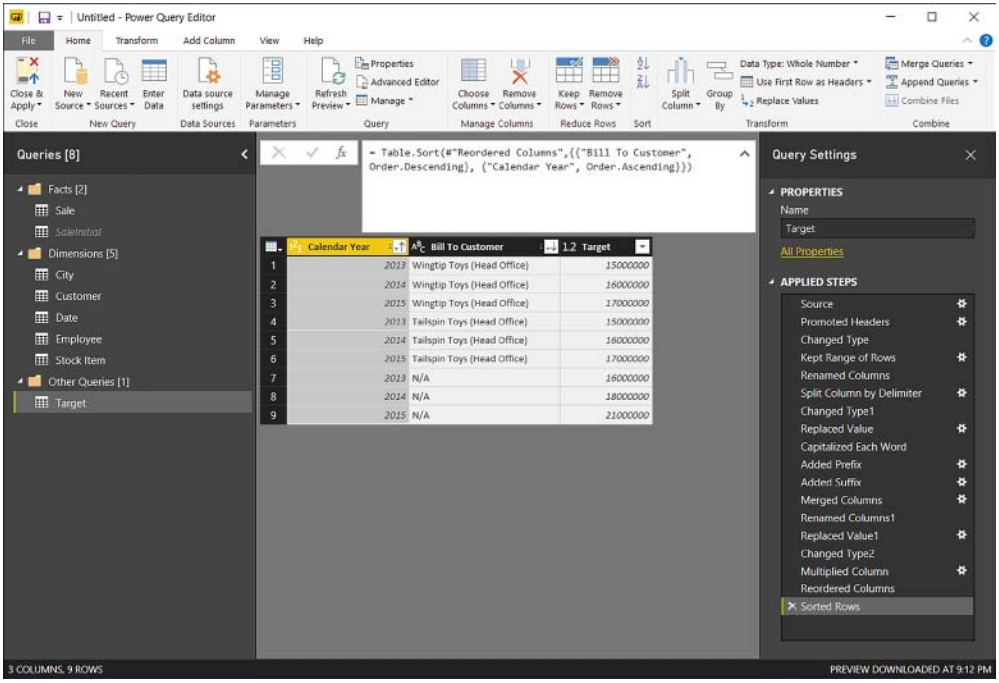


FIGURE 1-27 Power Query Editor after applying all transformations to Target.txt

If we wanted to reverse the order of rows, we could select **Transform** > **Table** > **Reverse Rows**.

> ***MORE INFO*** **CLEAN AND TRANSFORM YOUR DATA WITH THE POWER QUERY EDITOR**
>
> For more examples of transforming data in Power Query, see the "Clean and Transform Your Data with the Power Query Editor" page on Power BI Guided Learning at *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-1-3-clean-and-transform-data-with-query-editor/*.

Advanced transformations

So far, we have reviewed the basic transformations, and now we can review the advanced transformations. We can continue with our example by adding 2016 targets.

First, we need to connect to Target20152016.xlsx from this book's companion files. There is just one sheet, and we want to import it. Once you click **OK**, you see pivoted data that is not suitable for analysis; you need to transform it before we can use it. Furthermore, we can note that there are two levels of headers: year and month. Finally, there are some subtotals, and we should get rid of them. Before continuing, we should rename our query to **Target20152016** and disable loading of it.

To address the two-row header problem, we can transpose our table. To do that, select **Transform** > **Table** > **Transpose**. This switches columns to rows and rows to columns. After transposing the table, you should see a table similar to Figure 1-28.

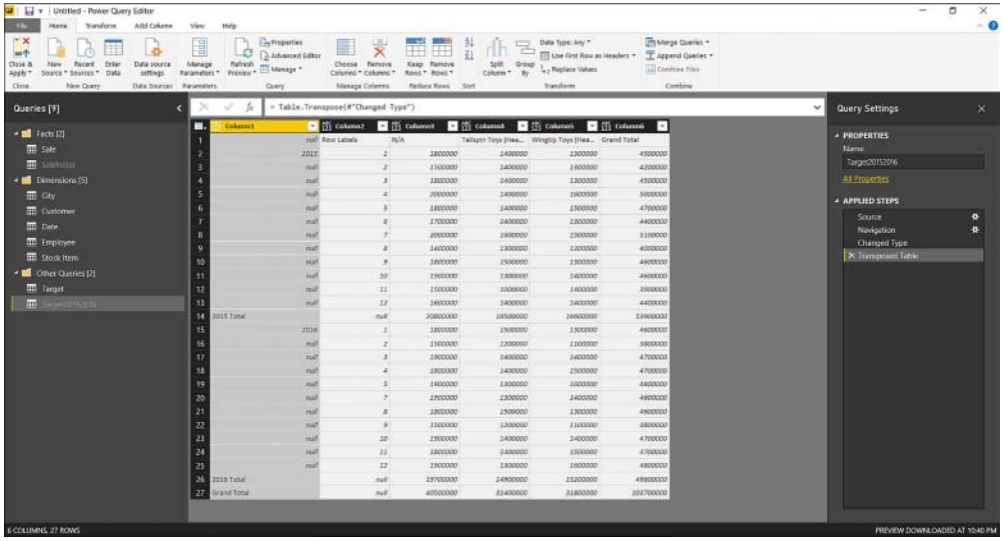


FIGURE 1-28 Power Query Editor after transposing table

Note that each year is written only once, and between year values we see nulls. Fix this by filling the values below years with years above them. To do that, right-click on Column1 and select **Fill** > **Down**. You should also filter out subtotals. Do so by clicking on the **AutoFilter** drop-down arrow next to Column1 and de-selecting **2015 Total**, **2016 Total**, and **Grand Total**. Note that we want to keep the null value in, as it belongs to the first row where the Bill to Customer values are.

At this stage, we need to click **Home** > **Transform** > **Use First Row As Headers**. Now you can see that the last column is Grand Total, so you can safely remove it.

To turn this pivoted table into a suitable format, unpivot some of its columns. Power Query has a very useful function called Unpivot. There are two ways it can be used: either unpivot specific columns, or select the columns to keep as-is and unpivot all other columns. The latter option is preferable when there is a possibility of more columns being added later. In our case, you can select the first two columns, then right-click the header of either of them and select **Unpivot Other Columns**. Note that there are also other options: Unpivot Columns and Unpivot Only Selected Columns. The function used by Unpivot Columns is the same one as Unpivot Other Columns uses: Table.UnpivotOtherColumns. The only difference is in the columns you select—either the ones you want to unpivot, in case of Unpivot Columns, or the columns you want to keep, as in the case of Unpivot Other Columns. Unpivot Only Selected Columns uses a different function: Table.Unpivot.

After you have unpivoted the columns, you can see that the Bill to Customer values are in the column called Attribute, and the values are in the column called Value. You should now rename the columns as shown in Table 1-3.

TABLE 1-3 Old and new column names

Old name	New name
Column1	Year
Row Labels	Month
Attribute	Bill To Customer
Value	Target

Let's summarize our table by Year and Bill To Customer, as we do not need the monthly targets. To do that, you can select **Year** first, then click **Home** > **Transform** > **Group By Or Transform** > **Table** > **Group By**. The Group By window then opens; you'll see a radio button to switch between **Basic** and **Advanced** settings. Specify one or more columns to group by and how to aggregate data. To group by more than one column, switch to **Advanced** settings, or you could have pre-selected multiple columns before clicking **Group By**. Once you have done it, click **Add Grouping** and select **Bill To Customer**. In **New Column Name**, type **Target** instead of **Count**, and select **Sum** as the operation and **Target** as the column. What this means is that we are doing a summation of the Target column and calling the new column Target as well.

Note that there are other aggregation options in the Operation drop-down list: Average, Median, Min, Max, Count Rows, Count Distinct Rows, and All Rows. The last option, All Rows, groups all relevant rows into a table, so you will have nested tables as values in your new column instead of scalar values produced by other operations.

Click **OK** to see a new table that has only six rows and three columns. At this point, if you open Advanced Editor, you should see code similar to Listing 1-7.

LISTING 1-7 M query of Target20152016

```
let
    Source = Excel.Workbook(File.Contents("C:\Companion\Target20152016.xlsx"), null,
 true),
    Sheet1_Sheet = Source{[Item="Sheet1",Kind="Sheet"]}[Data],
    #"Changed Type" = Table.TransformColumnTypes(Sheet1_Sheet,{{"Column1", type text},
 {"Column2", Int64.Type}, {"Column3", Int64.Type}, {"Column4", Int64.Type}, {"Column5",
 Int64.Type}, {"Column6", Int64.Type}, {"Column7", Int64.Type}, {"Column8", Int64.Type},
 {"Column9", Int64.Type}, {"Column10", Int64.Type}, {"Column11", Int64.Type},
 {"Column12", Int64.Type}, {"Column13", Int64.Type}, {"Column14", type any},
 {"Column15", Int64.Type}, {"Column16", Int64.Type}, {"Column17", Int64.Type},
 {"Column18", Int64.Type}, {"Column19", Int64.Type}, {"Column20", type any},
 {"Column21", type any}}),
    #"Transposed Table" = Table.Transpose(#"Changed Type"),
    #"Filled Down" = Table.FillDown(#"Transposed Table",{"Column1"}),
    #"Filtered Rows" = Table.SelectRows(#"Filled Down", each ([Column1] = null or
 [Column1] = 2015 or [Column1] = 2016)),
    #"Promoted Headers" = Table.PromoteHeaders(#"Filtered Rows",
 [PromoteAllScalars=true]),
    #"Changed Type1" = Table.TransformColumnTypes(#"Promoted Headers",{{"Column1",
 Int64.Type}, {"Row Labels", Int64.Type}, {"N/A", Int64.Type}, {"Tailspin Toys (Head
 Office)", Int64.Type}, {"Wingtip Toys (Head Office)", Int64.Type}, {"Grand Total",
 Int64.Type}}),
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type1",{"Grand Total"}),
    #"Unpivoted Other Columns" = Table.UnpivotOtherColumns(#"Removed Columns",
 {"Column1", "Row Labels"}, "Attribute", "Value"),
    #"Renamed Columns" = Table.RenameColumns(#"Unpivoted Other Columns",{{"Column1",
 "Calendar Year"}, {"Row Labels", "Month"}, {"Attribute", "Bill To Customer"}, {"Value",
 "Target"}}),
    #"Grouped Rows" = Table.Group(#"Renamed Columns", {"Calendar Year", "Bill To
 Customer"}, {{"Target", each List.Sum([Target]), type number}})
in
    #"Grouped Rows"
```

Notice how we have 2015 targets in both Target and Target20152016 queries. Assume we are told that the figures in the Target20152016 query are more accurate, and we should use them instead of figures in the Target query. To filter out 2015 values from the Target query, select the query, then click on the drop-down button next to the **Calendar Year** and select **Number Filters** > **Less Than**. In the opened Filter Rows window, you will see the basic settings, which allow you to enter two conditions joined in either "And" or "Or" logic. Switching to **Advanced Settings** will let you specify as many conditions as required, and you can refer to other columns with the user interface. In our case, we can keep the settings basic and only select **2015** from the drop-down list. As an alternative, you can also have entered **2015** manually. Once you've finished that, click **OK**, and this will filter Targets to 2013 and 2014 only. To prevent confusion, rename the Target query to **Target20132014**. You should also disable its load by right-clicking on the query and de-selecting Enable Load.

NOTE **USING VALUE FILTERS IN POWER QUERY**

Power Query gives you options to filter numbers, text, and datetime values based on specified criteria. For numbers, you can select from the following options:

- **Equals**
- **Does Not Equal**
- **Greater Than**
- **Greater Or Equal To**
- **Less Than**
- **Less Than Or Equal To**
- **Between**

For text values, you can choose from the following:

- **Equals**
- **Does Not Equal**
- **Begins With**
- **Does Not Begin With**
- **Ends With**
- **Does Not End With**
- **Contains**
- **Does Not Contain**

When applying text filters, it is important to remember that Power Query is case-sensitive. This can be overridden by using Comparer.OrdinalIgnoreCase. Imke Feldmann wrote about it on her blog at *http://www.thebiccountant.com/2016/10/27/tame-case-sensitivity-power-query-powerbi/*.

Datetime values can be filtered in more than 50 different ways, including absolute and relative filters. The relative filters, such as Last Quarter or Next Month, use the local date and time as at query execution time.

All filter options above can be combined with "And" or "Or" logic using advanced settings.

Appending queries

Let's combine the Target 20132014 and Target20152016 queries into one Target query. To do that, select either of them and click **Home** > **Combine** > **Append Queries** > **Append Queries as New**. This option will create a new query that will consist of appended queries; selecting **Append Queries** appends another query to the currently selected one, keeping it in the same query.

Selecting **Append Queries As New** opens the Append window, where we can select a query to append to the currently selected query. In this case, use the basic settings, because we are appending two tables. If we had three or more tables to append, we could select **Three Or More Tables**, and that would allow us to select more than two tables. If you chose **Target20132014** as the primary table, select **Target20152016** as the table to append to the primary table and click **OK**. This creates a new query called Append1, which consists of only one step: Source. The step uses the Table.Combine function to append queries. We should rename the query to Target.

Let's open the Query Dependencies view, noticing two groups of queries: one group of queries, on the left, originates from a SQL Server database; the other group of queries, on the right, are independent and originate from files. You can see the Query Dependencies view at this stage in Figure 1-29.

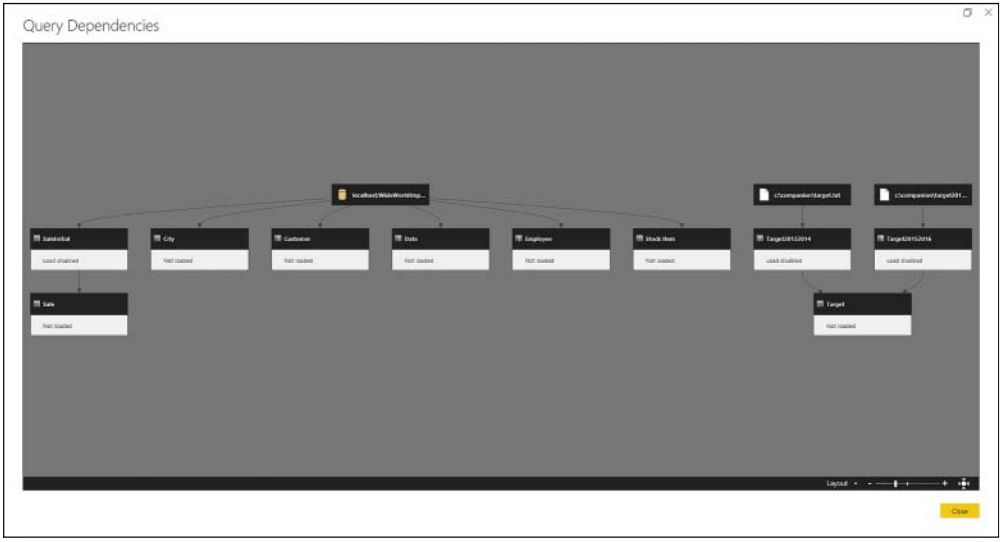


FIGURE 1-29 Query Dependencies view after appending queries

Merging queries

If appending queries is combining queries vertically, then merging queries is combining them horizontally. Our Target query can be enriched with target quantities for each year and customer combination. Because we are going to add other targets, we should rename our Target column in the Target query. A good name would be **Target Amount Excluding Tax**.

Once you have renamed the column, connect to a new text file, TargetQuantity.txt, from this book's companion files folder. In the connection settings window, we can accept the default settings and click **OK**. The query is named TargetQuantity by default, and we can keep the name. We should disable its loading by right-clicking on the query and de-selecting Enable Load.

To merge two queries, first select the primary query, then click **Home** > **Combine** > **Merge Queries**. In our case, we should select **Merge Queries**, not **Merge Queries as New**, because we do not want to create another query.

In the Merge dialog box, shown in Figure 1-30, the first table is pre-selected, and in this case, it is the Target query. You can see a query preview below its name. Under the preview area, select the second query from the drop-down list. After making a selection—**TargetQuantity**—you will see its preview. Below the preview, select **Join Kind**. These are the options available:

- **Left Outer** Keeps all rows from the first table and matching rows from the second table
- **Right Outer** Keeps all rows from the second table and matching rows from the first table
- **Full Outer** Keeps all rows from both tables
- **Inner** Keeps matching rows only
- **Left Anti** Keeps rows that are present in the first table but not in second
- **Right Anti** Keeps rows that are present in the second table but not in first

The last two options can be particularly useful when you are looking for items that are present in one table but not the other. In our case, we can keep **Left Outer** selected. Then you need to select the matching columns. In other words, we need to tell Power Query how to match the tables. Hold the **Ctrl** key and select **Calendar Year**, **Bill To Customer in Target**. Note that Calendar Year has a small numeral 1 included in its header, and Bill To Customer has numeral 2 in its header.

After following the same steps for TargetQuantity, we will see a message reading "The selection has matched 10 out of the first 12 rows." Even though it is not a perfect match, we can leave it as is for now, and we will then check which rows did not have a match. Click **OK**.

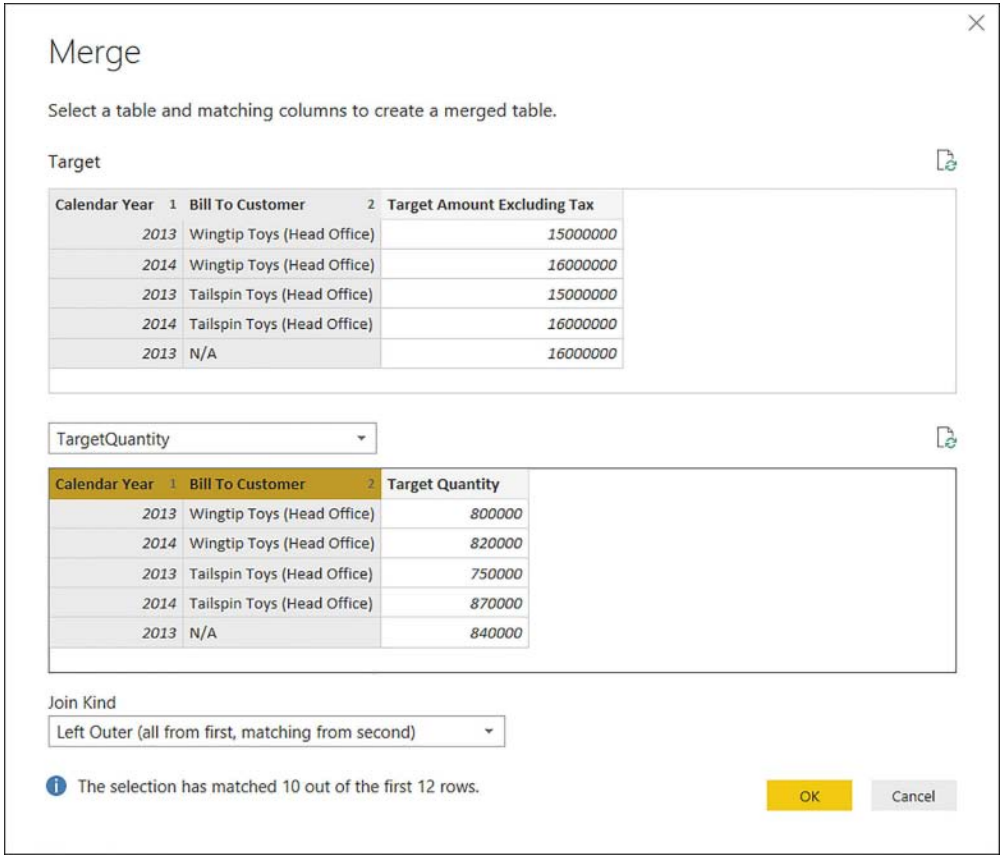


FIGURE 1-30 Merge dialog box

After clicking **OK**, a new column called TargetQuantity is added to the Target table. Note a new step: Merged Queries. This column consists of tables that contain matching rows from the TargetQuantity table. This column can be expanded by clicking the double arrow button. Clicking it opens a window where you can select which columns you would like to add and whether you want to aggregate them. The options are shown in Figure 1-31.

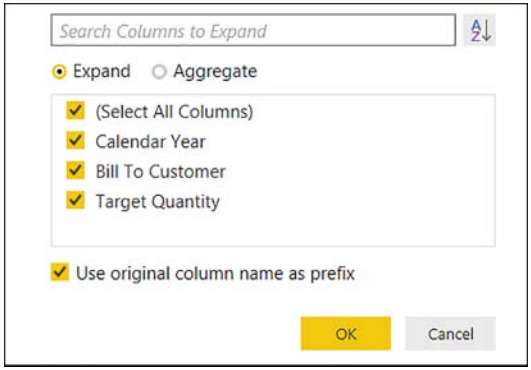


FIGURE 1-31 Table expansion options

Column names can be displayed either in the natural or alphabetical order. If you select **Aggregate**, you will be able to select columns along with their aggregation methods. For example, you can select **Bill To Customer**. In this case, choose **Expand** instead of **Aggregate**. Also, because we already have Calendar Year and Bill To Customer in our query, we can leave only **Target Quantity** selected.

If we leave the **Use Original Column Name As Prefix** option checked, the expanded column will be called TargetQuantity. In our case, this option should be unchecked. We can now click **OK**. We can see a new column in place of the TargetQuantity column: **Target Quantity**. This column contains matching Target Quantity values from TargetQuantity query. In the Applied Steps pane, we can notice a new step: **Expanded TargetQuantity**.

Note that there are two null values in the new column. This is in line with the message shown in Figure 1-29, and it happened because the values did not match perfectly. If we investigate why, we will see that some values in the **Bill To Customer** column in the Target2032014 query have spaces in the end, and they need to be trimmed for the merge to be correct. If we trim the spaces now, it will be of no use, because the merge has already taken place. Instead, we should go to the Renamed Columns step in the Target query, then right-click on the **Bill To Customer** column and select **Transform** > **Trim**. In the subsequent dialog box, you will be asked if you are sure to insert a step, and you can click **Insert** to confirm. A new step, **Trimmed Text**, will be inserted between **Renamed Columns** and **Merged Queries**. Once you go to the final step, **Expanded TargetQuantity**, you will note that nulls will have disappeared, because the merge has now been done with the correct values.

Because we have finished adding targets to our data, we can group all four target-related queries into a group called **Targets**, leaving no queries in the **Other Queries** group.

MORE INFO **MORE ADVANCED DATA SOURCES AND TRANSFORMATION**

For more examples of advanced data transformation, see "Shape and combine data in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-shape-and-combine-data/*.

For video examples, see "More Advanced Data Sources and Transformation" on Power BI Guided Learning at *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-1-4-advanced-data-sources-and-transformation/*.

Creating new columns in tables

You can enrich your data model by adding columns to your tables. From the **Add Column** tab, in the **General** group, you can select the following options:

- **Column from Examples**
- **Custom Column**
- **Invoke Custom Function**
- **Conditional Column**
- **Index Column**
- **Duplicate Column**

In the examples that follow, we will review these options, and we will be removing the new columns after creating them because we do not need them in our data model.

Column from Examples

Column from Examples allows you to create a new column by typing one or more values. If the values you type come from one or more existing columns, Power Query will be looking for ways to extract the new values from the existing ones. When selecting **Column from Examples**, there are two options: **From All Columns** and **From Selection**. The difference between the two options is the number of columns Power Query will be scanning when trying to arrive at the same values. When you select either option, the data preview pane is transformed: the Queries and Query Settings panes are shaded. Also, there is a dialog area above column headers and a new column area on the right where you can type values. By default, the new column is called Column1, unless you already have a column with this name. You can change the name by double-clicking on **Column1** or selecting the new column and pressing **F2**. Every existing column has a check box in its header, which serves the same purpose as the **From All Columns** and **From Selection** options before: the check boxes determine which columns Power Query will be working with when creating a column from examples. If you type a few values, but Power Query cannot find a way to reproduce your results with a formula, it will display the following message in the dialog area above column headers: "Please enter more sample values." Once Power Query finds a suitable transformation, its formula will be displayed instead of the message. If you have not changed the default column name, Power Query might give it another name it deems appropriate.

We can review the Column from Examples functionality using the following example. Select the **Calendar Year** column and click **Add Column** > **Column from Examples** > **From Selection**. The interface transforms, and we can enter sample values into the new column. Enter numbers that are greater than the year by one. For example, if you see "2013" in the first row, enter **2014**. After we enter one value, Power Query will display a message asking us to enter more sample values. If we enter one more, Power Query will display the following text instead of the message: "Transform: [Calendar Year] + 1". What follows "Transform:" is a formula written in M. Note that the column name is now "Addition" instead of "Column1." Clicking **OK** creates a new column and a new query step called **Inserted Addition**. Because this column is not needed in the data model, you can delete this step.

Custom Column

When you click **Add Column** > **General** > **Custom Column**, the Custom Column window opens where you can enter the new column name and its formula. There is already an equals sign in the custom column formula field, which cannot be removed. In the right part of the window, there is a list of available columns in the currently selected table. You can add them by either double-clicking on them or by selecting one of them and clicking the **Insert** button. To reproduce the same column that we created with Column from Examples, double-click **Calendar Year** in the columns list and type **+ 1** after. The whole formula will read as follows:

```
=[Calendar Year] + 1
```

Clicking **OK** will create the new column, as well as add a step called **Added Custom**. Note that the column's data type is not defined, and you need to apply it manually. If you followed along with this example, you could remove this column because it is not required in our data model.

Invoke Custom Function

Invoke Custom Function applies a custom function to each row of a table. For the purposes of this example, create a function that adds one to Calendar Year. Click **Home** > **New Source** > **Other** > **Blank Query**. Rename the query to **fAddOne**. Open Advanced Editor, delete everything, and paste the following code shown in Listing 1-8.

LISTING 1-8 fAddOne custom function

```
(MyNumber as number) as number =>
let
    Source = MyNumber + 1
in
    Source
```

Note that the query's icon is a function icon, and instead of data preview you see a prompt to enter parameter and invoke it. If you enter **2** and click **Invoke**, a new query, **Invoked Function**, is created with a 123 icon, meaning it is a number. This example shows that not all queries

in Power Query need to be tables; queries can return scalar values or be functions, among other things. We can now go back to the Target query and select the Calendar Year column, then click **Add Column** > **General** > **Invoke Custom Function**. In the **Invoke Custom Function** dialog box, define the name of the new column and select a function to apply from a drop-down list. Because we have only one custom function defined, there is only one choice. Once you make a selection, the **Calendar Year** column will be chosen automatically. If needed, this can be overridden by either selecting a different column from the drop-down list, or a static number can be specified by choosing **Decimal Number** to the left of the drop-down list. Clicking **OK** creates a new column and a new query step. As before, this step should be removed.

Conditional Column

The Conditional Column dialog box allows you to create a column based on specified rules in "if[nd]then[nd]else" fashion. For example, if Calendar Year is greater than 2014, the output can be "After 2014," otherwise "Before 2015." We will review this option in more detail later in the chapter.

Index ColumnPower Query also allows you to add an index column to your tables. There are three options when you select **Add Column** > **General** > **Index Column**:

- From 0
- From 1
- Custom

The first two options add a column starting from either 0 or 1 and incrementing by 1 with each row. If you select **Custom**, you can specify your own starting index and increment in the **Add Index Column** dialog box. This feature can be useful when you need to track the order of events: once you load your data, the row order is not guaranteed, and you can refer to an index column in this case.

Another way you can add a new column is by duplicating an existing column. For instance, if you plan to modify some values in a column but still need the old ones to be present, you can duplicate a column.

Note that there are many other buttons in the **Add Column** tab, and all of them are also present in the **Transform** tab. The buttons in the **Transform** tab modify values and keep the number of columns as-is, while their Add Column counterparts add new columns with modified values. These buttons are categorized into three groups: Text, Number, and Date/Time functions. With them, you can perform transformations on values either in existing columns (**Transform** tab) or add new columns based on existing ones (**Add Column** tab). For example, you can extract the first three characters from a text string, extract month name from a datetime value, or round a decimal number to one decimal number. It is worth reviewing the functionality on your own.

MORE INFO **COMMON QUERY TASKS**

For more examples on combining data, performing transformations, and using formulas, see "Common query tasks in Power BI Desktop" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-common-query-tasks/*.

Apply business rules

With conditional columns mentioned in the previous section, you can apply business rules to your data. For example, let's assume that the management of Wide World Importers is interested in viewing data for the contiguous United States and non-contiguous states separately. Furthermore, we would like to exclude unknown cities. We can create a conditional column based on the State Province column from the City table, and we can use the new column in our reports later.

Select **Add Column** > **General** > **Conditional Column** and type **CONUS** in the new column name field. Because we want to exclude Alaska, Hawaii, Puerto Rico (U.S. Territory), Virgin Islands (U.S. Territory), and N/A, we need to create a column that contains 1 for all states except the ones mentioned previously. The rules can be defined as follows:

- If State Province equals "Alaska" then 0
- Else If State Province equals "Hawaii" then 0
- Else If State Province equals "Puerto Rico (US Territory)" then 0
- Else If State Province equals "Virgin Islands (US Territory)" then 0
- Else If State Province equals "N/A" then 0
- Otherwise 1

The Conditional Column window should look like Figure 1-32.

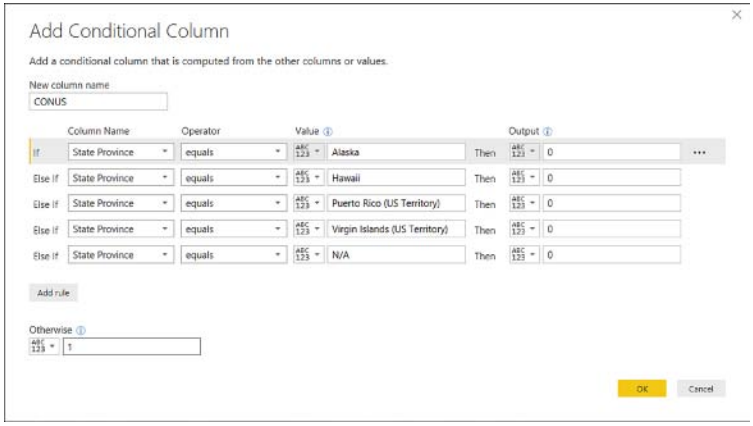


FIGURE 1-32 Add Conditional Column window

After we click OK, we need to apply the Whole Number data type to the column.

Change data format to support visualization

Power BI works best with tabular data when each metric is in its own column. Occasionally you might get heavily pivoted data, which requires complex transformations to make visualization easier. In the following example, we will be working with sample data from ChangeFormat.xlsx. The file can be found in this book's companion files.

The file contains a data array with three levels of headers: metric, year, and month. In addition to that, there are two attributes: Sales Territory and State Province. There are two metrics—Sales Amount and Tax Rate—and the two are in different scales. Tax Rate is a percentage, while Sales Amount is in dollars. This format makes analysis and visualization very difficult. For Power BI, the following columns would be preferable: Date, Sales Territory, Sales Province, Sales Amount, and Tax Rate.

We can start by connecting to the Excel file. There is only one sheet with no ranges formatted as tables or named ranges. Once we select the sheet, we can click **OK** and then disable the load of the query, because we will not need it in our data model. We can rename the query to **ChangingFormatReview**. Because the automatic **Changed Type** step does not add any value, we can remove it in this case. In this example, we are going to take a similar approach we took with 2015–2016 targets earlier in this section.

Note that we need to fill the nulls for both headers and attribute columns. First, while holding the Ctrl key, we can select the two left-most columns, **Column1** and **Column2**. Right-click the header of either of them and select **Fill** > **Down**. Next, click **Transform** > **Table** > **Transpose** and fill down the first three columns. Because unpivoting data at this stage is not going to help us, we need to merge the first three columns. We will need to split them later, so we should use a custom separator that cannot be found in data. In this case, I recommend using the rarely used caret character: ^. We can leave the name as is—**Merged**. Once this is done, we need to transpose our table again and click **Transform** > **Table** > **Use First Row As Headers**. The **Changed Type** step can be removed once again.

Now that we have headers, we can select the first two columns and unpivot all other columns. The third column, **Attribute**, should be split back into three, by the caret custom delimiter. After splitting the column, we should remove the **Changed Type** yet again.

At this stage, we can note that the two metrics that we have, Sales Amount and Tax Rate, are both contained within the same column, Value. The two metrics should be separated into two columns. This can be achieved by pivoting the Attribute.1 column. Select the column and click **Transform** > **Any Column** > **Pivot Column**. In the Pivot Column window, we need to select the **Values** column first, which, in our case, is the Value column. Also, in **Advanced Options**, we should select **Don't Aggregate As The Aggregate Value Function**. The settings window should look like Figure 1-33.

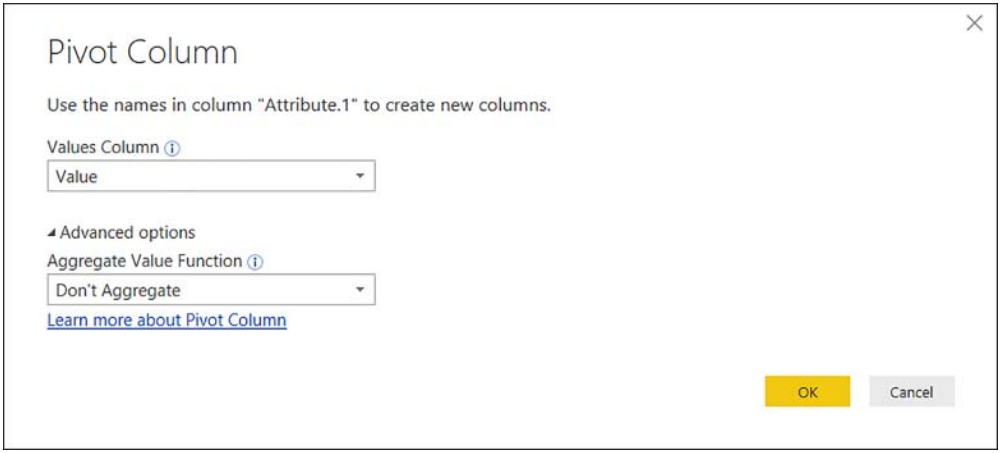


FIGURE 1-33 Pivot Column dialog box

MORE INFO **PIVOTING AND UNPIVOTING DATA IN POWER QUERY**

For more examples of using the Pivot and Unpivot functions in Power Query, you can read a blog post by Reza Rad at *http://radacad.com/pivot-and-unpivot-with-power-bi*.

The values in the Sales Amount column have a "k" suffix, meaning they are in thousands. This should be addressed by replacing the "k" symbol with nothing by right-clicking anywhere on the column and selecting **Replace Values**. We can type **k** without quotation marks in **Value To Find** and click **OK**. We will multiply the values by 1,000 later.

The next step is to merge the Year and Month values into a Date column. This can be achieved by merging the Attribute.2 and Attribute.3 columns into a new column called Date with a dash as a custom separator. In this case, the order in which you select columns is not important because Power Query will be able to parse dates either way. This creates a column of type text. We should transform column names and data types according to Table 1-4.

TABLE 1-4 Column names and data types

Old Name	New Name	Data Type
Sales Territory^Sales Territory^Sales Territory	Sales Territory	Text
State Province^State Province^State Province	State Province	Text
Date	Date	Date
Sales Amount	Sales Amount	Fixed Decimal Number
Tax Rate	Tax Rate	Percentage

Now that the Sales Amount column is of numeric type, multiply it by 1,000 by selecting it, clicking **Transform** > **Number Column**> **Standard** > **Multiply,** and entering 1,000 as Value. Finally, we can rearrange the columns by right-clicking on the header of the Date column and selecting **Move** > **To Beginning**. If you open **Advanced Editor**, you should see code similar to Listing 1-9.

LISTING 1-9 Full M code of the ChangingFormatReview query

```
let
    Source = Excel.Workbook(File.Contents("C:\Companion\ChangeFormat.xlsx"), null,
 true),
    SalesExport_Sheet = Source{[Item="SalesExport",Kind="Sheet"]}[Data],
    #"Filled Down" = Table.FillDown(SalesExport_Sheet,{"Column1", "Column2"}),
    #"Transposed Table" = Table.Transpose(#"Filled Down"),
    #"Filled Down1" = Table.FillDown(#"Transposed Table",{"Column1", "Column2",
 "Column3"}),
    #"Merged Columns" = Table.CombineColumns(Table.TransformColumnTypes(#"Filled Down1",
 {{"Column2", type text}, {"Column3", type text}}, "en-AU"),{"Column1", "Column2",
 "Column3"},Combiner.CombineTextByDelimiter("^", QuoteStyle.None),"Merged"),
    #"Transposed Table1" = Table.Transpose(#"Merged Columns"),
    #"Promoted Headers" = Table.PromoteHeaders(#"Transposed Table1",
 [PromoteAllScalars=true]),
    #"Unpivoted Other Columns" = Table.UnpivotOtherColumns(#"Promoted Headers", {"Sales
 Territory^Sales Territory^Sales Territory", "State Province^State Province^State
 Province"}, "Attribute", "Value"),
    #"Split Column by Delimiter" = Table.SplitColumn(#"Unpivoted Other Columns",
 "Attribute", Splitter.SplitTextByDelimiter("^", QuoteStyle.Csv), {"Attribute.1",
 "Attribute.2", "Attribute.3"}),
    #"Pivoted Column" = Table.Pivot(#"Split Column by Delimiter", List.Distinct(#"Split
 Column by Delimiter"[Attribute.1]), "Attribute.1", "Value"),
    #"Replaced Value" = Table.ReplaceValue(#"Pivoted
 Column","k","",Replacer.ReplaceText,{"Sales Amount"}),
    #"Merged Columns1" = Table.CombineColumns(#"Replaced Value",{"Attribute.3",
 "Attribute.2"},Combiner.CombineTextByDelimiter("-", QuoteStyle.None),"Date"),
    #"Renamed Columns" = Table.RenameColumns(#"Merged Columns1",{{"Sales Territory^Sales
 Territory^Sales Territory", "Sales Territory"}, {"State Province^State Province^State
 Province", "State Province"}}),
    #"Changed Type" = Table.TransformColumnTypes(#"Renamed Columns",{{"Sales Territory",
 type text}, {"State Province", type text}, {"Date", type date}, {"Sales Amount",
 Currency.Type}, {"Tax Rate", Percentage.Type}}),
    #"Multiplied Column" = Table.TransformColumns(#"Changed Type", {{"Sales Amount",
 each _ * 1000, Currency.Type}}),
    #"Reordered Columns" = Table.ReorderColumns(#"Multiplied Column",{"Date", "Sales
 Territory", "State Province", "Sales Amount", "Tax Rate"})
in
    #"Reordered Columns"
```

MORE INFO **CLEANING IRREGULARLY FORMATTED DATA**

For more examples on how to change data format to support visualization, see the "Cleaning Irregularly Formatted Data" page on Power BI Guided Learning at *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-1-5-cleaning-irregular-data/*.

Working with query parameters

In Power Query, a parameter is a query that returns a single value, which may have a specified data type. Parameters can be useful in many scenarios. For example, you can filter data in tables by parameter; changing the parameter value will change the data a table returns. We are going to review the usage of parameters by following an example.

One of the ways to create a parameter is by clicking **Home** > **Parameters** > **Manage Parameters** > **New Parameter**. Alternatively, right-click on a blank space in the **Queries** pane and select **New Parameter**. This will open the Parameters window, where you will need to specify settings for the new parameter. If we had some parameters already, they would be displayed on the left.

With our parameter, we are going to filter both **Date** and **Sale** tables by a starting date. Therefore, we can call the new parameter StartDate. The Description field is optional, and we should leave the Required check box enabled because filtering dates by a null value is not going to work. In the **Type** drop-down list, we should select **Date**. In the **Suggested Values**, there are three options:

- **Any value:** Lets Let's you enter any value.
- **List of values:** Lets Let's you pre-define a list of parameter values, which later translates into a drop-down list, from which you can select a value. You will need to define both the default and current values. The default value is used when you export your Power BI Desktop file as a Power BI template.
- **Query** Lets you reference a query that returns a list, from which you can later pick a parameter value; specifying the current value is also required. In our case, we can keep the default option, **Any Value**, selected and enter **1/1/2014** as the current value, then click **OK**. We can now see our parameter in the Queries pane with an icon different from other queries.

To use a parameter, go to the **Date** query, click the drop-down arrow in the header of the **Date** column and select **Date Filters** > **Custom Filter**. We need to specify one rule: "is after or equal to" and click on the calendar icon to the right. In the drop-down list, we can pick between a date value (currently selected), a parameter, or creation of a new parameter. Select **Parameter**, and then the **StartDate** parameter will be automatically selected. Clicking **OK** will filter the table. Note that Query Folding takes place: if you right-click on the **Filtered Rows** step and select **View Native Query**, you will see the native query in Listing 1-10, with the filtering translated natively into the WHERE statement, highlighted in bold below.

LISTING 1-10 Date native query

```
select [_].[Date],
    [_].[Day Number],
    [_].[Day],
    [_].[Month],
    [_].[Short Month],
    [_].[Calendar Month Number],
    [_].[Calendar Month Label],
    [_].[Calendar Year],
    [_].[Calendar Year Label],
    [_].[Fiscal Month Number],
    [_].[Fiscal Month Label],
    [_].[Fiscal Year],
    [_].[Fiscal Year Label],
    [_].[ISO Week Number]
from [Dimension].[Date] as [_]
where [_].[Date] >= convert(datetime2, '2014-01-01 00:00:00')
```

We can now apply the same filter to the Sale query; we should apply the same filter to the Invoice Date Key column. Just as in case of the Date query, the filtering is correctly translated into a native query, even though the query itself is referencing the SaleInitial query. The native query can be seen in Listing 1-11.

LISTING 1-11 Sale native query

```
select [_].[City Key],
    [_].[Customer Key],
    [_].[Bill To Customer Key],
    [_].[Stock Item Key],
    [_].[Invoice Date Key],
    [_].[Delivery Date Key],
    [_].[Salesperson Key],
    [_].[WWI Invoice ID],
    [_].[Quantity],
    [_].[Unit Price],
    [_].[Tax Rate],
    [_].[Total Excluding Tax],
    [_].[Tax Amount],
    [_].[Profit],
    [_].[Total Including Tax]
from
(
    select [City Key],
        [Customer Key],
        [Bill To Customer Key],
        [Stock Item Key],
        [Invoice Date Key],
        [Delivery Date Key],
        [Salesperson Key],
        [WWI Invoice ID],
        [Quantity],
        [Unit Price],
```

```
        [Tax Rate],
        [Total Excluding Tax],
        [Tax Amount],
        [Profit],
        [Total Including Tax]
    from [Fact].[Sale] as [$Table]
) as [_]
where [_].[Invoice Date Key] >= convert(datetime2, '2014-01-01 00:00:00')
```

We can edit the StartDate parameter by selecting it and by clicking **Manage Parameter**. Changing the year from **2014** to **2015** changes the native queries in both Date and Sale queries. Listing 1-12 shows fragments of both.

LISTING 1-12 Fragments of Date and Sale queries

```
// Fragment of Date query
select [_].[Date],
    . . .
from [Dimension].[Date] as [_]
where [_].[Date] >= convert(datetime2, '2015-01-01 00:00:00')

// Fragment of Sale query
select [_].[City Key],
    . . .
    from [Fact].[Sale] as [$Table]
) as [_]
where [_].[Invoice Date Key] >= convert(datetime2, '2015-01-01 00:00:00')
```

Note that we only had to change the parameter once to update both queries. There are other scenarios in which parameters can be useful.

MORE INFO **POWER BI DESKTOP QUERY PARAMETERS**

For more examples and use cases of query parameters in Power BI Desktop, you can refer to a series of blog posts by Soheil Bakhshi:

- ***http://biinsight.com/power-bi-desktop-query-parameters-part-1/***
- ***http://biinsight.com/power-bi-desktop-query-parameters-part2-dynamic-data-masking-and-query-parameters/***
- ***http://biinsight.com/power-bi-desktop-query-parameters-part-3-list-output/***

Creating custom functions

Apart from coding custom functions in M, you can create custom functions using parameters. For example, let's duplicate the Date query. This creates a query called Date (2). We should disable its load, move it to the Other Queries group and rename it to **FilteredDate**. Right-click the **FilteredDate** query and select **Create Function**. In the **Create Function** dialog box, enter **fDate** in the Function **Name** field and click **OK**. This creates a new query group called **fDate**, which consists of the FilteredDate query and the fDate custom function.

You can test how the fDate function works by selecting it and entering **2016** as the **StartDate** parameter; Power Query will interpret it as 1 January 2016. Note that you are no longer tied to the value of the StartDate parameter we defined earlier. The parameter in the fDate function only shares the name, but it can be of any value. After you click **Invoke**, a new query called **Invoked Function** is created, which returns the Date table filtered to dates after 1 January 2016. This time, however, the filtering is not translated into a native query. The **View Native Query** selection is disabled.

If you attempt to edit the **fDate** function directly, either in the Advanced Editor or the Formula Bar, you will get the message seen in Figure 1-34.

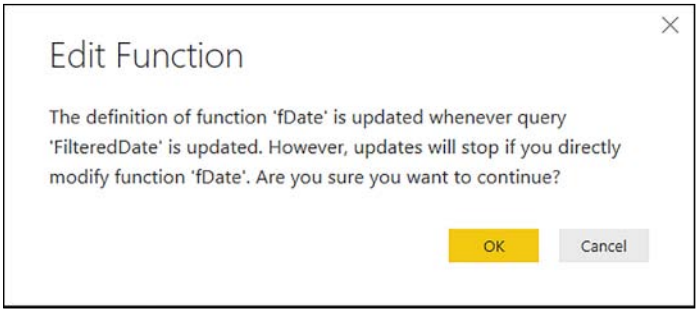


FIGURE 1-34 Edit Function dialog box

What this message means is that if you want to modify the function, you should modify the FilteredDate query instead, because all of the changes will be translated to the fDate function. Indeed, if we go to **FilteredDate**, select the **Date** column and remove all other columns, then go to the **Invoked Function** query, we will see that it has only one column, even though we did not modify either it or the fDate function directly.

MORE INFO **CUSTOM FUNCTIONS IN POWER BI**

Custom functions are particularly useful when you need to apply the same transformation multiple times. By encapsulating your transformation steps into a single query, you make your code easier to maintain. For more information on custom functions in Power BI, you can refer to Reza Rad's blog entry, "Custom Functions Made Easy in Power BI Desktop" at *http://radacad.com/custom-functions-made-easy-in-power-bi-desktop*.

Privacy levels

When you combine data from different data sources, it is important to set the privacy levels correctly. Privacy levels determine the rules according to which data will be combined. These rules may affect the performance of queries, and in some cases, queries will not be executed at all if it is not permitted by privacy levels. To illustrate what happens in an example, we are going to filter the Customer table by a parameter value from the Target table.

First, right-click on the **Target** table and select **Reference**. This will create a new query in the Targets queries group, called Target (2). Rename the query to **DistinctCustomer** and disable its loading. Next, right-click on the header of the **Bill To Customer** column and select **Drill Down**. This will turn the column into a list. Finally, we want to keep distinct values only, which can be accomplished by clicking **List Tools** > **Transform** > **Manage Items** > **Remove Duplicates**. The full query code should be the same as shown in Listing 1-13.

LISTING 1-13 DistinctCustomer query code

```
let
    Source = Target,
    #"Bill To Customer" = Source[Bill To Customer],
    #"Removed Duplicates" = List.Distinct(#"Bill To Customer")
in
    #"Removed Duplicates"
```

Second, click **Home** > **Parameters** > **Manage Parameters** > **New Parameter**. The new parameter should be called **CustomerParameter**, and its type should be **Text**. Select **Query for Suggested Values** and then select the **DistinctCustomer** query from the drop-down list. Type **N/A** in the **Current Value** field.

We can now use this parameter to filter the Customer table: go to the Customer table and click on the **AutoFilter** (arrow) button in the **Bill To Customer** header, then select **Text Filters** > **Equals** in the **Filter Rows** dialog box, click on the top **ABC** drop-down list and select **Parameter** (**CustomerParameter** should be selected automatically). After clicking **OK**, a message will prompt you to set privacy levels, as shown in Figure 1-35.

FIGURE 1-35 Data privacy prompt

If you do not see the prompt from Figure 1-35, it means you have already combined data from your drive and your SQL Server; the permissions you set can be cleared in **Home** > **Data Sources** > **Data Source Settings**. You will see a list of data sources used in the current file; to clear the permissions, below the list, click on the arrow next to **Clear Permissions** and select **Clear All Permissions**, then confirm by selecting Delete in the Clear All Permissions dialog box.

When we click **Continue** in the data privacy prompt from Figure 1-35, we open the **Privacy Levels** dialog box, where we are prompted to select privacy levels. The dialog box can be seen in Figure 1-36.

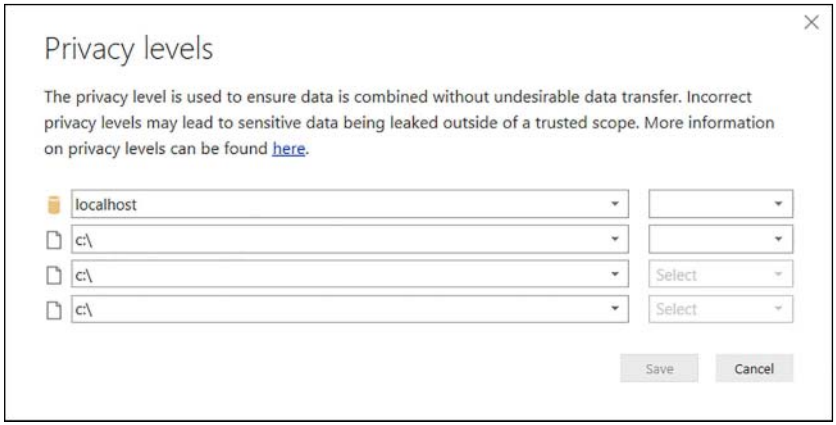


FIGURE 1-36 Privacy Levels dialog box

You can notice in Figure 1-36 that we have the C: drive repeated three times, though you can select the privacy level only once. This is because each drop-down list corresponds to a specific data source. For example, expanding the localhost drop-down list, shows both **localhost** and **localhost;WideWorldImportersDW** as options. This means that you can set the privacy level either to a specific data source (WideWorldImportersDW database, for instance), or any level above it (such as the whole localhost SQL Server). Because the Target query is composed of three files—Target.txt, Target20152016.xlsx, and TargetQuantity.txt—we can pick privacy levels for each file individually, or specify a common privacy level for any level above, such as for the whole C: drive.

In the drop-down lists on the right, you can select the following privacy levels:

- **Public** This option should be used for publicly accessible sources, such as Wikipedia pages.
- **Organizational** This can be used for data sources accessible to others within your network, such as a corporate database. This privacy level is isolated from the Public data sources, but it is visible to other Organizational data sources.
- **Private** Should be used for confidential or sensitive information, such as payroll information. This privacy level is completely isolated from all data sources, including other data sources marked as Private.

For now, select **Organizational** for both localhost and C: drive. If you previously cleared data source settings, you might need to specify credentials for your database.

At this stage, you will see the Customer table filtered to one row. The full M query can be seen in Listing 1-14.

LISTING 1-14 Filtered Customer M query

```
let
    Source = Sql.Databases("localhost"),
    WideWorldImportersDW = Source{[Name="WideWorldImportersDW"]}[Data],
    Dimension_Customer = WideWorldImportersDW{[Schema="Dimension",Item="Customer"]}
[Data],
    #"Filtered Rows" = Table.SelectRows(Dimension_Customer, each [Bill To Customer]
 = CustomerParameter)
in
    #"Filtered Rows"
```

The corresponding native query can be seen in Listing 1-15.

LISTING 1-15 Filtered Customer native query

```
select [_].[Customer Key],
    [_].[WWI Customer ID],
    [_].[Customer],
    [_].[Bill To Customer],
    [_].[Category],
    [_].[Buying Group],
    [_].[Primary Contact],
    [_].[Postal Code],
    [_].[Valid From],
    [_].[Valid To],
    [_].[Lineage Key]
from [Dimension].[Customer] as [_]
where [_].[Bill To Customer] = 'N/A'
```

Note that in the native query, filtering is translated with the WHERE clause. This is made possible because both data sources are marked with the Organizational privacy level.

We can now review what happens when we change the privacy level of one of the data sources to Private. First, click **Home** > **Data Sources** > **Data Source Settings**, then select **Target.txt** and click **Edit Permissions**. In the **Edit Permissions** dialog box, you will see **None** selected. This means that for this data source specifically, no privacy level has been selected. Therefore, it inherits its privacy level from the parent directory, which, in our case is drive C: with Organizational privacy level. Select **Private** in the drop-down list and click **OK**, then Close.

After you click **Home** > **Query** > **Refresh Preview**, you will see that the query still executes, but at step **Filtered Rows**, no query folding takes place. Instead, query folding instead ends at the previous step, Navigation. What this means is that no data from the Target table is sent to SQL Server; instead, the whole Customer table is downloaded from the server, then filtering is done inside Power Query. As a result, performance is degraded, but data is not leaked outside of Power Query. Even if a database administrator would run a Profiler trace, he or she would not be able to check which values are contained in our files.

While this is an artificial example, it illustrates how privacy levels work. If you are confident that privacy is not an issue with our data, you can disable privacy settings in **File** > **Options and Settings** > **Options**. You can set privacy settings either globally—for every file—or for this file only in the **Global** and **Current File** sections, respectively.

MORE INFO **PRIVACY LEVELS IN POWER BI DESKTOP**

For a general overview of privacy levels, see "Power BI Desktop privacy levels" at *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-privacy-levels/*. Additionally, Chris Webb has written a series of detailed blog posts on privacy levels:

- **"Data Privacy Settings In Power BI/Power Query, Part 1: Performance Implications" at *https://blog.crossjoin.co.uk/2017/05/24/data-privacy-settings-in-power-bipower-query-part-1-performance-implications/*.**
- **"Data Privacy Settings In Power BI/Power Query, Part 2: Preventing Query Execution" at *https://blog.crossjoin.co.uk/2017/05/31/data-privacy-settings-in-power-bipower-query-part-2-preventing-query-execution/*.**
- **"Data Privacy Settings In Power BI/Power Query, Part 3: The Formula.Firewall Error" at *https://blog.crossjoin.co.uk/2017/06/26/data-privacy-settings-in-power-bipower-query-part-3-the-formula-firewall-error/*.**
- **"Data Privacy Settings In Power BI/Power Query, Part 4: Disabling Data Privacy Checks" at *https://blog.crossjoin.co.uk/2017/07/04/data-privacy-settings-in-power-bipower-query-part-4-disabling-data-privacy-checks/*.**
- **"Data Privacy Settings In Power BI/Power Query, Part 5: The Inheritance Of Data Privacy Settings And The None Data Privacy Level" at *https://blog.crossjoin.co.uk/2017/07/10/data-privacy-settings-in-power-bipower-query-part-5-the-inheritance-of-data-privacy-settings-and-the-none-data-privacy-level/*,**

Skill 1.3: Cleanse data

Occasionally, you may need to cleanse the data you are using, unless you are using a data source where data quality is managed by someone. In this chapter, we have briefly covered various techniques with which you can clean your data, and in this section, we are going to review more of them.

This section covers how to:

- Manage incomplete data
- Meet data quality requirements

Manage incomplete data

Earlier in this chapter, we reviewed the Fill Down feature of Power Query. It has a Fill Up counterpart as well: Note that the feature works only on null values, not blank ones, or zero-length strings.

There are other ways in which you can add missing values to your data. For instance, you may choose to replace nulls with a column average rather than values from above or below. To review the process, start by connecting to ReplaceWithAverage.csv from this book's companion files. Keep the connection settings as is and click **OK**. You can see that there are several values missing. To replace nulls with an average, calculate the average first. To do this, right-click on the **Average Price** column and then select **Add as New Query**. This creates a new query with the column transformed into a list. Note that the icon of this query is different now. Lists are not tables; you can see that the **Transform** and **Add Column** tabs are inactive. However, there is a new **Transform** tab that is designated as **List Tools**, and it allows you to convert this list into a table. Apart from that, you also have an option to keep top, bottom, or range of items; remove top, bottom, or alternate items; remove duplicates and reverse the order of the items; sort; or perform statistical aggregations on the items. In this case, we want to select **Statistics** > **Average**. This transforms the query into a scalar value, as can be seen from its 123 icon. This value can now be used to replace nulls with it.

We can go back to the ReplaceWithAverage query and add a **Custom Column**, which should have the formula from Listing 1-16.

LISTING 1-16 Custom column formula to replace nulls with pre-calculated averages

```
=if [Average Price] is null then #"Average Price" else [Average Price]
```

A similar technique—checking if a column value is null—can be useful when dealing with nested tables, as some hierarchy levels often contain nulls instead of values.

In M, you refer to columns by enclosing them in square brackets; it does not matter if there are any spaces or not. You can refer to queries, including formulas and parameters, by referencing their names directly—unless there are special characters in the names, such as spaces. In this case, you need to enclose the name in double quotation marks and add a hash (#) prefix. So, the formula, translated into English, reads: "If the Average Price column is null, then use the Average Price query, otherwise keep the value from Average Price column."

Meet data quality requirements

Power Query has features that can help you with handling errors, duplicate values, and undesirable characters. Errors can be filtered out, replaced, or otherwise dealt with, depending on the objective.

Handling error values

Connect to Target.txt again by selecting **Home** > **Recent Sources** > **Target.txt**, and we can accept the default settings. This creates a query called **Target (2)**, which we should rename to **ErrorHandling**. First, we should promote headers, which creates another step, **Changed Type1**. To generate an error, we can change the type of the CalendarYear column to **Whole Number**. We will then see the **Change Column Type** dialog box from Figure 1-37.

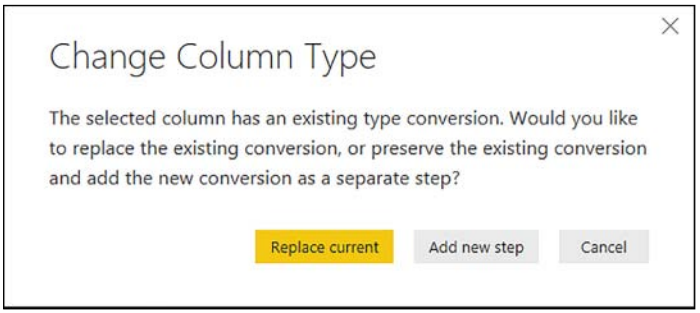


FIGURE 1-37 Change Column Type dialog box

In our case, we can select **Replace Current**. Note that row 13 now has an error, because Power Query tried to convert ***To be confirmed** into a number. There are at least three ways to deal with the error.

First, you can right-click on the **ColumnYear** header and select **Remove Errors**. Alternatively, you can select the column and then click **Transform** > **Reduce Rows** > **Remove Rows** > **Remove Errors**. This will remove the row completely. Because there is only one error in the whole row, you can achieve the same effect by clicking the table icon above row numbers and select **Remove Errors**. This removes all rows where at least one column contains an error.

Second, after right-clicking on the header, you can select **Replace Errors**. In the **Replace Errors** dialog box, you will be able to specify a replacement value. For instance, you can type "null" without quotation marks, and this will replace errors with nulls. This procedure will keep the row but replace the error with a value of your choice.

Third, you can create a custom column that checks whether there is an error in the column. For this, you will need to use the "try otherwise" construct. Select **Add Column** > **General** > **Custom Column** and type the formula from Listing 1-17.

LISTING 1-17 Custom column formula that handles errors

```
=try [CalendarYear] otherwise null
```

This formula checks whether the value in the CalendarYear column is an error, and if it is, then it returns a null value; if the CalendarYear value is not an error, then this value is returned.

Removing extra spaces and non-printable characters

If you have extra spaces in your text strings, Power Query can trim them with a Trim transformation. To trim spaces in a column, right-click its header and select **Transform** > **Trim**. Alternatively, you can select a column and click **Transform** > **Text Column** > **Format** > **Trim**. The function removes all spaces from both sides of the string.

If you are using a SQL database as a data source, the function does not break Query Folding. For example, if we trim spaces in the Employee column from Employee table, it will be translated as shown in Listing 1-18.

LISTING 1-18 Fragment of a query with Trim translated to SQL

```
select [_].[Employee Key] as [Employee Key],
    [_].[WWI Employee ID] as [WWI Employee ID],
    ltrim(rtrim([_].[Employee])) as [Employee],
    [_].[Preferred Name] as [Preferred Name],
    [_].[Is Salesperson] as [Is Salesperson],
    [_].[Photo] as [Photo],
    [_].[Valid From] as [Valid From],
    [_].[Valid To] as [Valid To],
    [_].[Lineage Key] as [Lineage Key]
from [Dimension].[Employee] as [_]
```

If you need to remove non-printable characters from a column, you can select **Transform** > **Clean**. Note that this function prevents Query Folding, so it is best to do it after all transformations that support Query Folding.

EXAM TIP

The exam does not test your knowledge on advanced M syntax, but you should be familiar with functions that are generated when working with the user interface. Every step's formula can be seen in Formula Bar when a step is selected. For a general overview of functions, see "Understanding Power Query M functions" at *https://msdn.microsoft.com/en-us/library/mt185361.aspx*.

For the complete function reference, see "Power Query M function reference" at *https://msdn.microsoft.com/en-us/library/mt779182.aspx*.

Thought experiment

In this thought experiment, demonstrate your skills and knowledge of the topics covered in this chapter. You can find the answer to this thought experiment in the next section.

You are the BI developer at Contoso responsible for creating Power BI reports. The current database environment includes an on-premise SQL Server 2017 OLTP system and Azure SQL Data Warehouse that contains all historical data since 1990 and synchronizes with the OLTP system every five minutes. Azure SQL Data Warehouse contains over 500 GB of data.

Additionally, every month, the Sales Planning department produces Excel files with sales targets for each customer and product category. These files are produced by planning software that always uses the same format. A sample file is shown in Figure 1-38. The files are stored in a folder on a network drive to which you have been granted access.

	A	B	C	D	E	F	G	H
1		Cameras and camcorders	Cell phones	Computers	Home Appliances	Music, Movies and Audio Books	TV and Video	Grand Total
2	St. LeonardsCompany	4424	467.88	3104.2	5149.86	2349.924444	239.98	15735.844
3	HamburgCompany1	10156	289.71	2781.35	3556	1659.933333	1409.91	19852.903
4	Hervey BayCompany	4916	136.57	11697.55	14116.74667	13.89	3948	34828.757
5	LeedsCompany	3843	4699.51	1317	27096.69	10885.89667	4904	52746.097
6	LondonCompany	8919.12	4508.76	10383.95	24379.81	657	18363.03864	67211.679
7	MeltonCompany	436.2	2384	51.8	4499.86	1712.9	5583.475926	14668.236
8	North RydeCompany	5672	2030	408	4552.79	2524.283333	5952.767778	21139.841
9	OxfordCompany	605.9	1465	33.99	7614.61		2553	12272.5
10	PerthCompany		139.86	1163.8	6300.2	2739.86	1674	12017.72
11	Port MacquarieCompany	8645	3601.78	736.5	3871.85		269.7	17124.83
12	VersaillesCompany	13350	241.77	6230	269.91		NONE	20091.68
13	WoodburnCompany	48.65	154	4049.25	1199.9	479.97		5931.77
14	Grand Total	61015.87	20118.84	41957.39	102608.2267	23023.65778	44897.87235	293621.86

FIGURE 1-38 Sample sales targets file, 2018 May.xlsx

The management requested two Power BI reports to be produced: one that shows all historical data, including transactions that happened in the past 10 minutes, and another report that tracks sales targets versus actual figures for the past 12 months.

Based on background information and business requirements, answer the following questions:

1. Which data connectivity mode should you use for each report?
   - **A.** DirectQuery for both
   - **B.** Import data for both
   - **C.** DirectQuery for the historical data report, Import data for the sales targets report
   - **D.** DirectQuery for the sales targets report, Import data for the historical data report
2. How do you connect to Excel files that contain target figures? The solution should involve minimum manual work when new Excel files are created.
   - **A.** Connect to the files with Folder connector and use the Combine Binaries functionality
   - **B.** Connect to a new file each month, perform transformations, and use the Append function in Power Query Editor to combine all targets in the same table
3. After connecting to all the target Excel files, which of the following is going to transform data into a tabular shape that Power BI works best with?
   - **A.** Select first column, then click Pivot Column
   - **B.** Select first column, then Unpivot Other Columns
   - **C.** Select first column, then Unpivot
   - **D.** Transpose table
4. You need to filter out totals from target figures. Which function does that?
   - **A.** Table.Filter
   - **B.** CALCULATETABLE
   - **C.** Table.SelectRows
   - **D.** Table.FilterRows

5. After transforming the targets table, you have selected Fixed Decimal Number for the target figures column. Now there is one error in the column. Which of the following is NOT going to remove it in any way?
   A. Right-click on the header of the column and select Remove Errors
   B. Right-click on the header of the column, select Replace Errors, specify null
   C. Click **Home** > **Remove Rows** > **Remove Errors**.
   D. Click on the AutoFilter button of the column and de-select the error

Thought experiment answers

1. The answer is **C**. For the historical data report; you need to select the DirectQuery connectivity mode because the report needs to show the latest data, as well as all the available historical data, which is too large to fit into memory. Azure SQL Data Warehouse supports DirectQuery, and other data sources are not required, making DirectQuery a viable option. For the sales targets report, you need to combine actual data from Azure SQL Data Warehouse with Excel files. Furthermore, you will need to do transformations on Excel files, which leaves importing data as the only available option.
2. The answer is **A**. The Folder connector performs transformations on files that have the same format automatically. You need to define the transformations only once; then you can only refresh data when new files are created. This option is much less laborious than connecting to each file individually and performing transformations every time.
3. The answer is **B**. Option A requires a values column to be selected, and there is more than one—one for each product category. Option C will keep all the values columns in place. Option D will put customers on columns and product categories on rows. Option B will correctly transform the table into a table with three columns, which will contain Customer, Product Category, and Target values.
4. The answer is **C**. Options A and D do not exist. Option B is a DAX function.
5. The answer is **D**. When you have an error in a column, it is not possible to filter it out using AutoFilter because the error does not show in the results. All other options will work.

Chapter summary

- In most cases, the development of a Power BI Desktop report starts by creating a data source, which can be a relational database, file, folder, Excel, web service, SQL Server Analysis Services database, among many others. Power BI Desktop also supports generic data interfaces and custom data connectors, which makes the list of available data sources virtually unlimited.
- Many relational databases share the same steps that you take when you connect to them: first, you specify a server and, in some cases, a database name. Power BI Desk-

top will also import relationships between objects if they exist in the database and you chose to include relationship columns in the initial settings dialog box. Next, you need to specify authentication mode and credentials. You are then taken to the Navigator window, where you select the objects you want to include in your data model. Some data sources support objects other than tables and views. Once you have selected all desired objects, you can either load data from the objects right away, or edit it in Power Query Editor before loading.

- Power BI Desktop performs best and allows you to use all of its features when you import data. In some cases, it is not feasible—for example, when there is too much data to import, or when data is updated very frequently, and business requirements demand always showing the latest data. These issues can be addressed if the data source supports the DirectQuery connectivity mode. Some, but not all, databases support DirectQuery. With DirectQuery, no data is imported into Power BI. Instead, all data remains in the source, and every time Power BI needs to calculate values, it sends queries in data source's native query language. In some cases, you can apply certain types of transformations that can be translated to the native query language. There is a special case of DirectQuery called Live Connection, which is available with SQL Server Analysis Services (SSAS) and Power BI Service. If you are using either DirectQuery or Live Connection, you can only use one data source. You can switch from DirectQuery to Import mode. However, switching from Live Connection to Import mode is currently not supported.
- It is possible to either import files in Power BI Desktop individually or connect to a folder that contains files, given that they share the same format. Currently, files of the following types can be combined using Power Query Editor: Excel, Text/CSV, XML, and JSON. Besides importing data from Excel files, you can also import its workbook contents, which imports Power Query queries, Power Pivot data models, and Power View worksheets. Not all Power View visuals are currently supported in Power BI Desktop, and unsupported visuals result in error messages. The best way to migrate an existing Power Pivot data model to Power BI Desktop is by importing it.
- In addition to on-premise data sources, Power BI Desktop can connect to cloud data services, such as Azure SQL Database and SharePoint Online. Furthermore, you can connect to files and pages located on the Internet by using the Web Connector.
- Power Query Editor uses a strongly typed, functional language called M, and it is rich in features with which you can perform basic and advanced transformations. Each transformation is recorded in a step, which can be reordered or deleted. The full query code can be edited in the Advanced Editor. Query dependencies can be seen in the Query Dependencies view.
- The following are the most common tasks you can perform with Power Query Editor user interface:
  - Filter data by reducing rows or columns
  - Aggregate data by grouping it
  - Combine data from different sources either by appending or merging tables

    - Transpose, pivot and unpivot values
    - Use the first row as headers
    - Split columns
    - Create new columns from examples based on conditions by duplicating, applying a function, using indexes, by transforming other columns, or with custom code
    - Set column data types
    - Replace values
    - Remove errors
    - Remove duplicates
- Apart from tables, queries can also return lists, scalar values, and other data structures. To avoid repeating the same code, you can leverage parameters that you can create yourself. You can also create custom functions either by using parameters or by writing your own M code.
- Power Query can translate some transformations into the native language of data source, resulting in improved performance. This is known as Query Folding, and you can check if it takes place by right-clicking on a step and viewing the native query.
- Every data source has its own privacy level, which can be one of the following: Private, Organizational, Public, and None. Each Private data source is completely isolated from all other data sources. Organizational data sources are visible to each other and are isolated from Public data sources. By default, Power BI combines data from different sources according to each; this behavior can be disabled in Power BI settings, though privacy is not guaranteed in this case.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X