Analyzing and Visualizing Data with Microsoft Power BI

Exam Ref 70-778

Daniil Maslyuk

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

Trademarks

Warning and Disclaimer

Special Sales

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

Quick access to online references

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these addresses (also

known as URLs) can be painstaking to type into a web browser, so we've compiled all of them into a single list that readers of the print edition can refer to while they read.

Download the list at *https://aka.ms/examref778/downloads*

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

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

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.

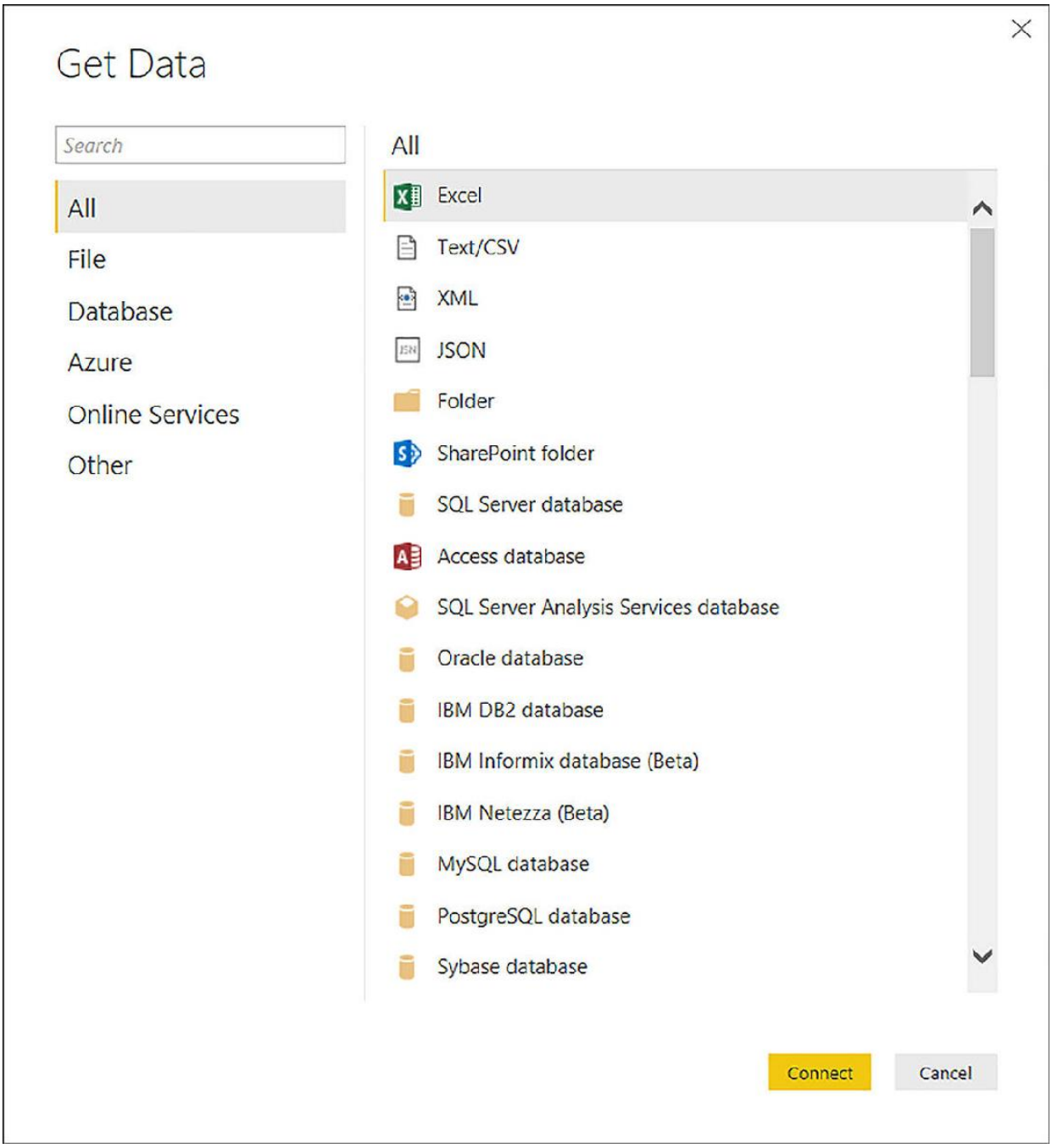


FIGURE 1.1 Get Data window

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.

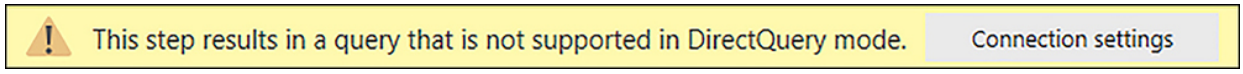


FIGURE 1.3 Unsupported by DirectQuery transformation error

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

Category	Import data	DirectQuery	Live Connection
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.

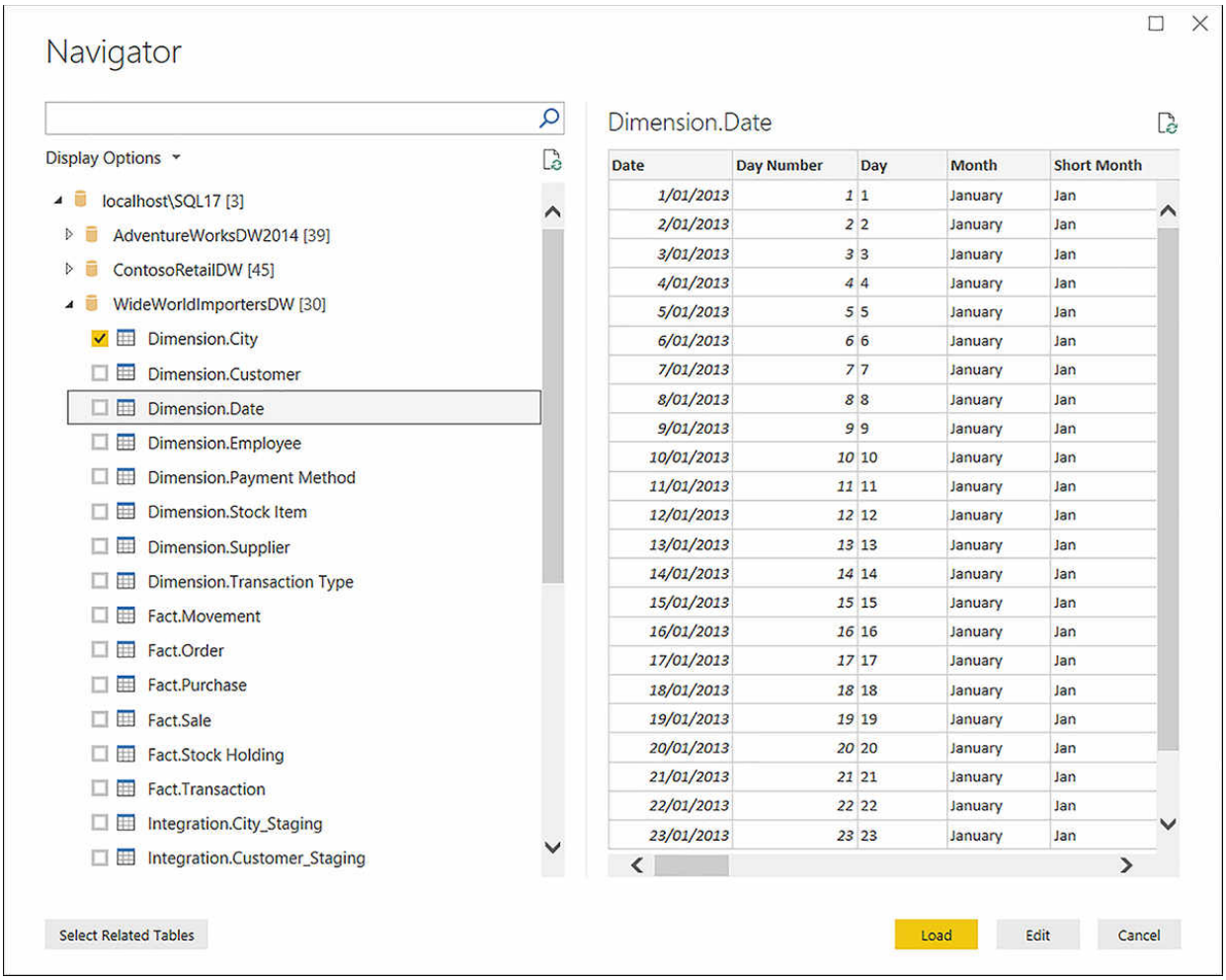


FIGURE 1.4 The Navigator window

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).

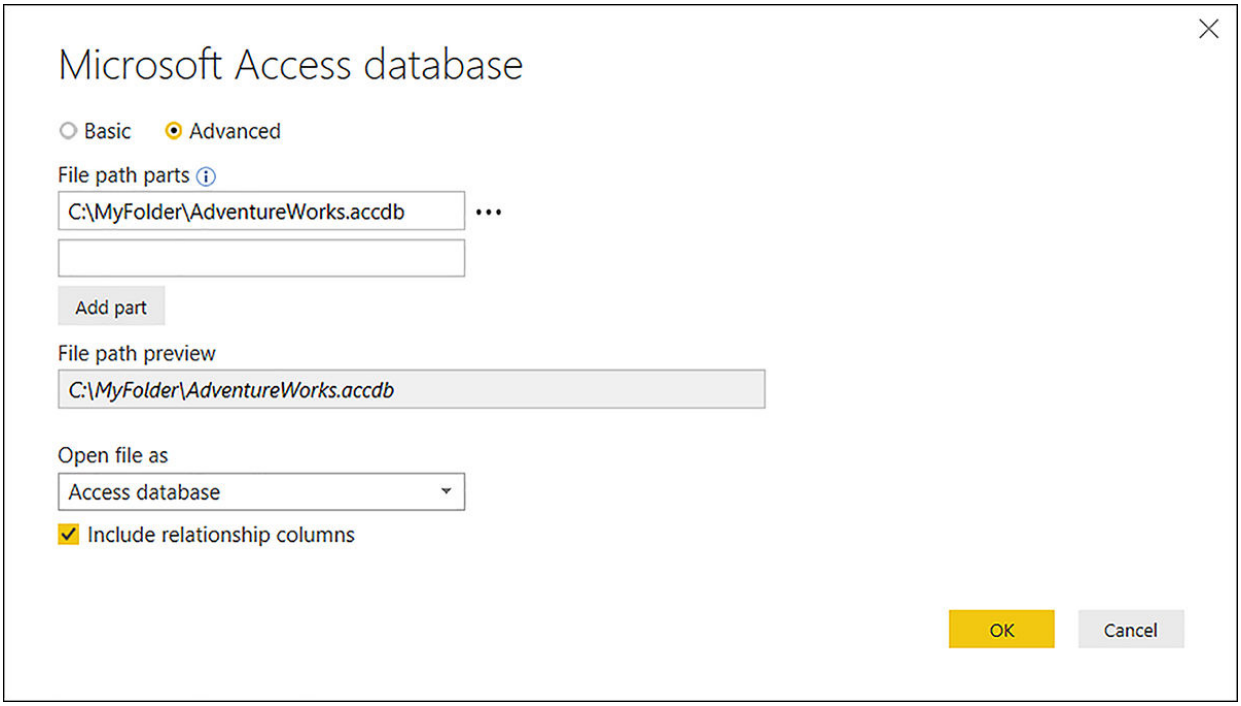


FIGURE 1.5 Advanced Access database connection settings

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**.

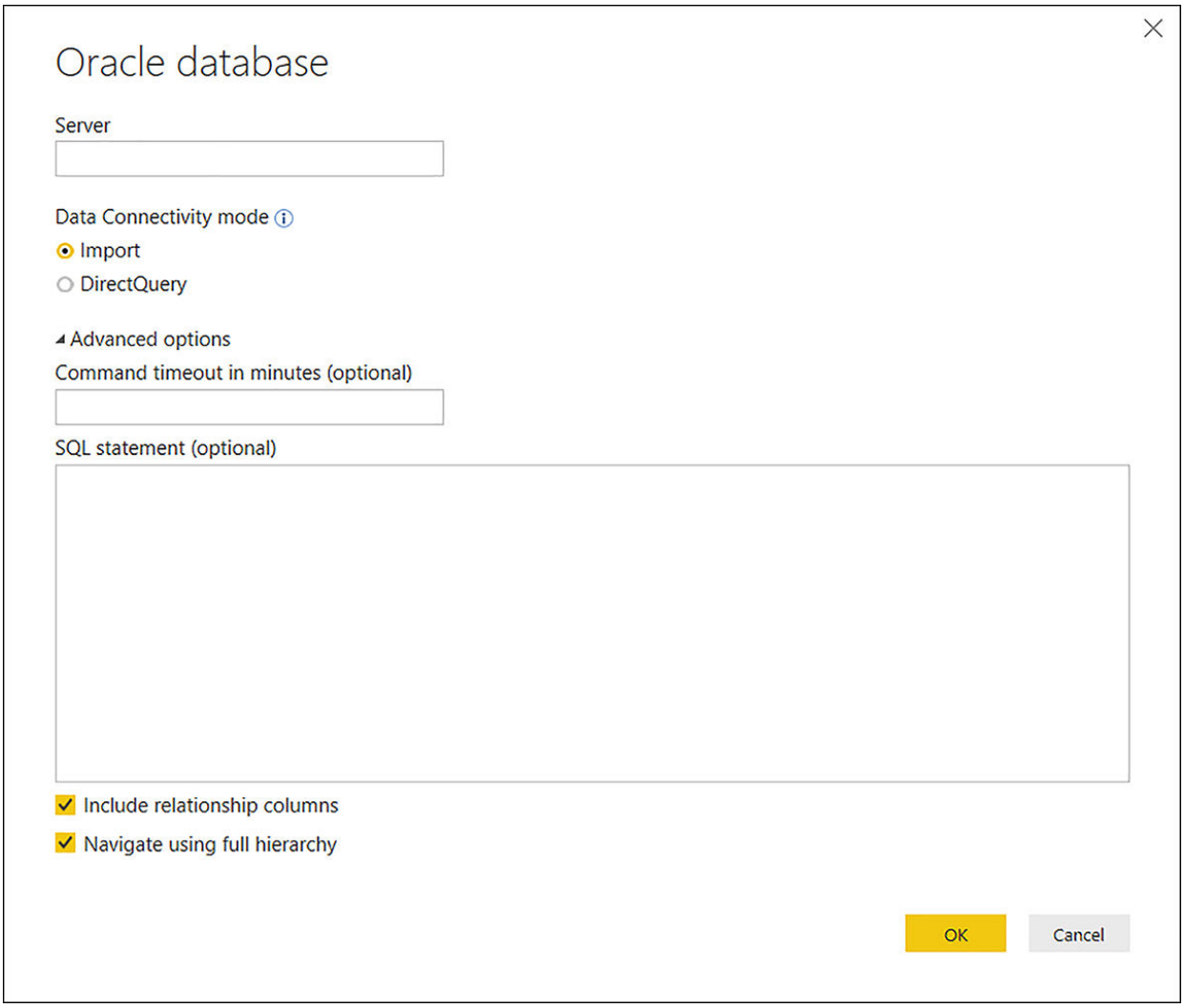


FIGURE 1.6 Oracle database connection settings window

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.

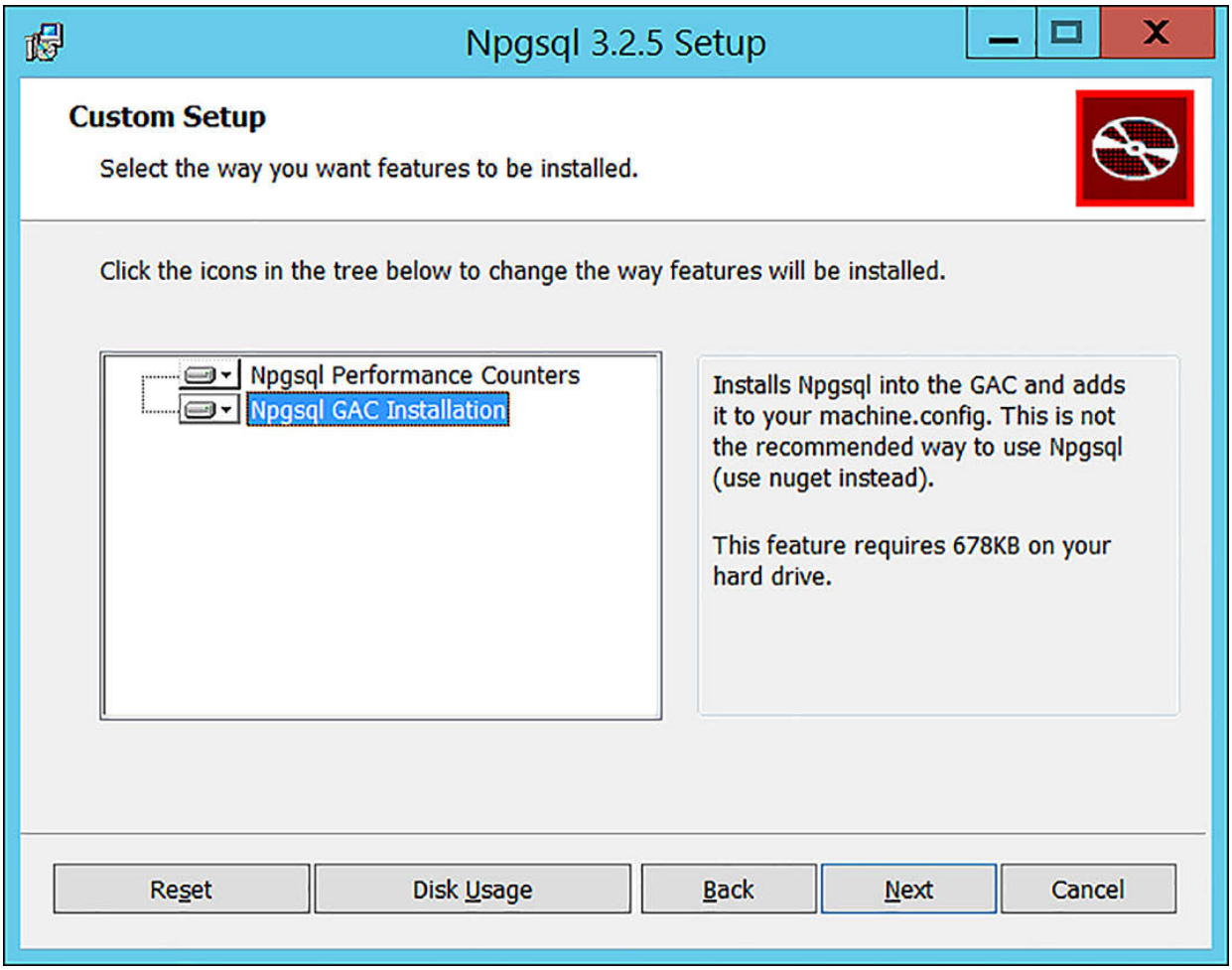


FIGURE 1.7 Npgsql setup features selection window.

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.

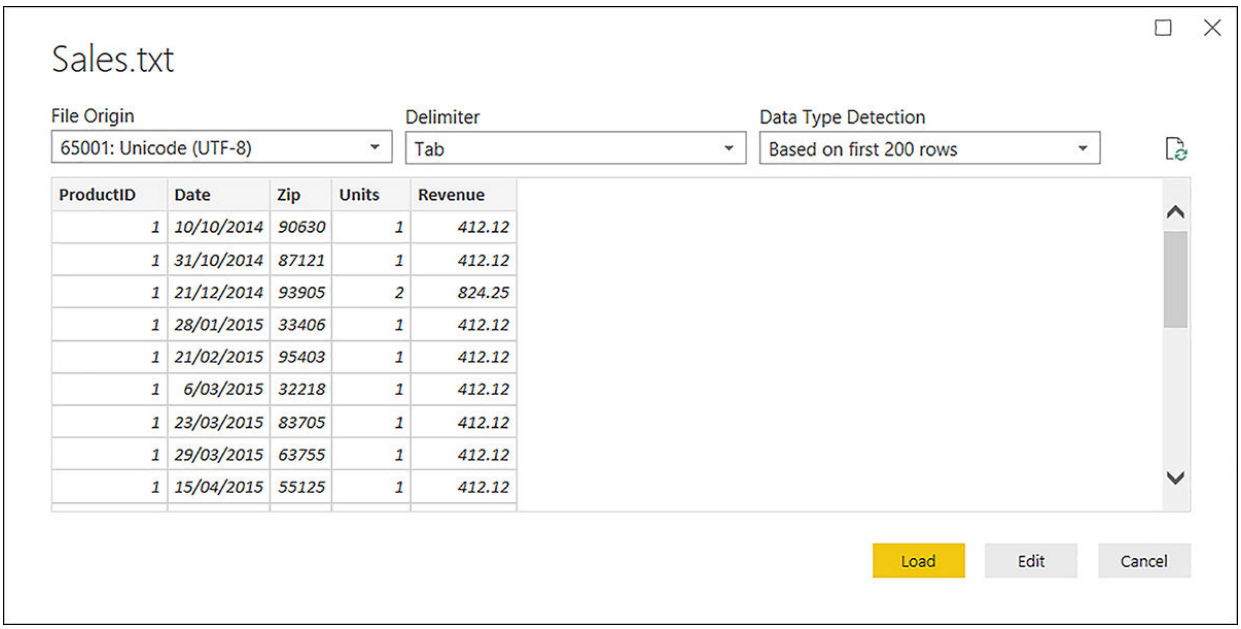


FIGURE 1.8 The Text/CSV file settings and data preview window

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.

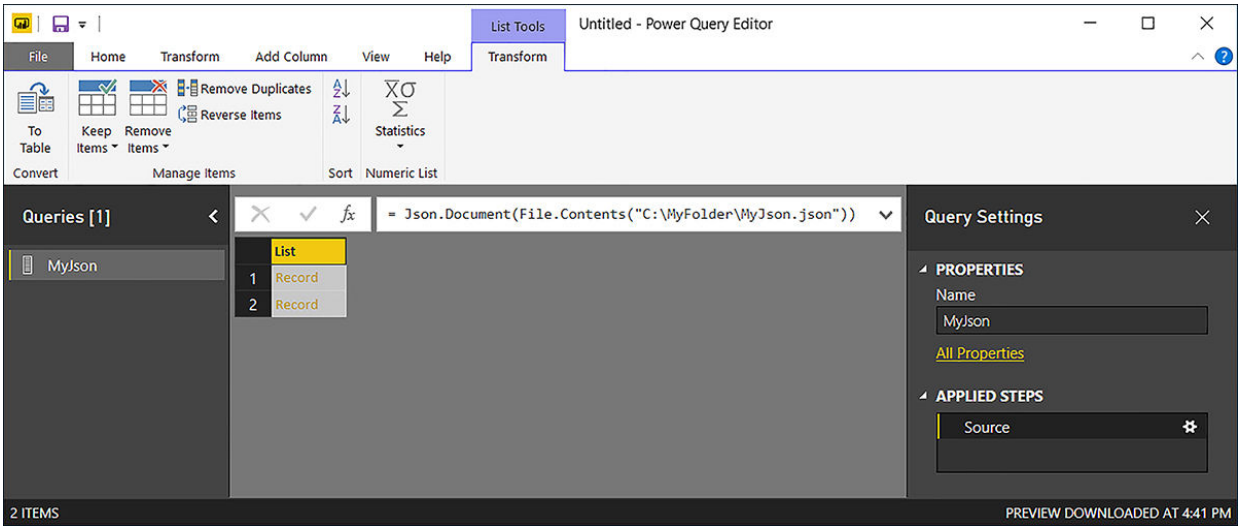


FIGURE 1.9 Power Query Editor after opening a JSON file

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.

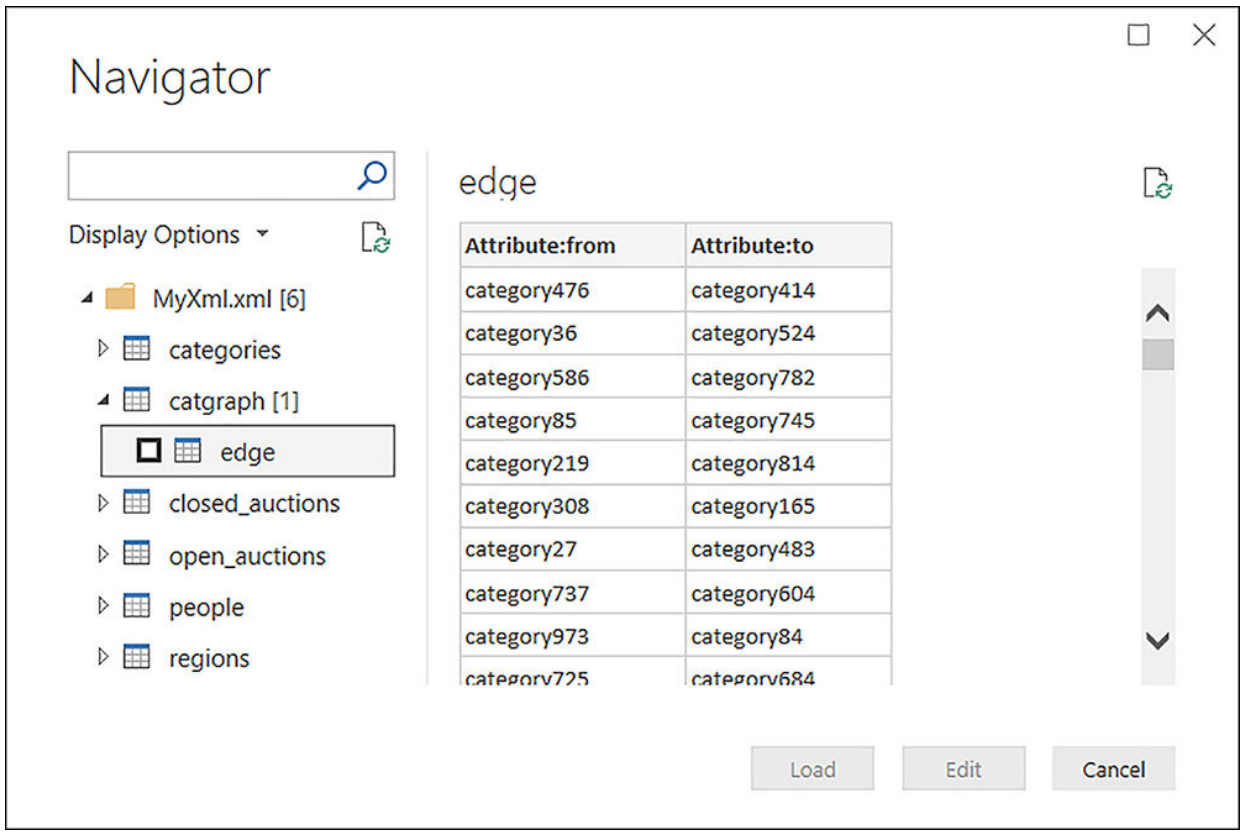


FIGURE 1.10 Contents of a sample XML file

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

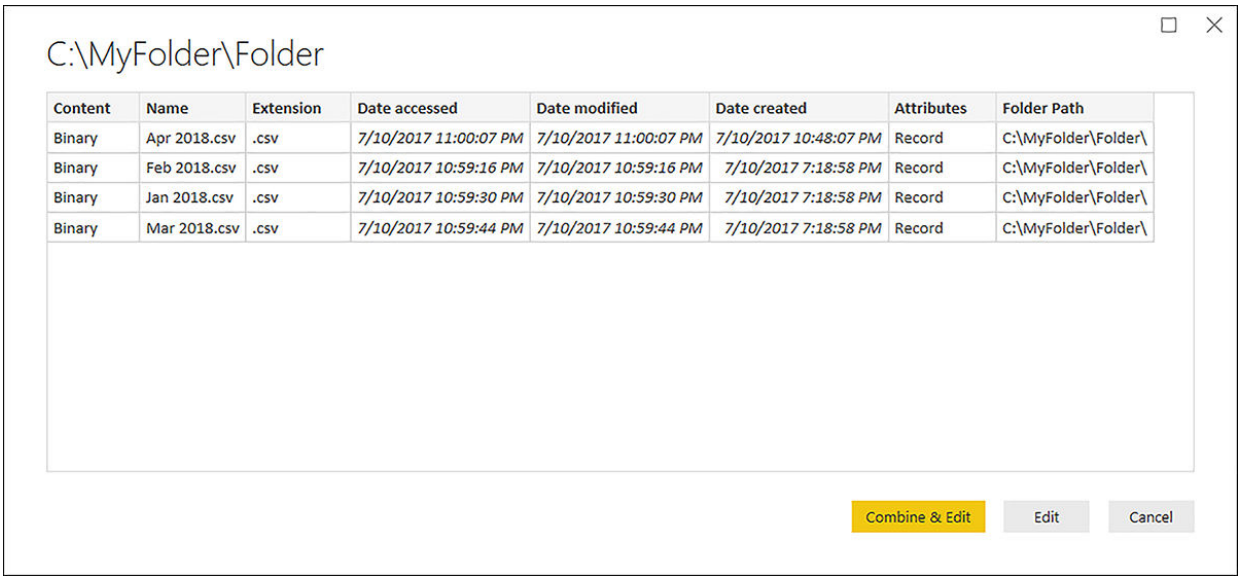


FIGURE 1.11 Folder preview window

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).

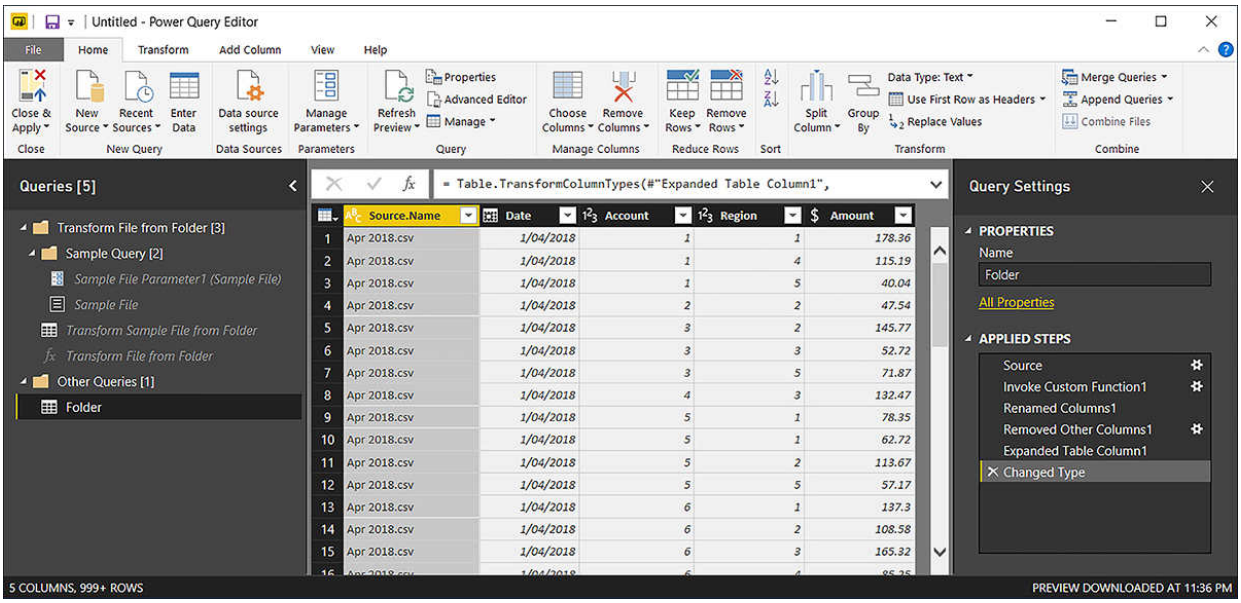


FIGURE 1.12 Objects created after combining files

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.

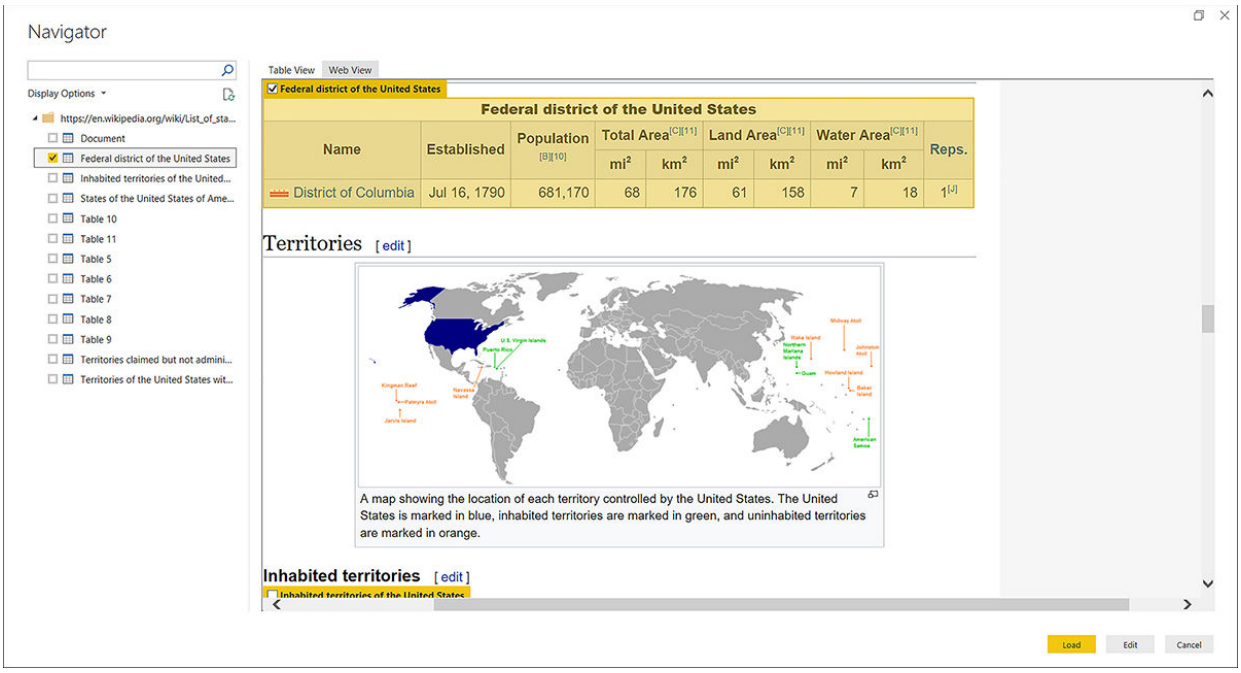


FIGURE 1.13 Navigator window when connecting to a web page

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).

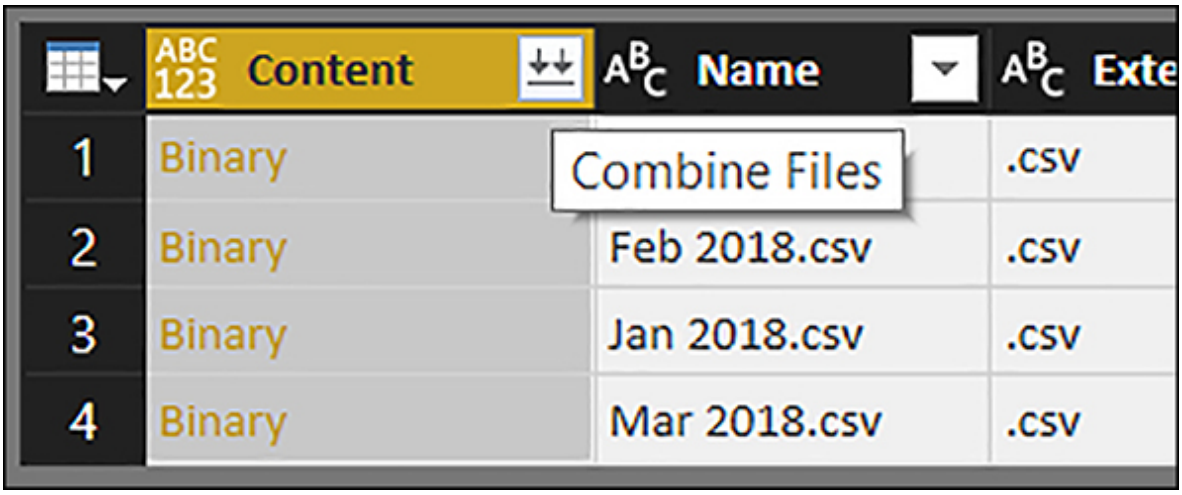


FIGURE 1.14 Combine Files button

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.

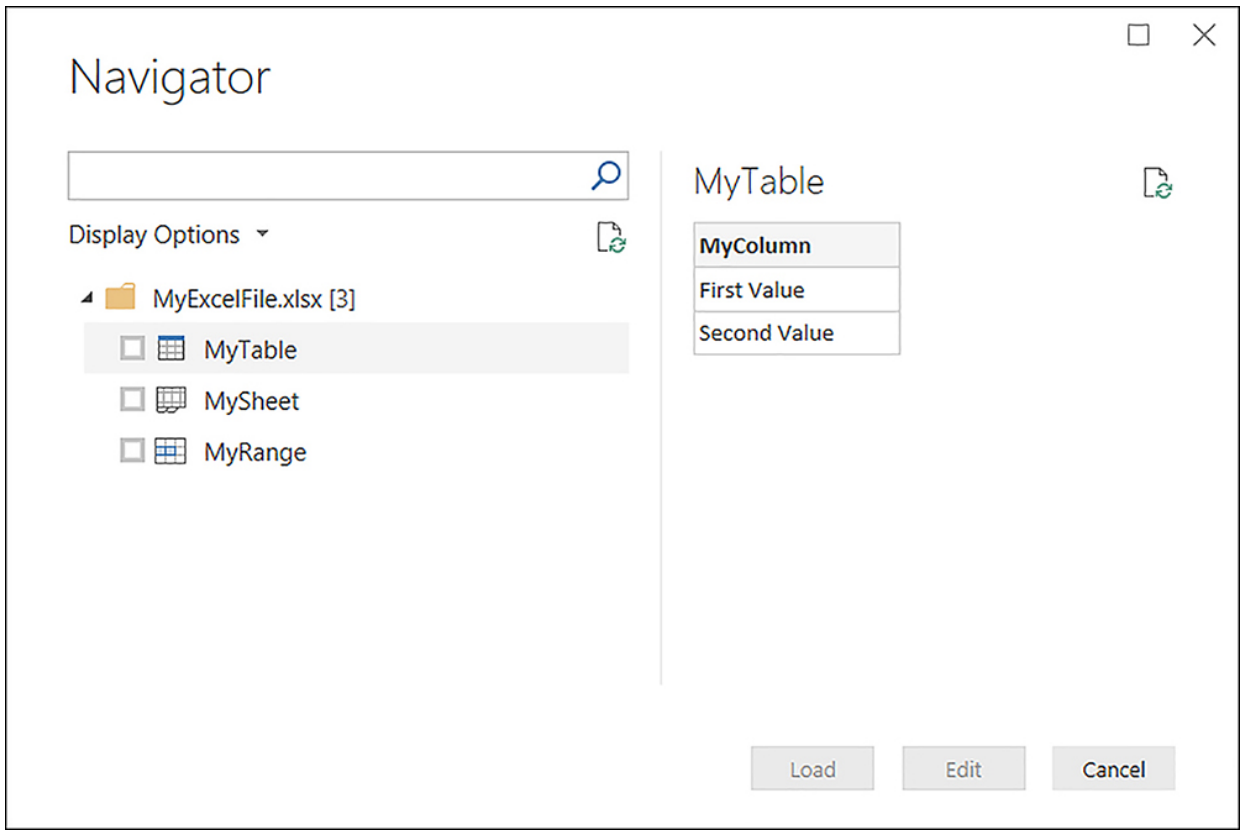


FIGURE 1.15 Navigator window when connecting to Excel

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).

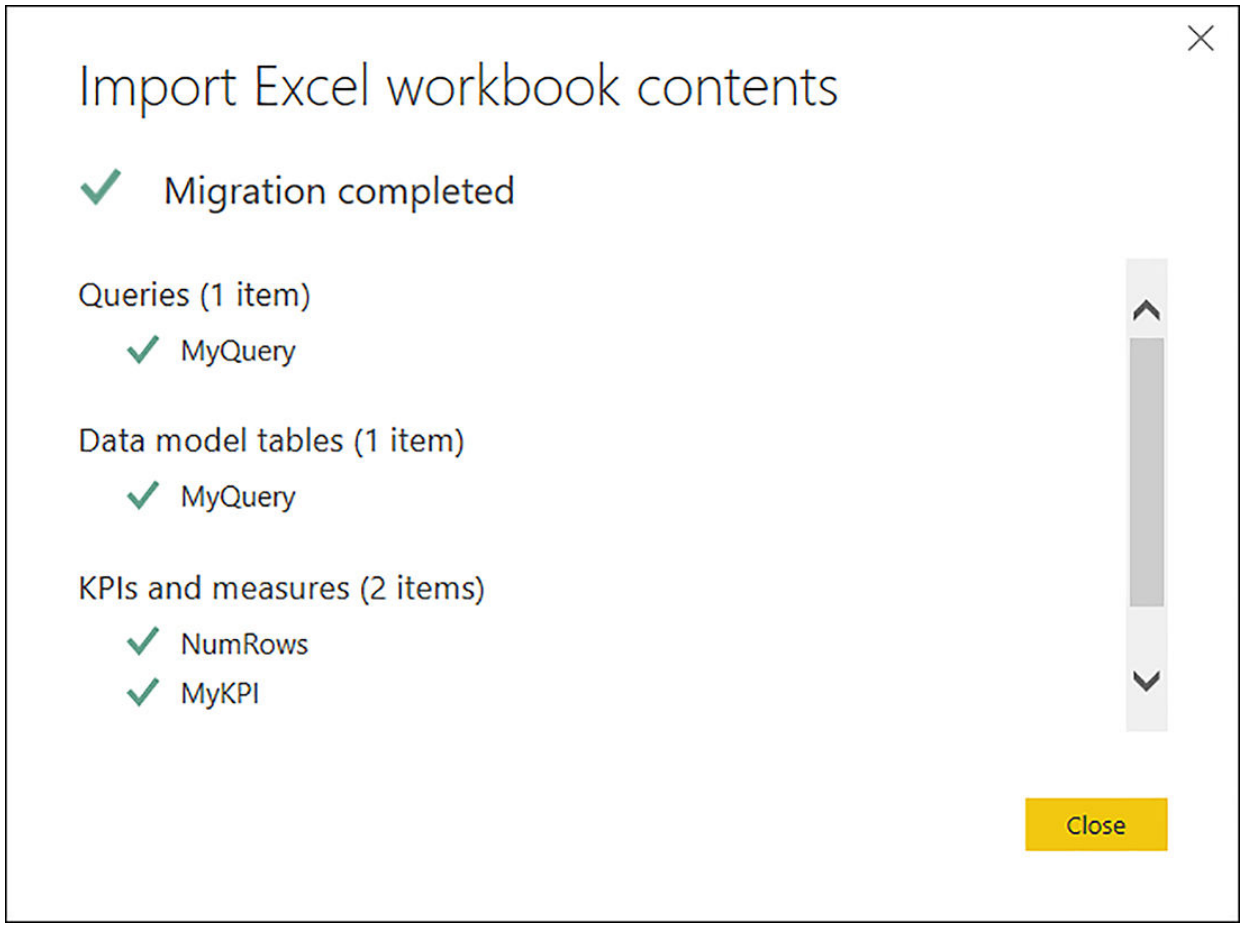


FIGURE 1.16 Import Excel workbook contents status window

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**.

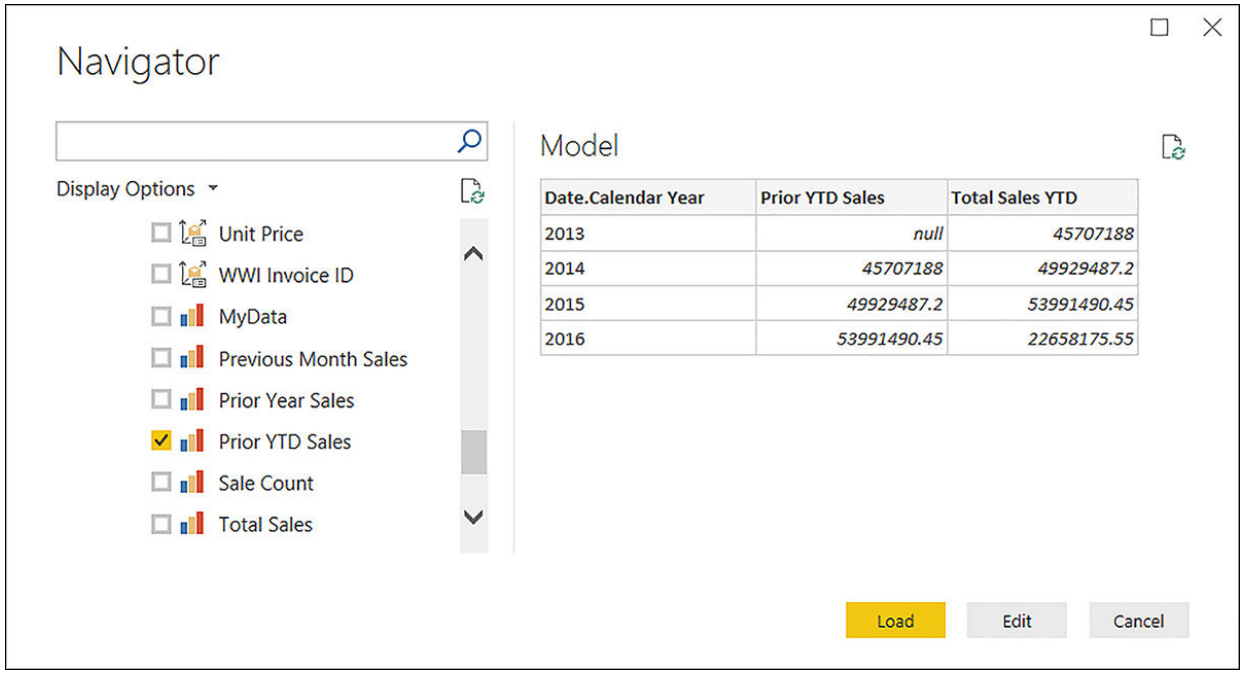


FIGURE 1.17 Navigator window after selecting Import

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.

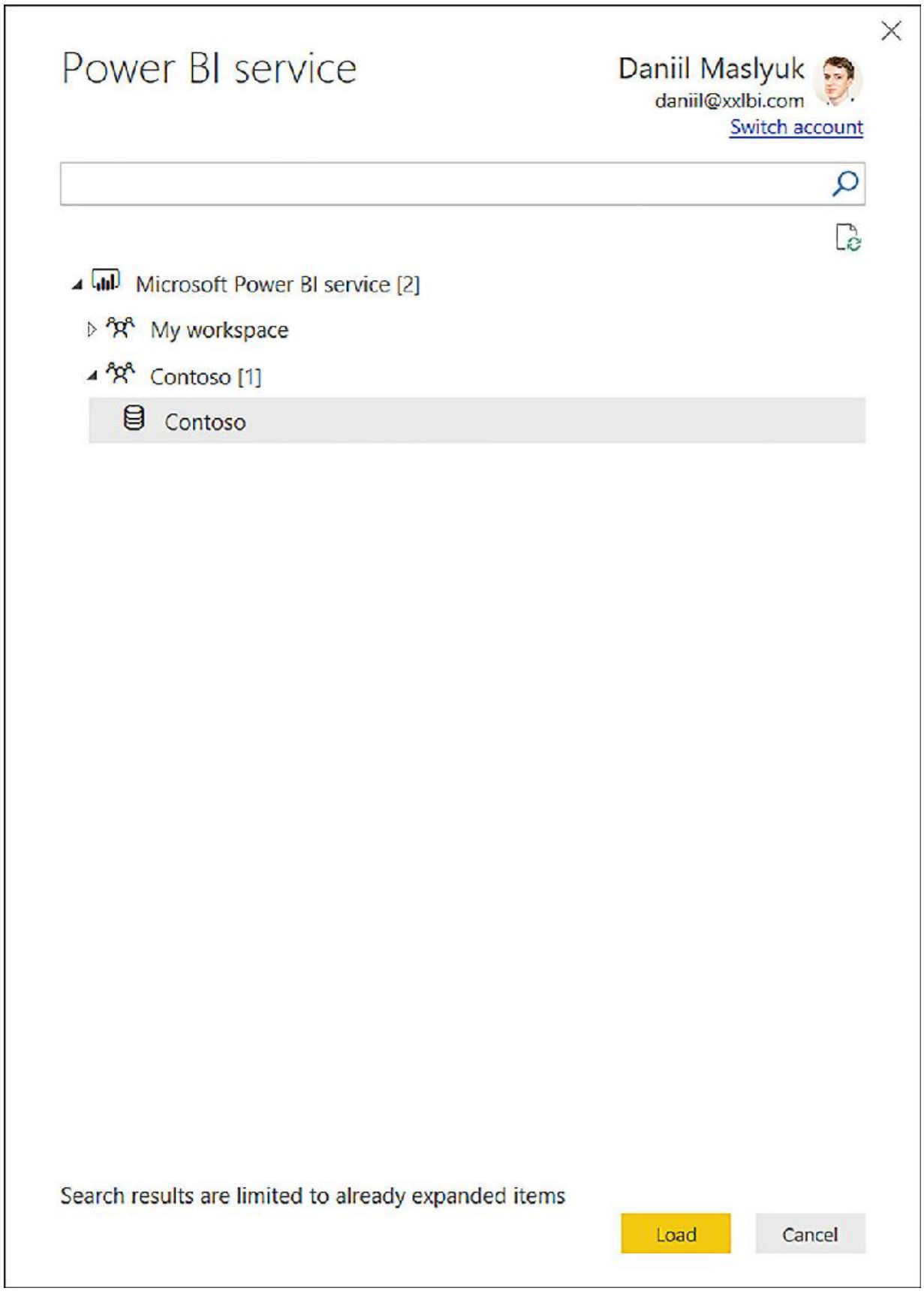

Power BI service
Daniil Maslyuk
daniil@xxlbi.com
Switch account
Microsoft Power BI service [2]
My workspace
Contoso [1]
Contoso
Search results are limited to already expanded items
Load
Cancel

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*.

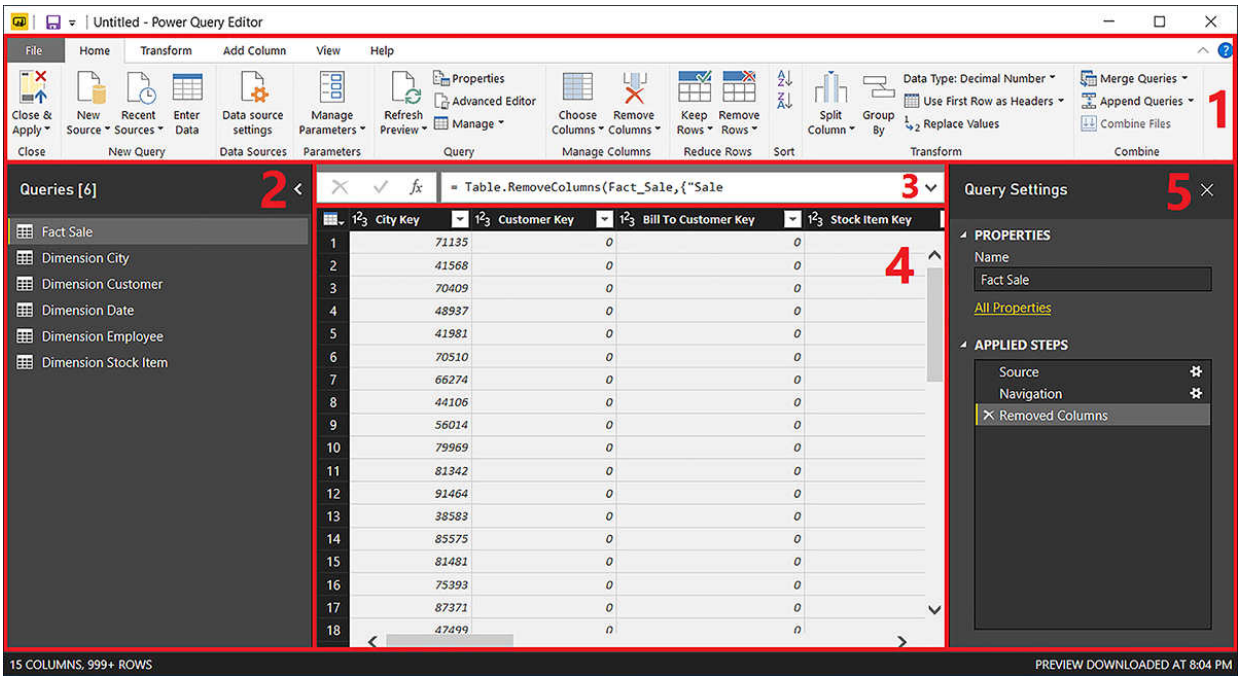


FIGURE 1.19 The Power Query Editor window

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.

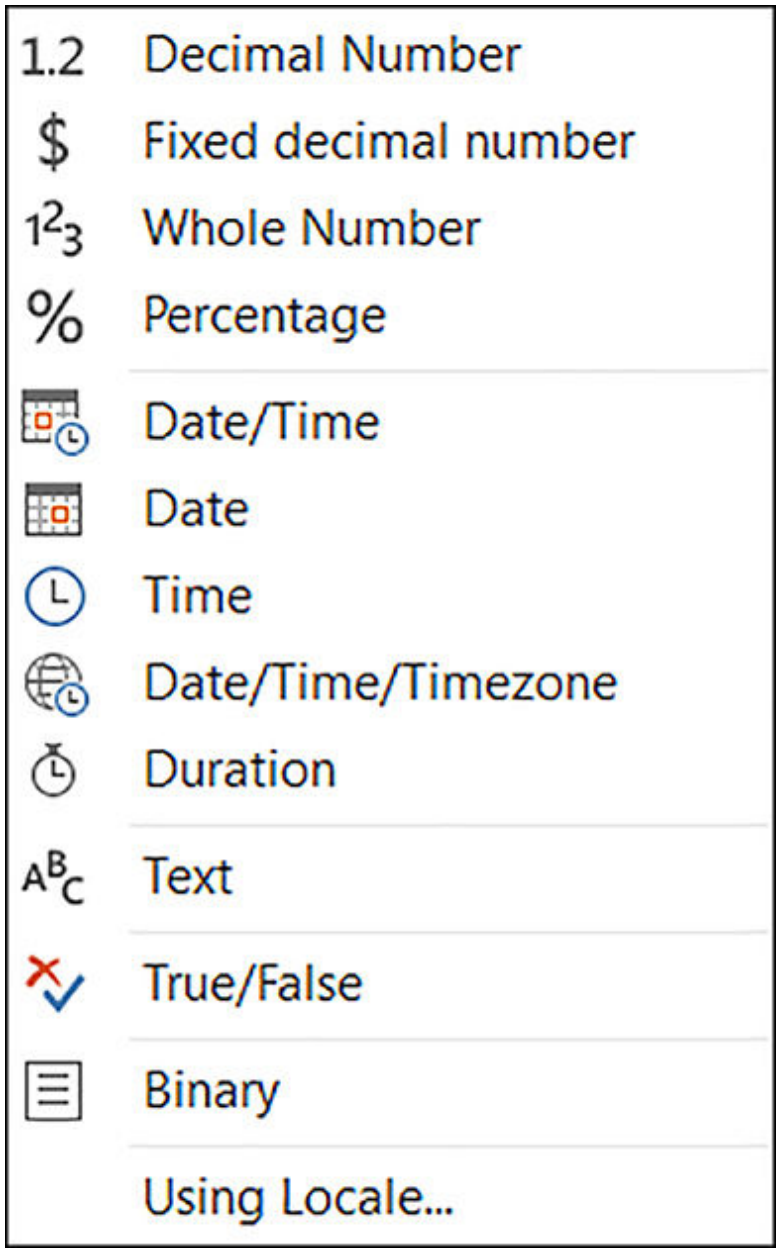


FIGURE 1.20 List of data types supported by Power Query

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:

Click here to view code image

```
= 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:

Click here to view code image

```
= 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.

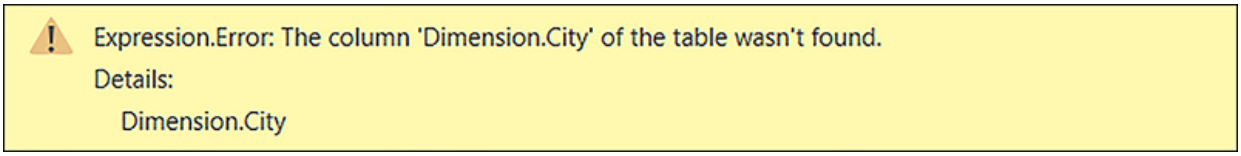


FIGURE 1.22 Error message after we excluded relationship columns

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:

Click here to view code image

```
= 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

Click here to view code image

```
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, WideWorldImportersDW, 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 SaleInitial query. Both queries can be seen in Listing 1-2.

LISTING 1-2 SaleInitial and Fact Sale queries

Click here to view code image

```
// 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

Click here to view code image

```
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

Click here to view code image

```
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.

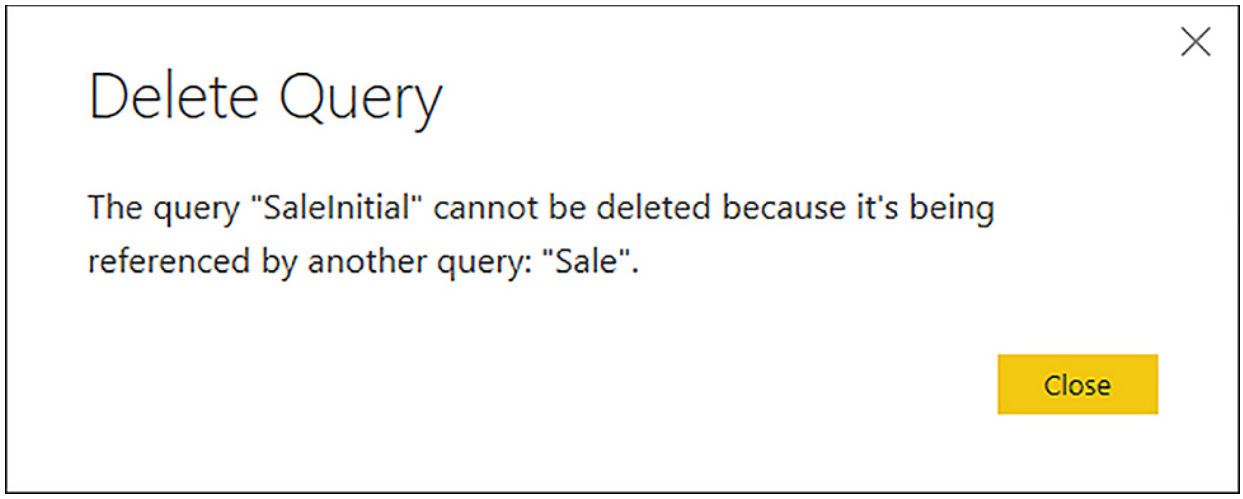


FIGURE 1.23 Error when deleting the SaleInitial query

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.

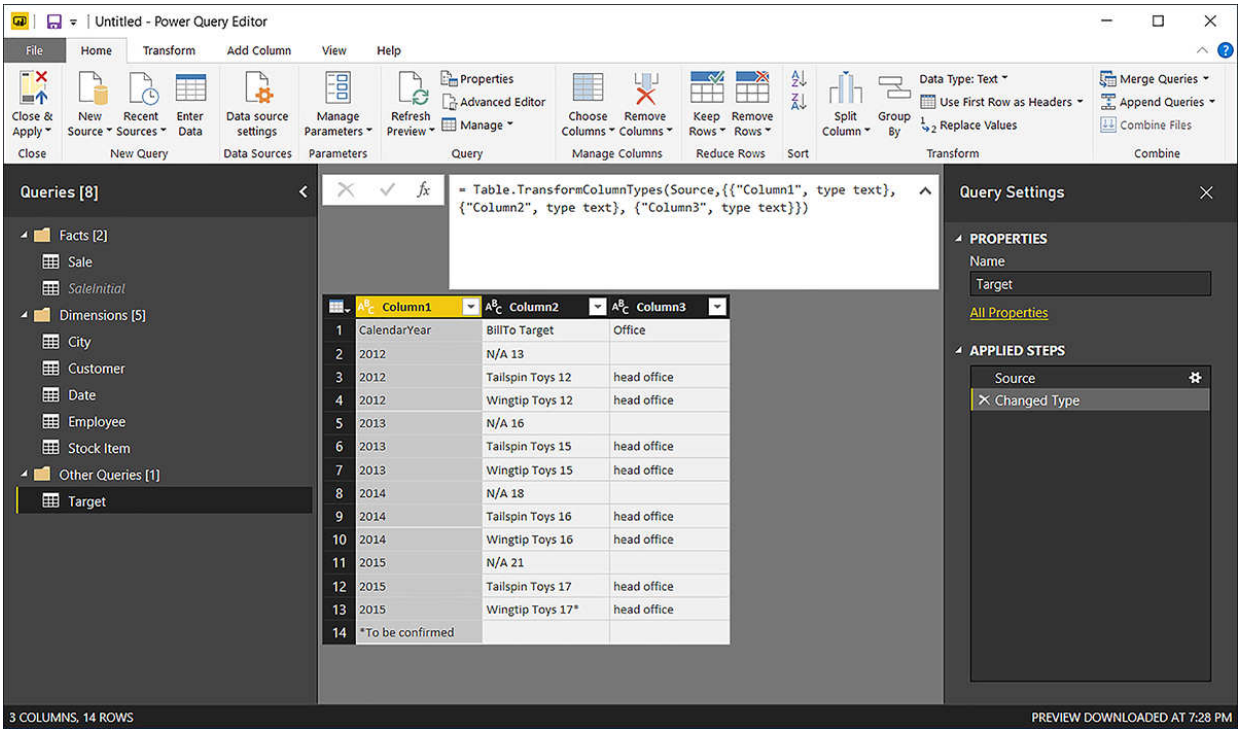


FIGURE 1.25 Power Query Editor after connecting to Target.txt

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

Click here to view code image

```
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.

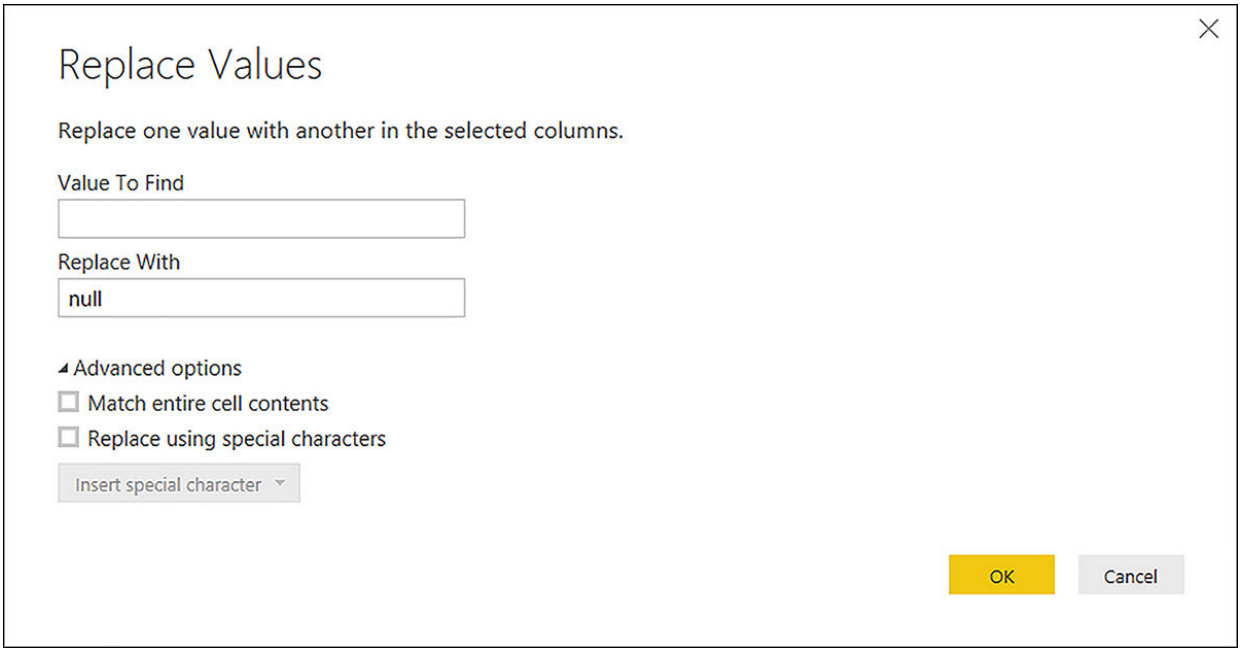


FIGURE 1.26 Replace Values window

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

Click here to view code image

```
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.

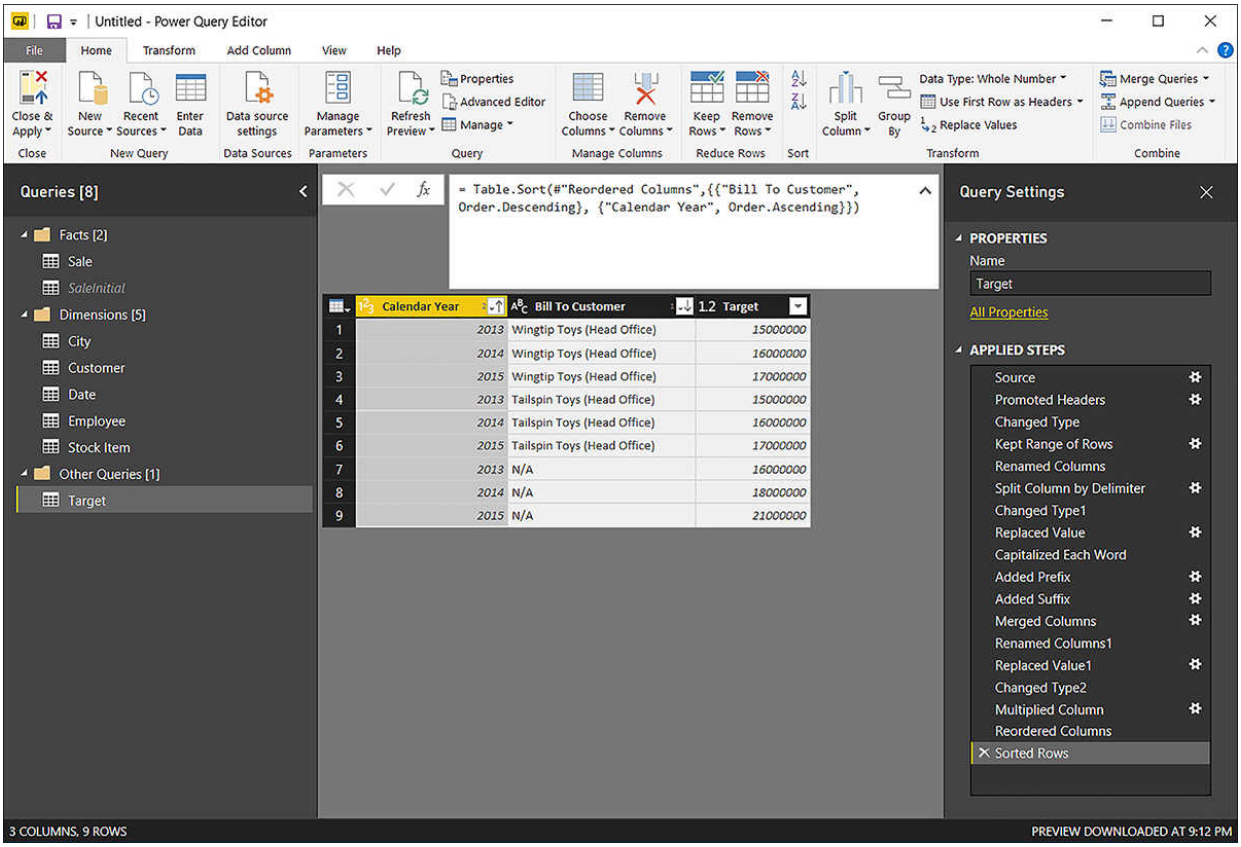


FIGURE 1.27 Power Query Editor after applying all transformations to Target.txt

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.

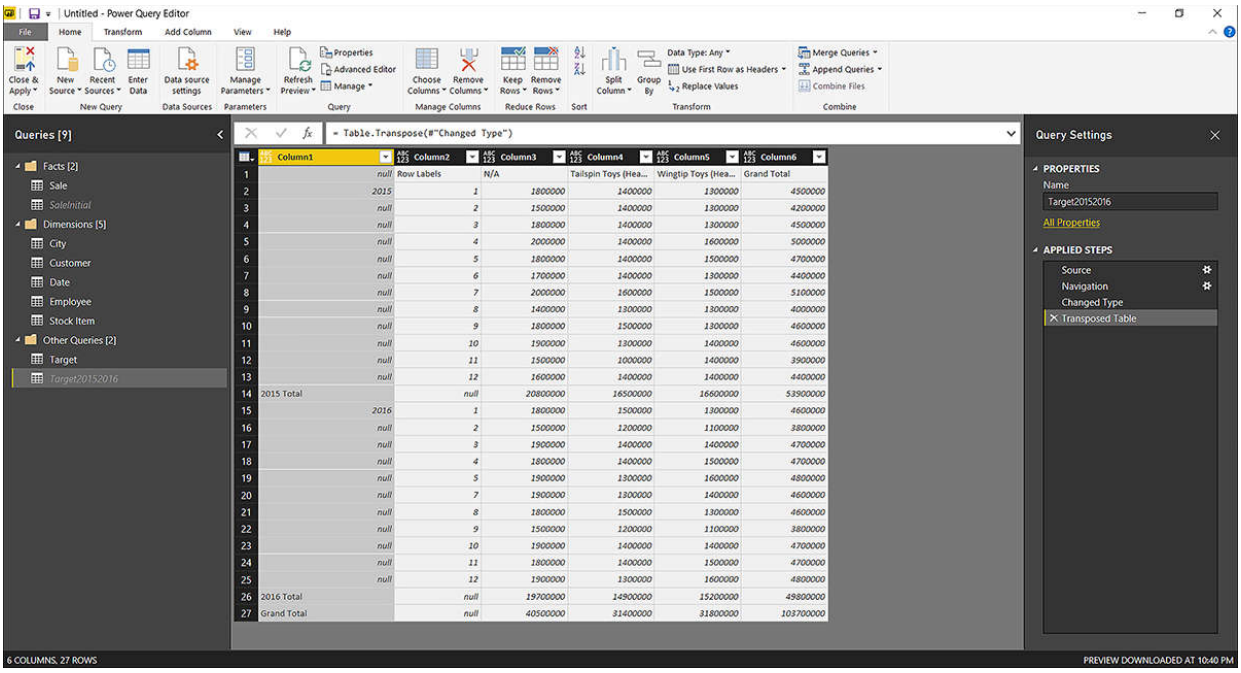


FIGURE 1.28 Power Query Editor after transposing table

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

Click here to view code image

```
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**.

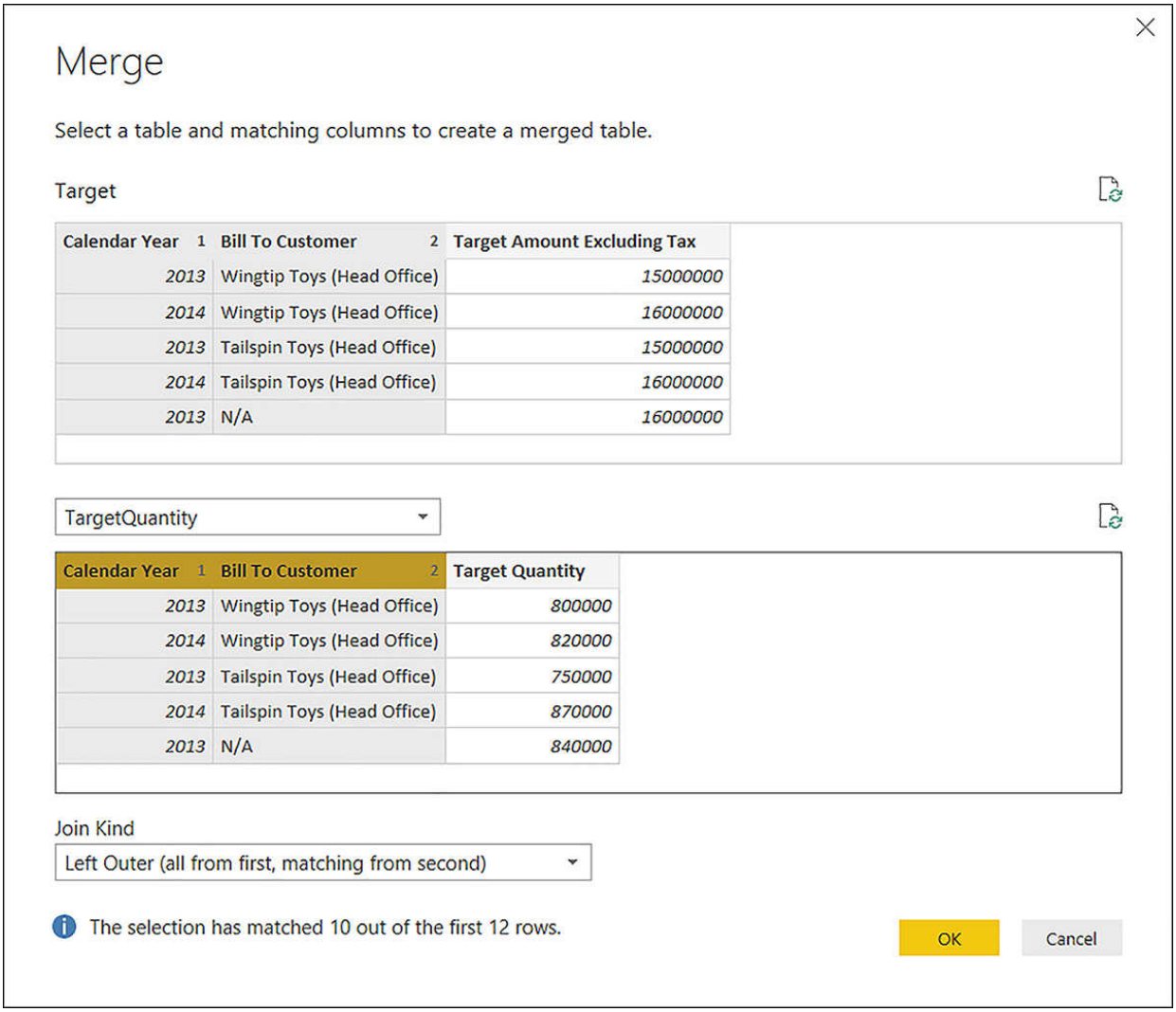


FIGURE 1.30 Merge dialog box

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.

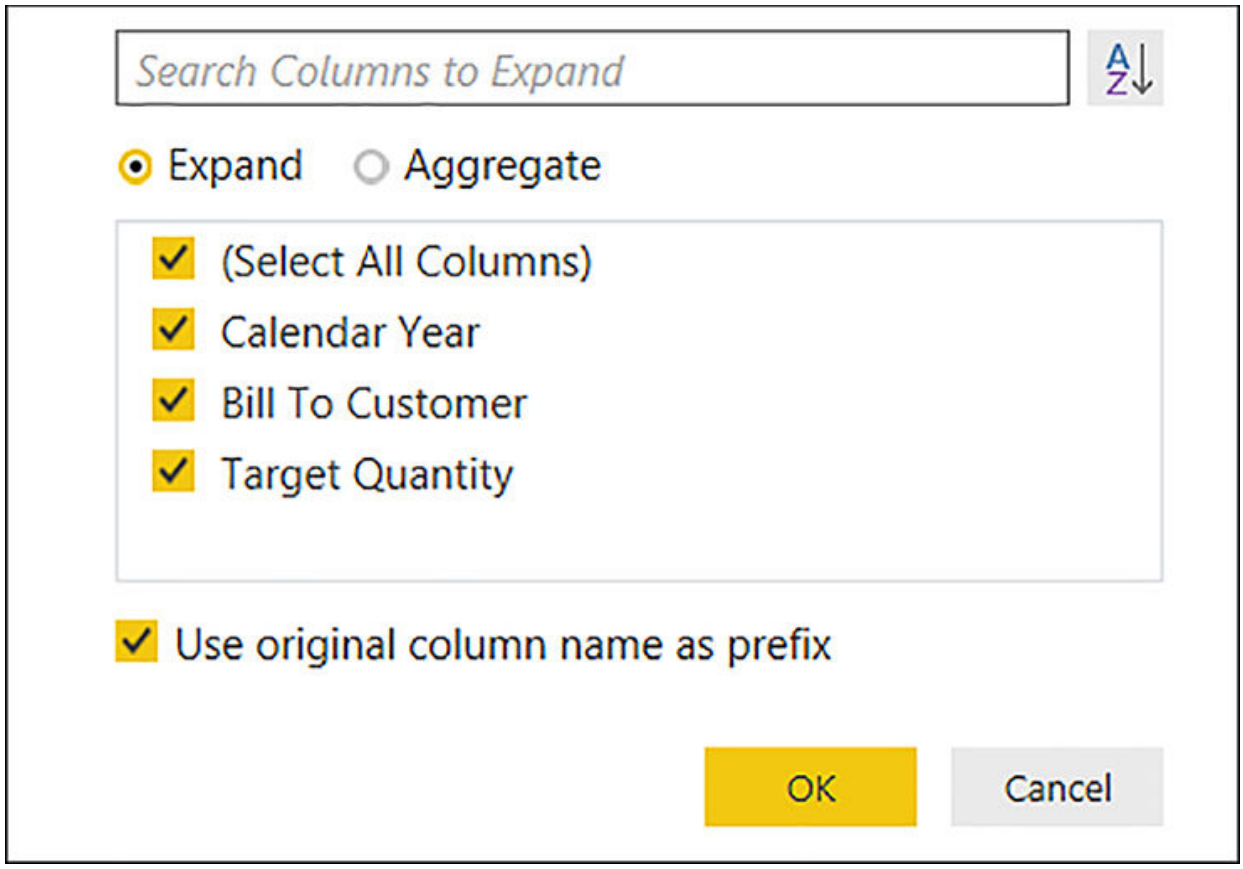


FIGURE 1.31 Table expansion options

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

Click here to view code image

```
(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.

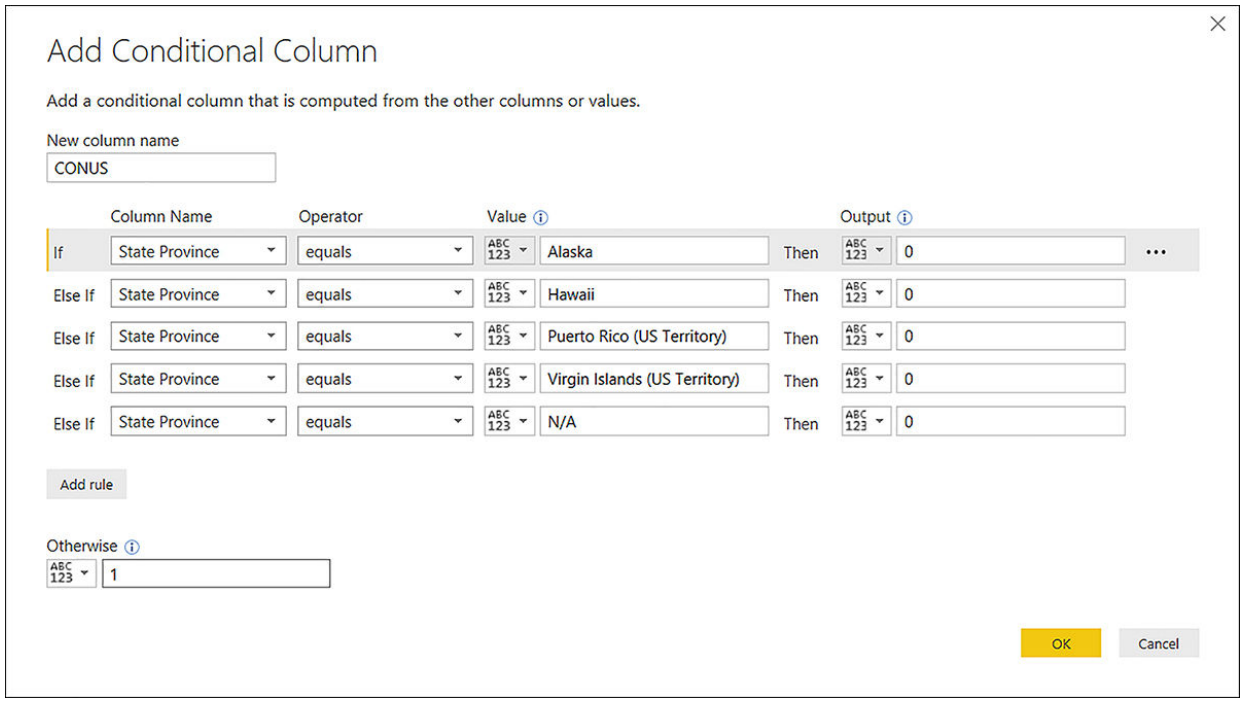


FIGURE 1.32 Add Conditional Column window

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.

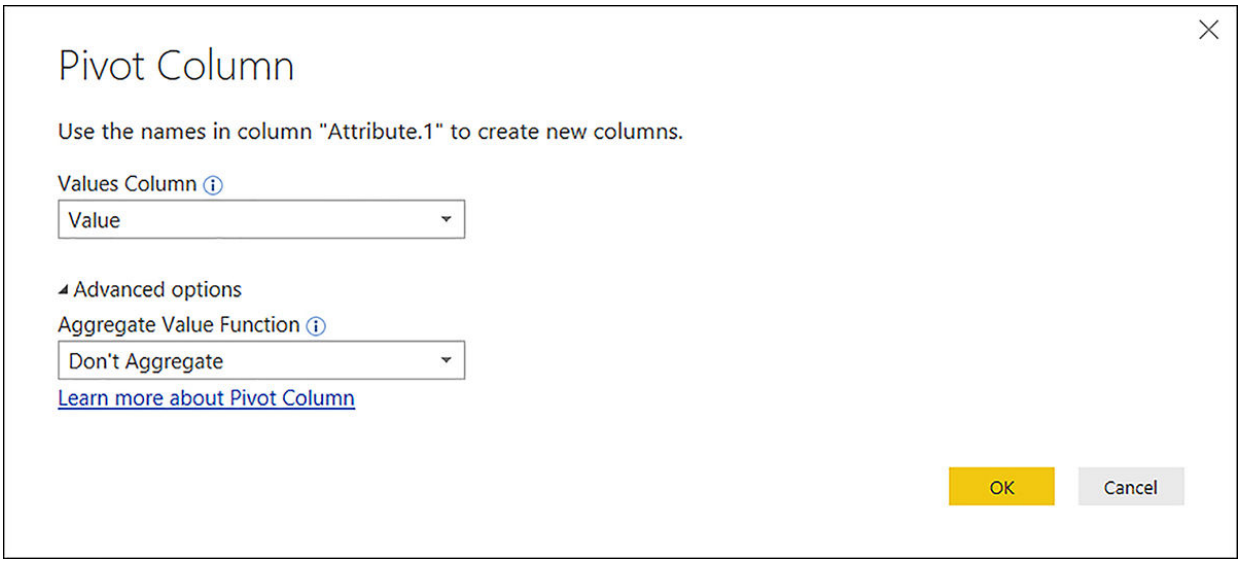


FIGURE 1.33 Pivot Column dialog box

> ***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

Click here to view code image

```
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

Click here to view code image

```
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

Click here to view code image

```
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

Click here to view code image

```
// 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.

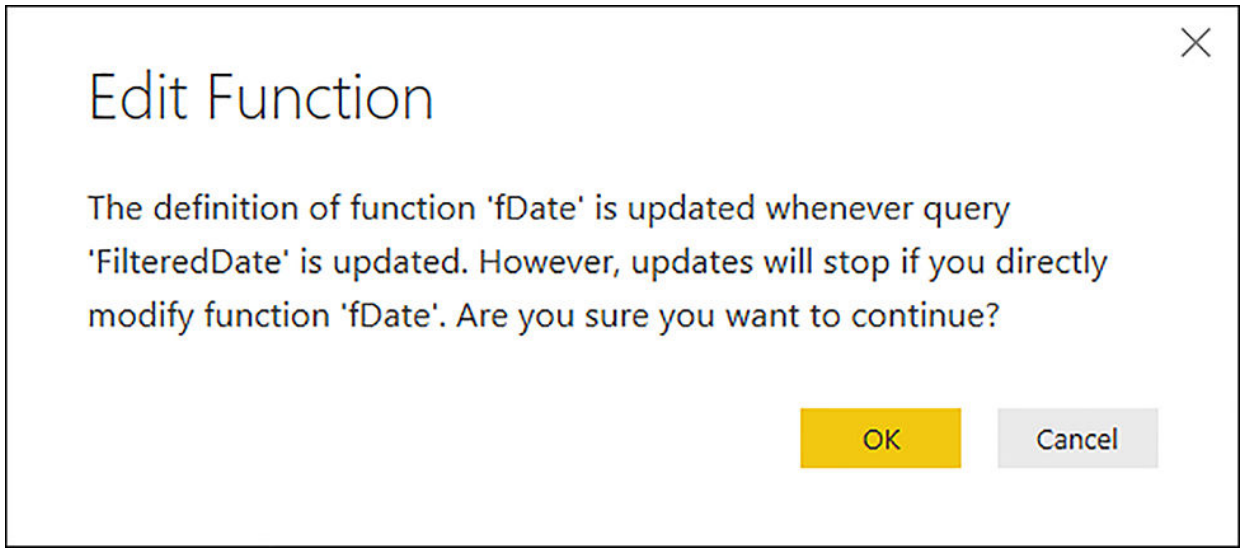


FIGURE 1.34 Edit Function dialog box

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

Click here to view code image

```
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.

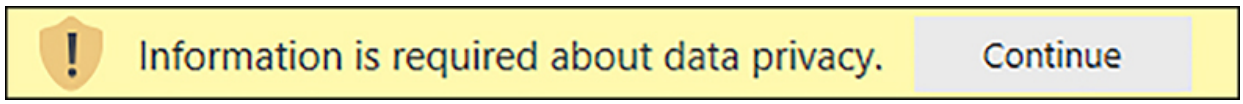


FIGURE 1.35 Data privacy prompt

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.

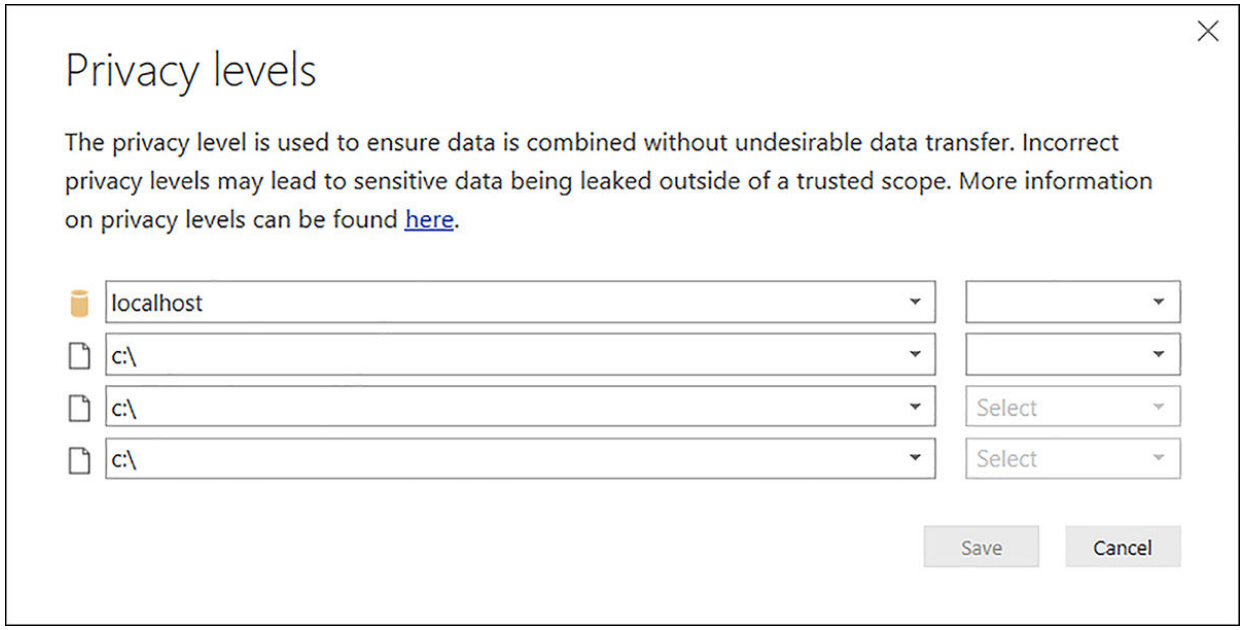


FIGURE 1.36 Privacy Levels dialog box

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

Click here to view code image

```
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

Click here to view code image

```
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

Click here to view code image

```
=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.

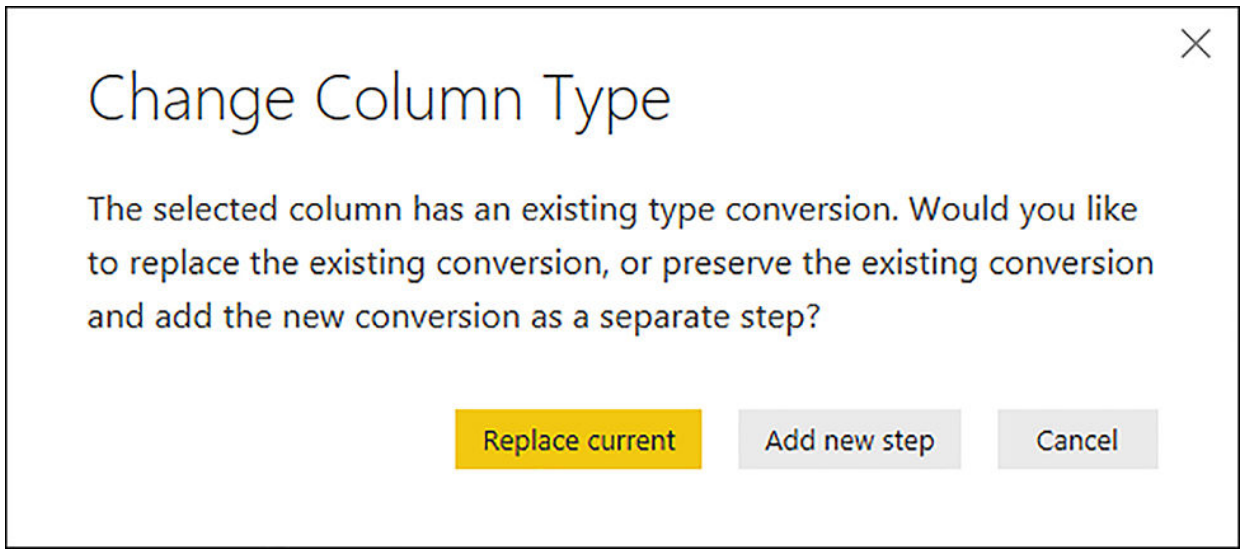


FIGURE 1.37 Change Column Type dialog box

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

Click here to view code image

```
=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

Click here to view code image

```
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 Desktop 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.

CHAPTER 2
Modeling and visualizing data

The previous chapter reviewed the skills necessary to connect to data sources and transform data using Power Query Editor—the process also known as data shaping.

This chapter starts by covering data modeling skills. In Power BI, a data model is a collection of one or more tables connected by relationships. Apart from the M language, which is used for data shaping, Power BI uses DAX, which is its native formula and query language. Power BI possesses rich data modeling capabilities, which include creating relationships as well as enriching a data model with hierarchies, measures, calculated columns, and calculated tables.

Next, we review the data visualization skills: working with different types of charts and shapes, managing interactions between visuals, and configuring pages layout. We also review custom reporting solutions: configuring and accessing Power BI Embedded, using REST API, as well as working with custom visualizations.

The skills required to work with Power BI service will be covered in Chapter 3, "Configure dashboards, reports, and apps in the Power BI Service."

Skills in this chapter:

Skill 2.1: Create and optimize data models

Before you can create any visuals, you need to create your data model by loading data and creating relationships. So far in Chapter 1, we have only created queries and performed transformations. In this section, we start with loading data, and creating relationships between tables and optimizing the data model for reporting.

This section covers how to:

- Manage relationships
- Optimize models for reporting
- Manually type in data
- Use Power Query

Manage relationships

in Chapter 1, we mostly worked in Power Query Editor. In this chapter, we work in the main Power BI Desktop window, which can be seen in Figure 2.1.

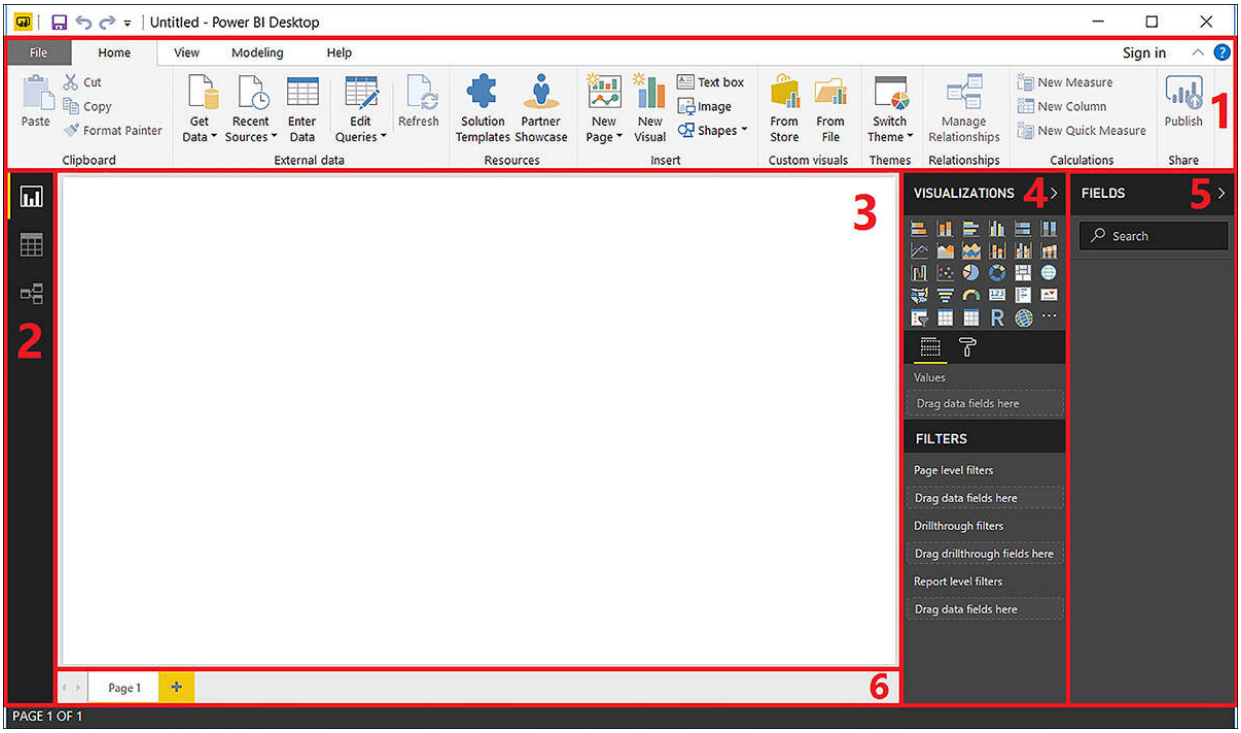


FIGURE 2.1 The main Power BI Desktop window

This window can be divided into six parts, each labeled with a number:

1. **Ribbons pane** The Home, View, Modeling, and Help ribbons can be seen by default. The File button is to the left of the Home ribbon. If needed, the Ribbons pane can be collapsed by double-clicking on the name of the active ribbon.
2. **View buttons** By default, you can change the view between Report (top button), Data (middle button), and Relationships (bottom button). The Data view will be unavailable if you choose the DirectQuery data connectivity mode; both the Data and Relationship views will be unavailable if you consume data with Live Connection.
3. **Report canvas** All visuals that you use will be placed on the canvas.
4. **Visualizations pane** By default, this pane contains the standard visuals that you can use. Below the visuals, you can choose between the Fields and Format tabs. The Fields tab contains field wells and should not be confused with the Fields pane described next. In the Format tab, you can format the report canvas or visuals and shapes when you select them. This pane can be collapsed by clicking on the arrow at the top of it.
5. **Fields pane** All tables that you import will be shown here, with each column being a field under each table. You can search for fields by typing in the Search area. The Fields pane can be collapsed in the same manner as the Visualizations pane.
6. **Page tabs** You can rename, duplicate, delete, and create new pages here.

We will be reviewing the interface, including the options not seen by default, in more detail in this chapter. As Power BI Desktop is updated every month, your interface might differ from Figure 2.1.

In this chapter, we are going to continue using the Wide World Importers data in addition to the data we added from text and Excel files in the previous chapter. If you followed along the examples in Chapter 1, you used your Power BI Desktop file; alternatively, you can use the CH02-Start.pbix file from this book's companion files. This file only contains the queries that are needed; extra queries used for review purposes have been removed. When you open the file, you will see the Apply Changes button (Figure 2.2).

You will also see this button every time you make changes to your queries without applying them.

FIGURE 2.2 Apply Changes button

If you use CH02-Start.pbix, you might not see the Apply Changes button immediately, and you might need to change some data source settings so that you can load data from your data sources. This file relies on the database being hosted at localhost, and companion files being at C:\Companion\Chapter 1. If any of this is different, clicking on the Apply Changes button will result in either partial load of data or an error. To point queries to the correct addresses, you can select **Home** > **External data** > **Edit Queries** > **Data source settings**. In the Data Source Settings window, you will see four data sources:

- Target.txt
- Target20152016.xlsx
- TargetQuantity.txt
- WideWorldImportersDW database

To point a data source to the correct address, you will need to click on the Change Source button for each source. In case of files, you will then need to click Browse and point to the relevant file in the Open window. In case of the database, you will need to specify the correct server and database names, as well as the credentials. After this, you might need to close and re-open the file to see the Apply Changes button.

If you do not have access to a SQL Server, you can still follow along with the examples in this chapter using CH02-Loaded.pbix file that has all data loaded, though you will not be able to refresh the data.

Relationships

In Power BI, relationships are often required to produce the right values when you have a data model with several tables. If you import your tables from a database where foreign keys are defined, as in Wide World Importers case, then Power BI imports the relationships as well, but only if you import all tables at once. In any case, you can manually create relationships.

After you click Apply Changes, data is loaded into the data model, and you can see all of the tables and columns appear in the Fields pane on the right. You can now switch to the Relationships view (Figure 2.3) to review the imported relationships. The Relationships view displays the data model diagram.

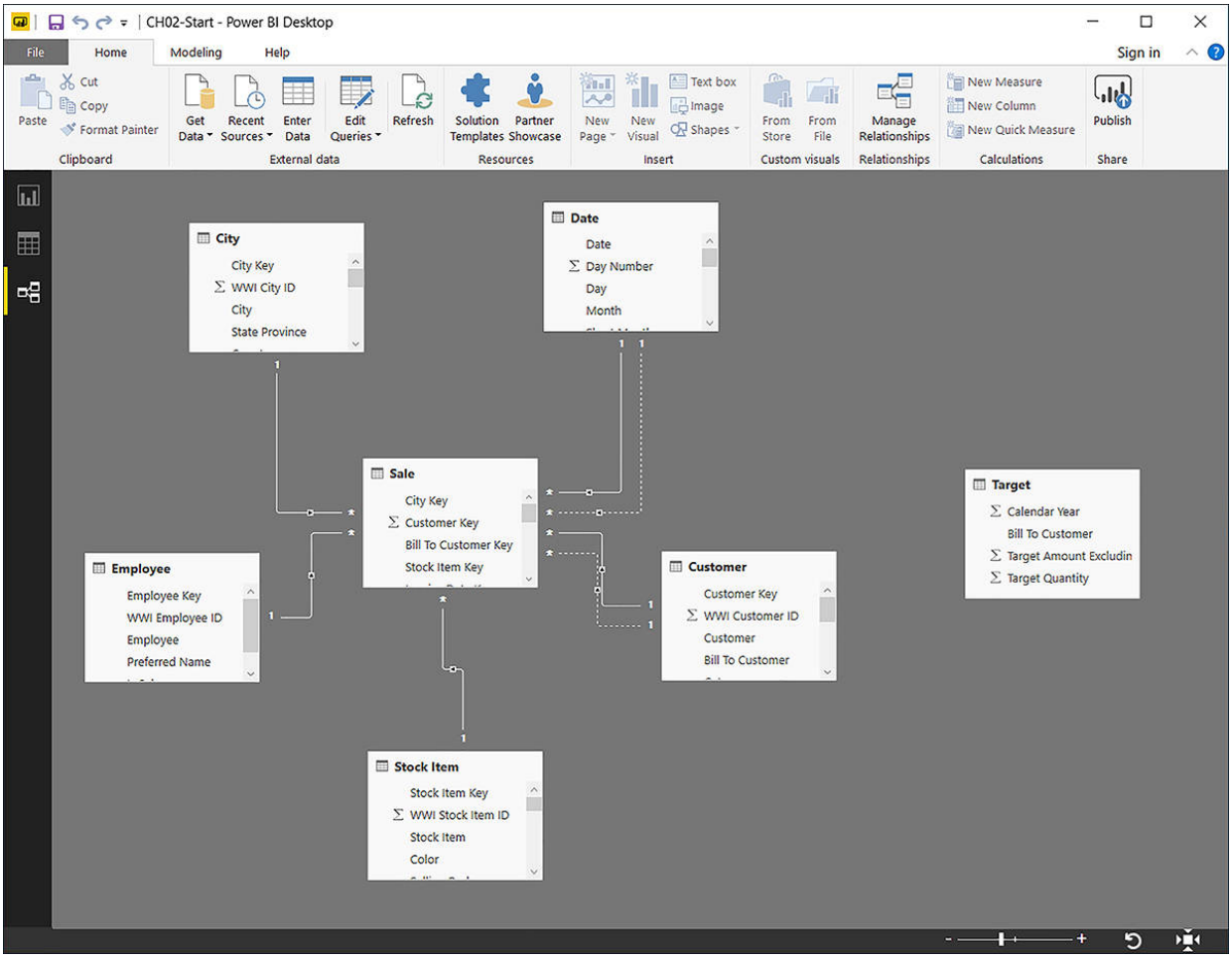


FIGURE 2.3 Relationships view

In the Relationships view you can see all of the imported tables, with some of them connected by lines. Each line represents a physical relationship. For example, there is a line connecting the Employee and Sale tables. The line has a 1 on the Employee end and an asterisk on the Sale end. This means that the relationship is one-to-many with Employee on the one side and Sale on the many side of the relationship. This line is solid, which means that the relationship is active. Finally, there is an arrow pointing in the

direction of Sale, which means filtering the Employee table also filters the Sale table, but the reverse is not true.

Tables can be moved around by dragging their headers. You can also resize a table by clicking on its edge and dragging it. If you want the layout to be decided by Power BI Desktop, you can click on the Reset Layout button, which is in the bottom-right corner of the Relationships view and looks like a circular arrow. If needed, the zoom can be changed by adjusting the slider near the Reset Layout button. The right-most button adjusts the zoom to a level in which all the elements fit in a page.

If you click on a relationship line, the columns that are part of the relationship in each table will be highlighted. In the Employee table, it is the Employee Key column, and in the Sale table, it is the Salesperson Key column. A relationship can only be created between two columns; it is not possible to create a relationship between two columns in table A and two columns in table B, for instance. To create a physical relationship between multiple columns in one table and multiple columns in another table, you will need to create one new column in each table that combines the relationship columns. This can be achieved both in Power Query Editor and by creating a calculated column with DAX.

> ***NOTE*** **CREATING A RELATIONSHIP WITH MULTIPLE COLUMNS**
>
> For more details on how to create a relationship between several columns, see an article by Reza Rad, “Relationship in Power BI with Multiple Columns” at: *http://radacad.com/relationship-in-power-bi-with-multiple-columns*.

In Power BI, it is possible to have more than one relationship between any two tables, but no more than one can be active at a time. This also means that tables can have multiple inactive relationships or no relationship at all. For example, there is an inactive relationship between the Date column from the Date table and Invoice Date Key column from the Sale table. This line is dashed, which signifies an inactive relationship. Apart from the line style, the relationship looks the same as others: it has a 1 and an asterisk at its ends, as well as the filter direction arrow. Inactive relationships can be

activated using the USERELATIONSHIP function in DAX, covered later in the chapter.

Having several relationships between two tables can be a substitute for having role-playing dimensions, which are tables that are duplicated to filter the same table by different keys. An example is a calendar table that filters a sales table by order date, and another calendar table that filters the same sales table by delivery date. With multiple relationships, you could filter the sales table by both delivery date and order date using the USERELATIONSHIP function. DAX is going to be covered in Section 2.2, “Create calculated columns, calculated tables, and measures.”

You can edit a relationship by double-clicking on it, and you can delete one by right-clicking on a relationship and selecting Delete. Alternatively, you can bring up a list of all relationships by selecting **Home** > **Relationships** > **Manage Relationships**, or by choosing **Modeling** > **Relationships** > **Manage Relationships**. The Manage Relationships window is shown in Figure 2.4.

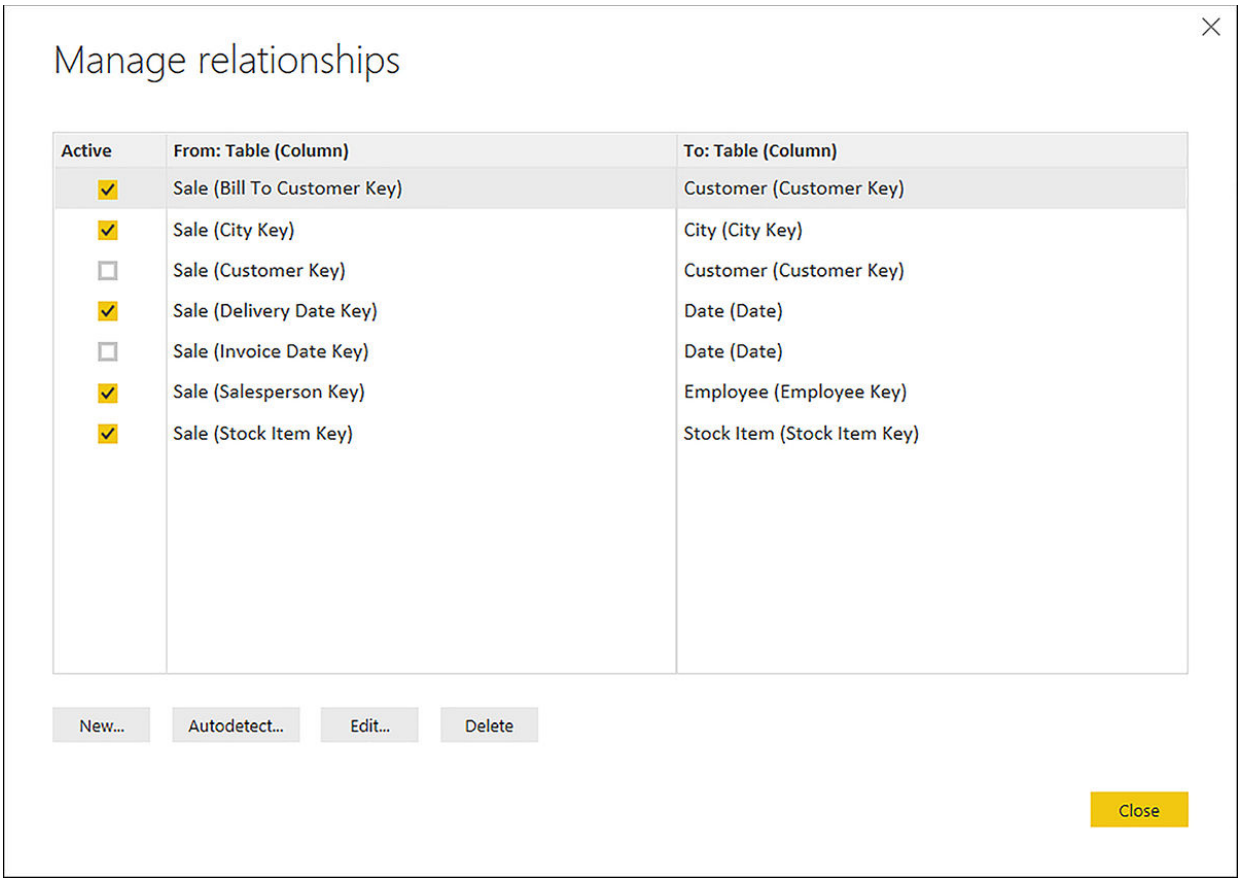


FIGURE 2.4 Manage Relationships window

When you edit a relationship by either double-clicking it in Relationships view or Edit in Manage Relationships, the Edit Relationship window opens. Figure 2.5 shows the Edit Relationship window when editing the relationship between the Bill To Customer Key column in the Sale table and the Customer Key column in the Customer table.

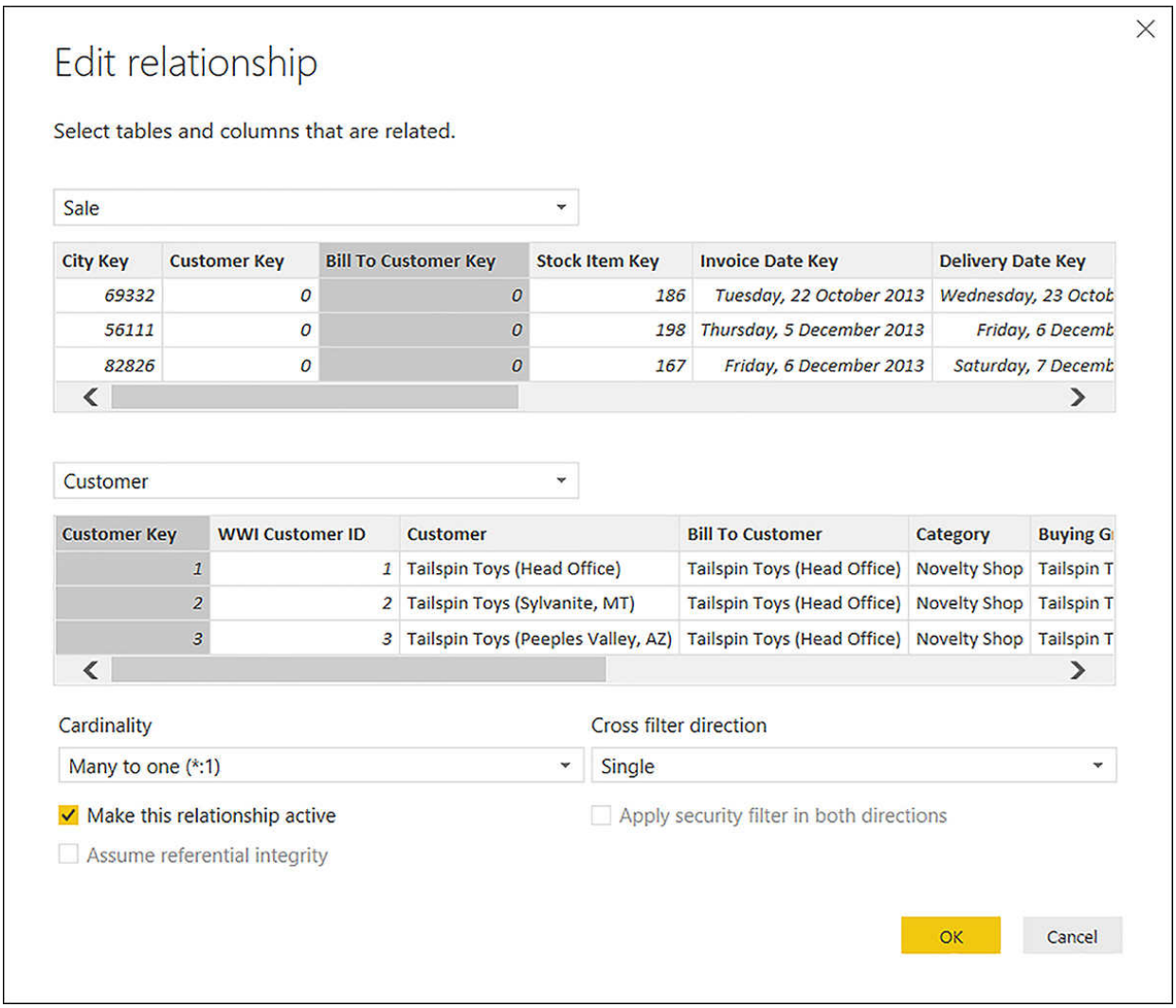


Figure 2.5 Edit Relationship window

In the Edit Relationship window, the column selection area looks similar to the Merge Queries window we reviewed in the previous chapter. Tables can be selected from drop-down lists, and for each table, you pick the column that will be part of the relationship. Below, you can choose the cardinality of the relationship. Currently, Power BI supports only the following three options:

- Many to one (*:1)
- One to one (1:1)
- One to many (1:*)

When creating a relationship, the column on the one side of a relationship must have unique values, and there must be no blank values. If you attempt

to use a column with a blank value in the one side of a relationship, you will see an error message like the following: “You can’t create a relationship between these two columns because the ‘ProductKey’ column in the ‘Product’ table contains null values.” Empty strings are allowed given that there is only one empty string in the column.

If you try to create a relationship between two columns, neither of which has unique values, you will see the following error message: “You can’t create a relationship between these two columns because one of the columns must have unique values.” If you need to create a many-to-many relationship, also known as M2M, you will need to create a bridge table that contains unique values.

> ***NOTE*** **MANY-TO-MANY RELATIONSHIPS IN POWER BI**
>
> Many-to-many relationships can be found in many businesses. For example, a client can have multiple bank accounts, including a joint account with their partner. For more details on how to enable many-to-many relationships in Power BI, see an article by Marco Russo, “Many-to-many relationships in Power BI and Excel 2016” at: *https://www.sqlbi.com/articles/many-to-many-relationships-in-power-bi-and-excel-2016/*.

The Cross filter direction drop-down list has two options: Single or Both.

- The Single option means that when you filter the table that is on the one side of a relationship, the table on the many side of the relationship is filtered as well, but filtering the many side of the table does not automatically filter the one side of the table.
- The Both option, also known as a bidirectional relationship, makes sure that both tables filter each other. This option should be considered carefully, however, because it results in performance implications. Furthermore, this setting may prevent you from creating active relationships in certain cases.

> ***NOTE*** **BIDIRECTIONAL RELATIONSHIPS AND AMBIGUITY**

For more information on bidirectional relationships in Power BI and why sometimes relationships are inactive and cannot be made active, you can read Melissa Coates's article, "Why Is My Relationship Inactive in Power BI Desktop?" at: *http://www.sqlchick.com/entries/2015/11/7/why-is-my-relationship-inactive-in-power-bi-desktop*.

Filters can flow through a chain of relationships, and the length of the chain does not matter. For example, if you have Category and Subcategory in a one-to-many relationship, and Subcategory and Product in a one-to-many relationship, then the Category will also filter the Product. Let's also say that Product and Sales are related in a one-to-many relationship, and Sales is related to Calendar in many-to-one relationship. This means that filters will flow all the way from Category through Subcategory and Product to Sales, but Category will not filter Calendar unless the relationship between Sales and Calendar has Cross filter direction set to both.

We can review the effect of bidirectional relationships with the following example:

1. In the Report view, click on the check box next to the Calendar Year Label field from the Date table. This will create a table visual.
2. Click on the check box next to the Lineage Key from the City table.

The Date and City tables are in a many-to-many relationship through the Sale table. At this stage, our table visual should look as follows:

TABLE 2-1 A table with a Calendar Year Label and a Lineage Key from the City table

Calendar Year Label	Lineage Key
	116295
CY2013	116295
CY2014	116295
CY2015	116295
CY2016	116295
Total	116295

There are two things to note in Table 2-1. First, all Lineage Key values, which are counts, are the same. Second, the first row has a blank value for the Calendar Year Label, even though we only have four values in the column. This can be confirmed in the following way:

1. Open Power Query Editor.
2. Select the Date table.
3. Press Ctrl + G and double-click on the Calendar Year Label column.
4. Click on the AutoFilter arrow in the column's header.
5. Click Load More next to the "List may be incomplete" Message.

Note that there are only four values: CY2013, CY2014, CY2015, and CY2016. In Table 2-1, we see a blank value among Calendar Year Label values because both the Date and City tables have a many-to-one relationship with the Sale table. The Date table's active relationship is with the Delivery Date Key column in the Sale table, which contains null values. For all values on the many side of a relationship that do not have a corresponding value on the one side of the relationship, Power BI adds a virtual blank row to the table on the one side. The number of values without a match does not matter because only one virtual blank row is added. However, if there are extra values on the one side, an extra row is not added to the many side. Filtering out values without a match from the table on the many side removes the blank row.

You can now see how bidirectional filtering affects calculations. If you change the cross filter direction of the relationship between the Sale and City

tables from Single to Both, the table visual will look as shown in Table 2-2.

TABLE 2-2 A table with Calendar Year Label and Lineage Key from the City table after applying bidirectional filtering

Calendar Year Label	Lineage Key
	76
CY2013	872
CY2014	829
CY2015	786
CY2016	656
Total	116295

Note that the values are now all different, and the Total row still shows the same figure: 116,295. The filters from the Date table now reach the City table. Here is how the table can be read:

- For sales with no delivery date, there are 76 rows in the City table.
- In CY2013, orders have been delivered to 872 cities.
- In CY2014, orders have been delivered to 829 cities.
- In CY2015, orders have been delivered to 786 cities.
- In CY2016, orders have been delivered to 656 cities.
- In total, we have 116,295 rows in the City table.

Note that the word “Total” in the table is slightly misleading. This is not a subtotal; instead, it shows a value with no filters from the column applied. In this case, when the Calendar Year Label column is not filtered, the Date table is not filtered either, so the Sale table is not filtered by the Date table. As a result, the City table is not filtered by the Date table, and we have 116,295 rows in the City table, which explains the results that we see. At this stage, we should change the cross filter direction of the relationship back to Single.

If you choose the one-to-one cardinality in relationship settings, then Both will be automatically selected as the cross filter direction. If you try to

choose Single, you will see the following error message: “The filter direction you selected isn’t valid for this relationship.”

Below the Cardinality drop-down list, there is a Make This Relationship Active check box with which you choose whether a relationship is active. Below the cross filter direction drop-down list, there is an Apply security filter in both directions check box. This check box is deactivated if you choose Single as the cross filter direction. If you select Both as the cross filter direction, you will be able to change the default selection, which is unchecked. This check box is responsible for carrying the security filters from the many-to-one side of a relationship when you use Row-Level Security, which gives you more granular control over filtering. Because bidirectional relationships might result in poor performance, allowing security filters to flow from the many side to the one side of a relationship might be undesirable.

> ***MORE INFO*** **ROW-LEVEL SECURITY IN POWER BI**
>
> More information on Row-Level Security and how to configure it can be found in Section 3.4: “Configure security for dashboards, reports, and apps.”

The third check box in the Edit Relationship window, Assume Referential Integrity, is only relevant if you use the DirectQuery data connectivity mode. This option results in more efficient queries because queries will be using INNER JOIN statements instead of OUTER JOIN. For the Assume Referential Integrity option to run correctly, there are two requirements.

- First, the values in the column used on the many side of a relationship must never be null or blank.
- Second, each value on the many side of a relationship must have a corresponding value in the column used on the one side of a relationship.

If these conditions are not met, you will still be able to enable the option in some cases, but you may see inconsistent results in your visuals.

> ***MORE INFO*** **ASSUME REFERENTIAL INTEGRITY SETTING**

For more details on the Assume Referential Integrity setting, including examples and what happens when you enable the setting when the requirements for using it properly are not met, see the "Assume referential integrity settings in Power BI Desktop" article in Power BI documentation at: *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-assume-referential-integrity/*.

If needed, Power BI Desktop can try to autodetect relationships when you click Autodetect in the Manage Relationships window, though this feature is not always reliable. To review the functionality, delete some existing relationships and try to use this feature to re-create the relationships. At the same time, review the effect of having no active relationships between tables.

First, let's go back to the Report view and create a new visual by checking the Total Including Tax field from the Sale table. A clustered column chart will be created automatically. We will review working with visuals in detail later in this chapter. For now, click on the Bill To Customer field in the Customer table. This will put Bill To Customer into the Axis field well in the clustered column chart we created earlier. The visual can be seen in Figure 2.6.

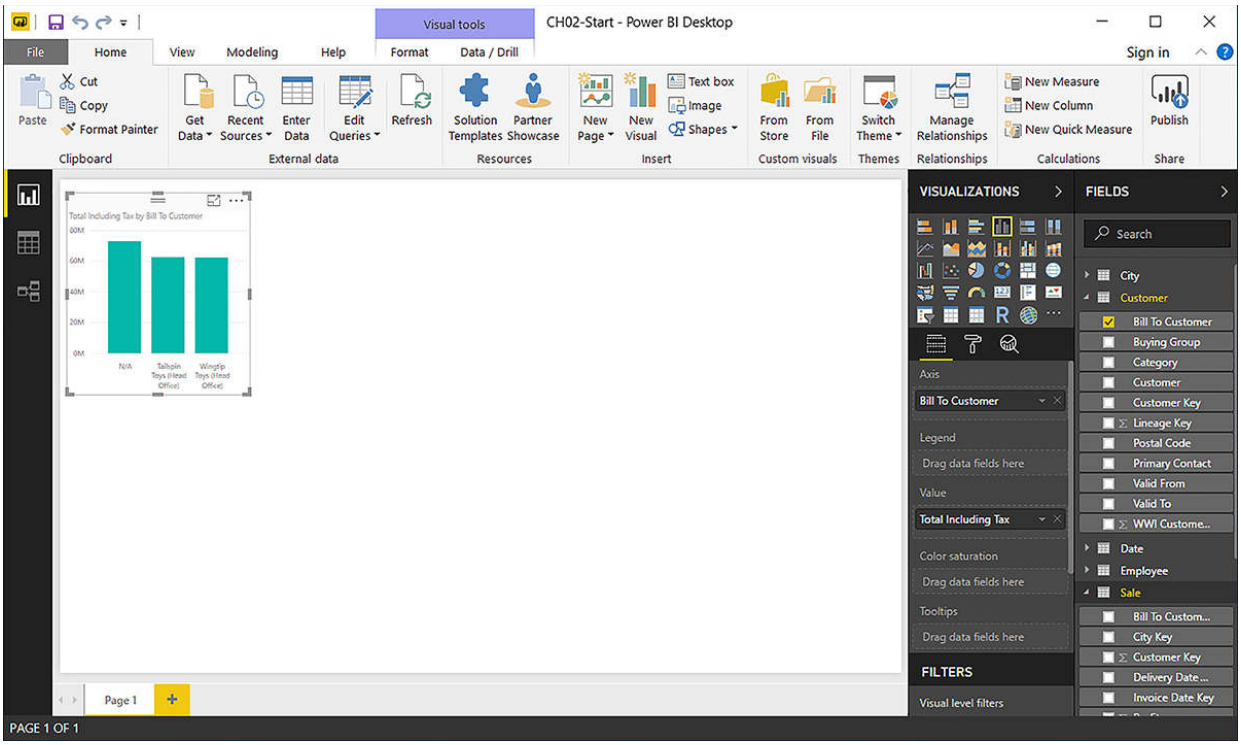


FIGURE 2.6 Bill To Customer and Total Including Tax in a visual

Note that the three columns are of different lengths. To make the difference easier to see, click on the ellipsis in the top-right corner of the visual and select Show Data, which brings up a table underneath the graph. The table has two columns: Bill To Customer and Total Including Tax. There are three rows, and each row has a different value. The table is shown in Table 2-3.

TABLE 2-3 Bill To Customer and Total Including Tax values with relationship

Bill To Customer	Total Including Tax
N/A	73,037,043.78
Tailspin Toys (Head Office)	62,654,262.56
Wingtip Toys (Head Office)	62,352,133.11

Next, select **Home** > **Relationships** > **Manage Relationships**. The first relationship, the one from Sale (Bill To Customer Key) to Customer (Customer Key) is selected. Click Delete underneath the list of relationships.

You will then be asked to confirm your action; you should select Delete again. Once you click Close, you will see the numbers in the table change, and the columns in the chart will be of the same length. The new table values can be seen in Table 2-4.

TABLE 2-4 Bill To Customer and Total Including Tax values without relationship

Bill To Customer	Total Including Tax
N/A	198,043,439.45
Tailspin Toys (Head Office)	198,043,439.45
Wingtip Toys (Head Office)	198,043,439.45

The new values are all the same because the Customer table does not filter the Sale table now. Even though there is a relationship, it is inactive. Therefore, each row displays the total for all customers.

EXAM TIP

When every value in a visual is the same, there is a good chance that something is wrong with the relationships. In an exam question, if you are given information that actual values are different from what they appear to be in a visual, you should create an active relationship.

Before recreating the relationship, review the Autodetect functionality. Let's open the Manage Relationships window again and select Autodetect. After a few seconds, the following message will be displayed: "Found 1 new relationship(s)." Upon clicking Close, you can see the new relationship at the bottom of the list: from Date (ISO Week Number) to Customer (Customer Key). Because this relationship is obviously meaningless, we should delete it. If you click New, the Create Relationship window will be opened, which looks the same as the Edit Relationship window. Select the Customer Key column from the Customer table on the one side and the Bill To Customer Key from the Sale table on the many side of the relationship and keep all other settings as-is. After clicking OK and Close, the values in

the table will again be the same as Table 2-3. There is an alternative way to create relationships. In the Relationships view, you can drag a field from one table on top of a field from another table to create a relationship between the two tables. The direction in which you drag the fields does not matter. Power BI Desktop will automatically detect which column is on the one side and which one is on the many side, or it might create a one-to-one relationship if it is appropriate.

Note that if upon data refresh, a column that is on the one side of a relationship contains some duplicate or null values, the data refresh will be canceled and old data will be preserved.

***MORE INFO* RELATIONSHIPS IN POWER BI DESKTOP**

For a video overview of relationships in Power BI Desktop, see "How to Manage Your Data Relationships" at: *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-2-2-manage-data-relationships/*.

For more examples, see the official documentation article, "Create and manage relationships in Power BI Desktop" at: *https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-create-and-manage-relationships/*.

Optimize models for reporting

After creating relationships, you may want to optimize your data model for reporting. This includes tasks such as defining the sort order of columns by other columns and hiding extraneous fields.

Sort by column

In some cases, you might need to override the default sort order of a column. For example, if you go to the Report view and create a new visual by clicking on the check box next to the Month field in the Date table, a table visual will be created with values from this field, shown in Figure 2.7 below.

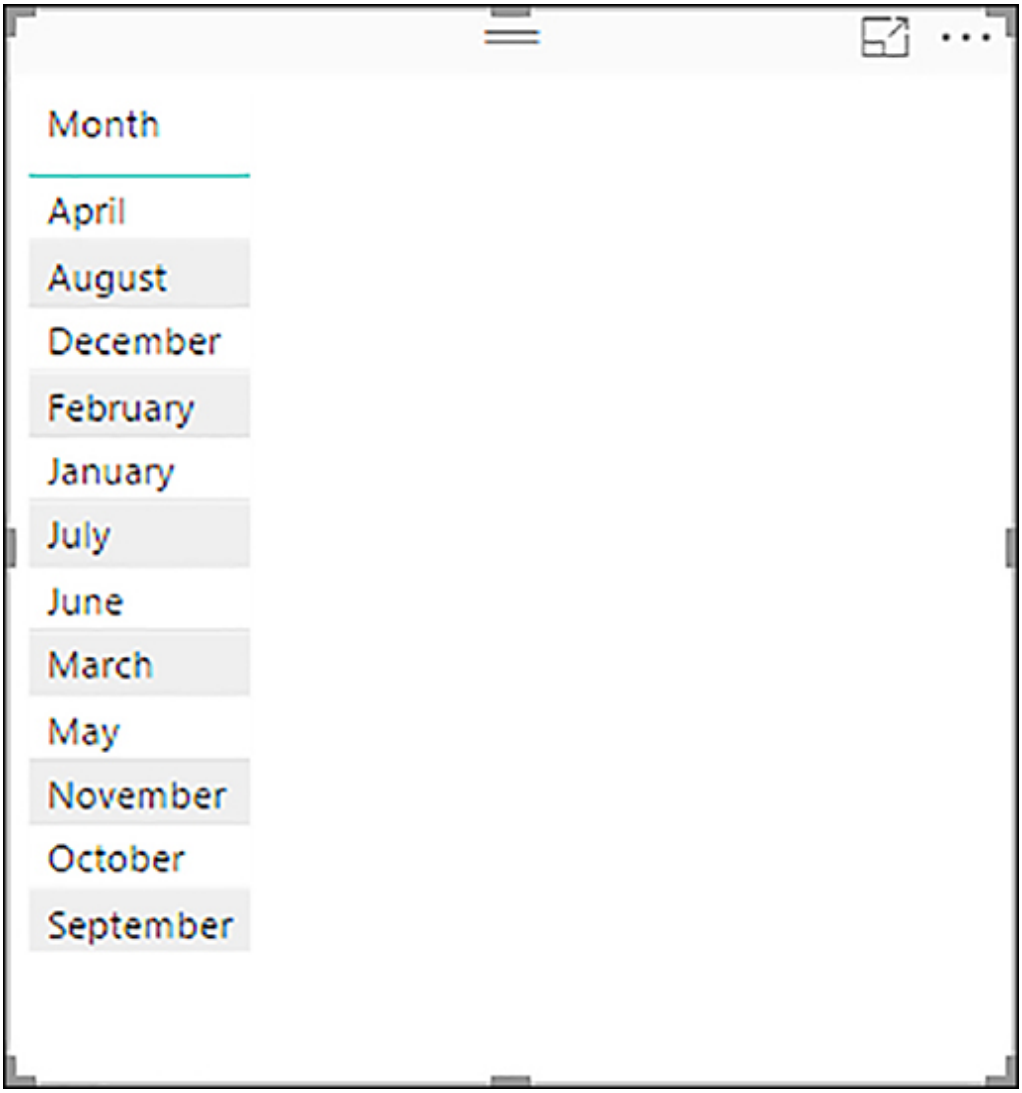


Figure 2.7 Table visual with Month as the only field

Note that the values appear in alphabetic order. We can change the sorting order of this column, given that we have another column to sort by. In this case, we can sort the Month column by the Calendar Month Number column. To do this, click on the Month field name in the Fields pane, then select **Modeling** > **Sort** > **Sort by Column**. You will then see a list of columns from the Date table. In this case, select Calendar Month Number. You can then immediately notice that the values now appear in the right order: from January to December.

When you sort column A by column B, each value in column A must have only one corresponding value in column B. However, each value in column B does not necessarily need to have only one corresponding value in column

A. For example, create a table visual with the Calendar Month Label field from the Date table. Because we have not yet applied a new sort order, the values are sorted alphabetically.

At this stage, go to the Data view, which is divided into two parts: most of the space is taken up by a table preview, and on the right side we can see the Fields pane that is almost the same as in the Report view. Note that at the bottom of the Data view, there is a line that gives you the following information:

- Table selected
- Number of rows in the table
- Column selected
- Number of distinct values in the column

This feature can be very useful when you want to quickly understand how many rows there are in a table or how many distinct values a column has.

If you select the Calendar Month Number column, you will see the following information below the table preview: “TABLE: Date (1,461 rows) COLUMN: Calendar Month Label (12 distinct values)”. If you attempt to sort the Calendar Month Number column by the Calendar Month Label column, you will encounter the following error message: “We can’t sort the ‘Calendar Month Number’ column by ‘Calendar Month Label’. There can’t be more than one value in ‘Calendar Month Label’ for the same value in ‘Calendar Month Number’. Please choose a different column for sorting or update the data in ‘Calendar Month Label’.”

However, you can successfully sort the Calendar Month Label column by the Calendar Month Number column, even though there are 48 distinct values in the former and 12 in the latter.

If you now go to the Report view, you will see that month names in Calendar Month Label values are sorted correctly, though the year-month combinations are not correct. First, you see four Januaries; one for each of the four years, then four Februaries, and so on. To mitigate this, create a custom column in Power Query Editor that has the same number of distinct values as the Calendar Month Label column, then sort the Calendar Month Label column by the new column. To achieve this, do the following:

1. Open Power Query Editor.

2. Select the Date query.
3. Create a custom column called “Year Month Number” with the following formula:

```
= [Calendar Year] * 100 + [Calendar Month Number]
```

4. Click OK.
5. In the formula bar, after “[Calendar Month Number]” and before the closing parenthesis, add “, Int64.Type” without quotation marks. The whole step formula should be the following:

Click here to view code image

```
= Table.AddColumn(Dimension_Date, "Year Month Number", each
[Calendar Year] * 100 + [Calendar Month Number], Int64.Type)
```

6. Rename the step to AddedYearMonthNumber. If you open the Advanced Editor, you should see the following code in Listing 2-1.

LISTING 2-1 Date query after adding Year Month Number

Click here to view code image

```
let
    Source = Sql.Databases("localhost"),
    WideWorldImportersDW =
Source{[Name="WideWorldImportersDW"]}[Data],
    Dimension_Date =
WideWorldImportersDW{[Schema="Dimension",Item="Date"]}
[Data],
    AddedYearMonthNumber = Table.AddColumn(Dimension_Date,
"Year Month Number", each [Calendar Year] * 100 + [Calendar
Month Number], Int64.Type)
in
    AddedYearMonthNumber
```

7. Click Close & Apply in Power Query Editor.
8. Go to the Data view.
9. Select the Calendar Month Label column.
10. Change the sort column for Calendar Month Label from Calendar Month Number to Year Month Number.
11. Go to the Report view.

***NOTE* CUSTOM COLUMN DATA TYPES**

In step 5 above, we added a piece of code to our formula that sets the data type for the new column to Whole Number, yet still allows Query Folding to take place. If you apply data type as a new step instead, Query Folding will break.

Notice that the values in the Calendar Month Label column are now sorted correctly. You should also create a new column called "Fiscal Year Month Number" to sort the Fiscal Month Label column values correctly. To create the new column, follow a process like before:

1. Right-click on the Date table in the Fields pane and select Edit Query. This will open Power Query Editor, taking you directly to the Date query.
2. Create a custom column called "Fiscal Year Month Number" with the following formula:

   ```
   = [Fiscal Year] * 100 + [Fiscal Month Number]
   ```

3. Insert ", Int64.Type" without quotation marks between "[Fiscal Month Number]" and the closing parenthesis, so that your step formula looks like the following:

   Click here to view code image

   ```
   = Table.AddColumn(AddedYearMonthNumber, "Fiscal Year Month
   Number", each [Fiscal Year] * 100 + [Fiscal Month Number],
   Int64.Type)
   ```

4. Rename the step to AddedFiscalYearMonthNumber, so that the whole query code is as follows:

LISTING 2-2 Date query after adding Fiscal Year Month Number

Click here to view code image

```
let
    Source = Sql.Databases("localhost"),
    WideWorldImportersDW =
Source{[Name="WideWorldImportersDW"]}[Data],
    Dimension_Date =
WideWorldImportersDW{[Schema="Dimension",Item="Date"]}
```

```
[Data],
    AddedYearMonthNumber = Table.AddColumn(Dimension_Date,
"Year Month Number", each [Calendar Year] * 100 + [Calendar
Month Number], Int64.Type),
    AddedFiscalYearMonthNumber =
Table.AddColumn(AddedYearMonthNumber, "Fiscal Year Month
Number", each [Fiscal Year] * 100 + [Fiscal Month Number],
Int64.Type)
in
    AddedFiscalYearMonthNumber
```

5. Click Close & Apply.

Change the sorting order of columns in the Date table as shown in Table 2-5.

TABLE 2-5 New sort columns

Column Name	Sort by Column
Fiscal Month Label	Fiscal Year Month Number
Day	Day Number
Short Month	Calendar Month Number

Once you have done this, all columns in the Date table will be sorted correctly.

> ***MORE INFO*** **SORT BY COLUMN**
>
> For more examples and details on the Sort by Column feature in Power BI Desktop and sorting in general, see "Sort by column in Power BI Desktop" at: *https://docs.microsoft.com/en-us/power-bi/desktop-sort-by-column*.

Hide fields and tables

Most data models contain data that is required for technical purposes, but not needed for reporting. A typical example is the key columns: they are required for relationships, but keys are rarely used in visualizations. It is best practice to hide such column in report view. This is different from deleting

columns: the columns are still kept in the data model, but they are hidden from the Report view unless you unhide or choose to show them.

To hide a field, right-click on it in the Fields pane and select Hide. To view all fields, including hidden ones, you can right-click on any field and select View Hidden. This will show hidden fields in darker color alongside the fields that are not hidden. You can also unhide all hidden fields by right-clicking on any field and selecting Unhide All. Note that you will not be asked to confirm your action. Alternatively, you can go to the Data view, where hidden fields are always shown. The header and values of hidden columns appear darker in table previews. You can also hide whole tables in the same way as hiding fields.

Apart from the Report and Data views, you can also hide fields and tables in the Relationships view.

In our Wide World Importers example, hide the columns shown in Table 2-6.

TABLE 2-6 Columns to be hidden

Table	Column
City	City Key
City	Lineage Key
City	Valid From
City	Valid To
City	WWI City ID
Customer	Customer Key
Customer	Lineage Key
Customer	Valid From
Customer	Valid To
Customer	WWI Customer ID
Date	Calendar Month Number
Date	Calendar Year
Date	Day Number
Date	Fiscal Month Number
Date	Fiscal Year
Date	Fiscal Year Month Number
Date	Year Month Number
Employee	Employee Key
Employee	WWI Employee ID
Sale	City Key
Sale	Customer Key
Sale	Delivery Date Key
Sale	Invoice Date Key
Sale	Salesperson Key
Sale	Stock Item Key

Table	Column
Sale	WWI Invoice ID
Stock Item	Lineage Key
Stock Item	Stock Item Key
Stock Item	Valid From
Stock Item	Valid To
Stock Item	WWI Stock Item ID
Target	Bill To Customer
Target	Calendar Year

Formatting columns

When importing data to Power BI, it applies default formatting to all columns. Formatting is different from setting data types: when you format a column, the underlying data types stays the same, but visually it appears differently. Text, True/False, and Binary data types have only one formatting option each. For the following data types, change the formatting:

- Decimal number
- Fixed decimal number
- Whole number
- Date/time
- Date
- Time

Depending on the data type, different formatting options are available. The first three data types—decimal, fixed decimal, and whole number—share the available formatting options. For example, you can format a date column to display the year only, or you can format a Decimal Number to appear as a Whole Number. Note that for all calculation purposes, values will stay the same, so decimals will add up as expected.

To format a column, select it in the Fields pane or in the Data view, then select **Modeling** > **Formatting** > **Format** and select the desired format.

When you change formatting in the Data view, you will immediately see the new format applied.

Apply the formatting options shown in Table 2-7 to the specified columns.

TABLE 2-7 New column formatting

Table	Column	Format	Options
City	Latest Recorded Population	Whole number	Thousands separator
Date	Date	*14/03/2001 (d/MM/yyyy)	
Sale	Profit	Currency general	
Sale	Quantity	Whole number	Thousands separator
Sale	Tax Amount	Currency general	
Sale	Total Excluding Tax	Currency general	
Sale	Total Including Tax	Currency general	
Sale	Unit Price	Currency general	
Stock Item	Recommended Retail Price	Currency general	
Stock Item	Unit Price	Currency general	
Target	Target Amount Excluding Tax	Currency general	
Target	Target Quantity	Whole number	Thousands separator

***NOTE* DATE FORMATS IN POWER BI**

The list of available date formats in Power BI Desktop depends on your locale. For example, if you are using a computer with locale set to the United States, you may not see the d/mm/yyyy format.

Instead, you should select the first option that has an asterisk in front of it.

NOTE **OPTIMIZING DATA MODELS FOR REPORTING**

For a video overview and more examples of how you can optimize your data model for reporting purposes, see "Optimizing Data Models for Better Visuals" at *https://powerbi.microsoft.com/en-us/guided-learning/powerbi-learning-2-4-optimize-data-models/*.

Manually type in data

Because not all data needs to come from a source, Power BI has an option to enter data manually. You can type in your data manually by selecting **Home** > **External data** > **Enter Data**, which opens the Create Table window (Figure 2.8). The same button is available in Power Query Editor in the New Query section of the Home tab.

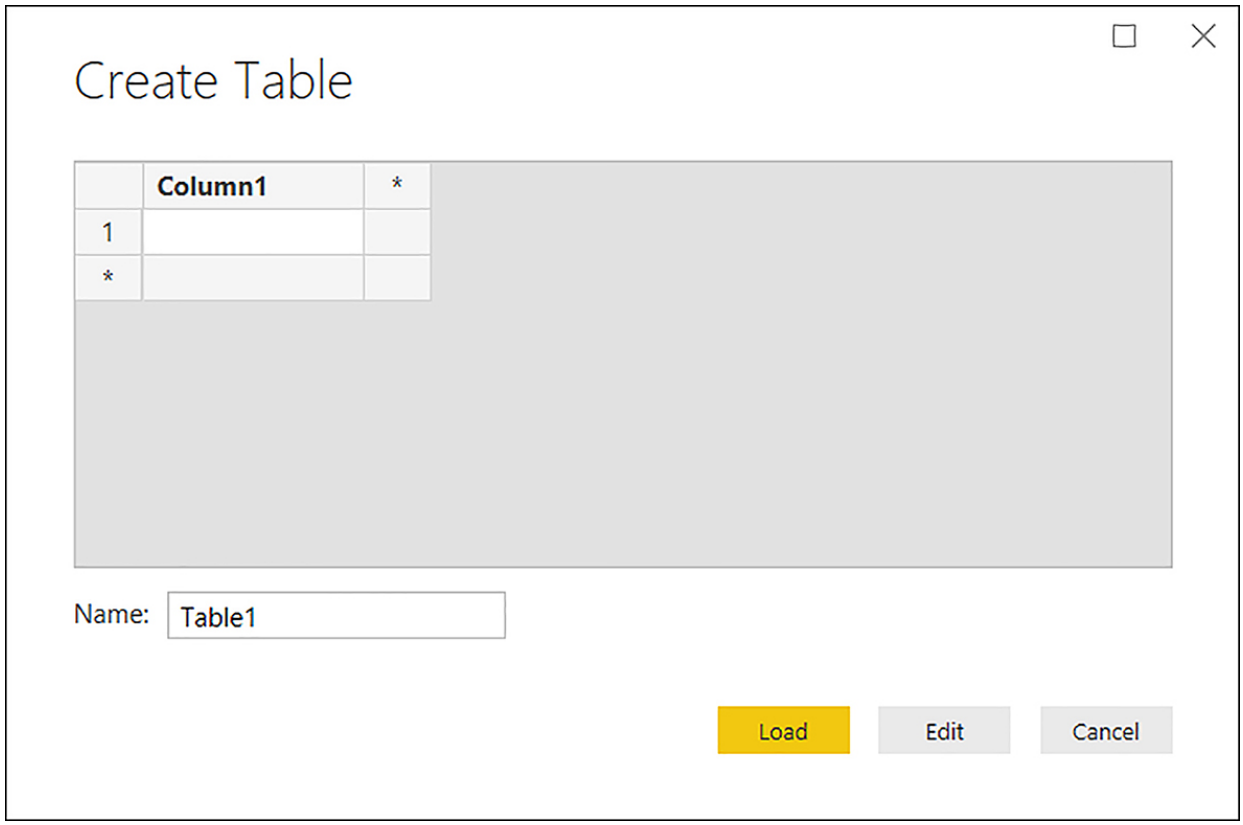


FIGURE 2.8 Create Table window

In the Create Table window, you can type data manually, as well as paste values. When you paste values, Power BI tries to analyze the data, and if it deems it appropriate, it will promote the first row of data to headers. If this is erroneous, you will be able to undo the action by clicking Undo Headers in the same window. Pasted values can be edited in this window as well. You can delete whole rows or columns by right-clicking on a row or column header and selecting Delete. In addition to that, you can insert new rows and columns either by clicking on an asterisk in the row or column headers, or right-clicking on a row or column header and selecting Insert. Standard operations such as Cut, Copy, and Paste are allowed in this window. You can select the whole table by pressing Ctrl + A.

Once you are satisfied with the entered data, you can either load it directly to the data model, or you can edit it in Power Query Editor before loading. Enter the following table, Table 2-8, and name it Scale:

TABLE 2-8 Scale table to be entered manually

Scale
1
1000
1000000

Once you are done, click Edit. Note that Power Query automatically detected the data type of the column—Whole Number. Power Query would do it regardless of our choosing to load the table directly or edit it first.

Note that the Source step has a gear icon next to it. Click it, and you will see the Create Table. This way, you can edit your data in the Create Table window even after you close the window.

Power BI uses compressed JSON to convert the typed data into a table. If you open Advanced Editor, you will see the code in Listing 2-3, which has been formatted for presentation purposes.

LISTING 2-3 Entered data M code

Click here to view code image

```
let
    Source = Table.FromRows(
        Json.Document(Binary.Decompress(
            Binary.FromText("i45WMlSK1QGSBgYGcAaYHQsA ",
BinaryEncoding.Base64),
            Compression.Deflate)
        ),
        let _t = ((type text) meta [Serialized.Text = true]) in
type table [Scale = _t]
    ),
    #"Changed Type" = Table.TransformColumnTypes(Source,
{{"Scale", Int64.Type}})
in
    #"Changed Type"
```

> ***NOTE*** **ENTERING DATA DIRECTLY INTO POWER BI DESKTOP**
>
> For more examples on manual data entry, see "Enter data directly into Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-enter-data-directly-into-desktop*.

Use Power Query

Power Query enables you to create tables using pure M. For example, you can create Table 2-8 from a list, from records, or by using a table construct.

Lists

Lists can be defined in Power Query inside curly braces with values separated by commas. This is an example of a list that contains three elements—1, 2, and 3:

```
{ 1, 2, 3 }
```

Because the list contains three consecutive integers, it can be defined using the range notation with two periods between integers:

```
{ 1 .. 3 }
```

Because each symbol has a corresponding Unicode number, you can also define ranges of symbols using the same notation. The following list produces the Latin alphabet in lowercase:

```
{ "a" .. "z" }
```

Lists do not need to contain values of the same data type. The following list is valid:

```
{ "a", true, 2 }
```

Table 2-8 can be reproduced in M by following these steps:

1. In Power Query Editor, select **Home** > **New Query** > **New Source** > **Blank Query**.
2. Rename the query to ScaleFromList and disable its loading.
3. In the formula bar, type the following code:
   ```
   = { 1, 1000, 1000000 }
   ```
4. Select **List Tools Transform** > **Convert** > **To Table**.
5. In the To Table dialog box, accept the default settings and click OK.
6. Double-click on the Column1 header and type Scale.
7. Change the data type of the Scale column to Whole Number.

The resulting table is equivalent to the table we entered manually. If you open Advanced Editor, you should see the code from Listing 2-4.

LISTING 2-4 Scale table from list

Click here to view code image

```
let
    Source = { 1, 1000, 1000000 },
    #"Converted to Table" = Table.FromList(Source,
Splitter.SplitByNothing(), null,
null, ExtraValues.Error),
    #"Renamed Columns" = Table.RenameColumns(#"Converted to
Table",{{"Column1",
"Scale"}}),
    #"Changed Type" = Table.TransformColumnTypes(#"Renamed
Columns",{{"Scale",
Int64.Type}})
in
    #"Changed Type"
```

If you import a list into your data model, it will be converted to a table with one column that shares its name with the query. The column will be of type text regardless of values stored in the list.

Records

Records can be defined in M in square brackets following the “field name = value” notation. One record can contain multiple fields. The following is an example of a record with two fields: Quantity and Unit Price.

```
[ Quantity = 2, Unit Price = 3 ]
```

The Scale table from Table 2-5 can be built from records with the following code.

LISTING 2-5 Scale table from records

Click here to view code image

```
Table.FromRecords (
    {
        [Scale = 1],
        [Scale = 1000],
        [Scale = 1000000]
    },
    type table [Scale = Int64.Type]
)
```

If you import a record into your data model, it will be converted into a table with two text columns: Name and Value. The number of rows will correspond to the number of fields in the record.

Tables

Tables can be created in M using the #table construct, which can result in much more concise M queries. You can either specify columns as a list with no types defined, or you can specify column names and types in a record, like in Listing 2-6.

LISTING 2-6 Table constructs in M

Click here to view code image

```
// Untyped columns
#table (
    {"Quantity", "Unit Price"},
    {
        { 2, 3 },
        { 5, 7 }
    }
)

// Columns in data types defined
#table (
    type table [Quantity = Int64.Type, Unit Price =
Currency.Type],
    {
        { 2, 3 },
        { 5, 7 }
    }
)
```

The code from Listing 2-7 re-creates the Scale table from Table 2-8 in M:

LISTING 2-7 Scale table using the table construct in M

Click here to view code image

```
#table (
    type table [Scale = Int64.Type],
    {
        { 1 },
        { 1000 },
        { 1000000 }
    }
)
```

Lists, records, and tables can be nested inside each other. When these elements are imported, they are converted to text cells with “[List]”, “[Record]”, and “[Table]” written, respectively.

We will need only one Scale table later in this chapter, so if you defined more than one Scale table, you should delete the extras. You can then click Close & Apply in Power Query Editor.

EXAM TIP

The exam does not test you on advanced M, but you should be aware of the general M syntax and structured data values available in M. For a general overview of M syntax, see "Expressions, values, and let expression" at: *https://msdn.microsoft.com/en-us/library/mt299038.aspx*.

***MORE INFO* POWER QUERY CHEAT SHEET**

For a concise reference on Power Query data structures and basic operations, see Ivan Bondarenko's Power Query Cheat Sheet at GitHub: *https://github.com/IvanBond/Power-Query-Cheat-Sheet*.

Skill 2.2: Create calculated columns, calculated tables, and measures

With what we have reviewed so far, it is already possible to create reports and visualize data, though the calculations will be limited. For more sophisticated analysis, you might need to enrich your model with calculated columns, calculated tables, and measures. For this, Power BI uses Data Analysis Expressions (DAX). DAX is the language of Power BI, Power Pivot for Excel, and SQL Server Analysis Services Tabular.

With DAX, you can derive many more insights from your data compared to using just the existing fields. For example, DAX allows you to dynamically calculate period over period figures, as well as percentages, such as weighted averages. In this section, we are going to review the skills that are needed to perform calculations and query with DAX.

This section covers how to:

- Create DAX formulas for calculated columns
- Calculated tables
- Measures

- Use What If parameters

Create DAX formulas for calculated columns

DAX is a functional language that resembles the Excel formula language, and there are many functions that appear in both. Unlike the M language, DAX is not case-sensitive in most cases. At the same time, there are some important differences:

- In DAX, there is no concept of a cell. If you need to get a value from a table, you will need to filter a specific column down to that value.
- DAX is strongly typed: it is not possible to mix values of different data types in the same column.

A calculated column is an additional column in a table that you define with a DAX formula. The difference between a custom column created with M and a calculated column created with DAX is that the latter is based on data that has already been loaded into your model. Furthermore, calculated columns do not appear in Power Query Editor.

You can create a calculated column by selecting **Modeling** > **Calculations** > **New Column**. This will create a calculated column in the table that is selected in the Fields pane. Alternatively, you can right-click on a table in the Fields pane and select New Column. Power BI will then open a formula bar (Figure 2.9) where you can write your DAX formula, then click on the check mark or press Enter to validate the formula. Power BI will also create a new field in the Fields pane, and this new field will have a column icon next to it.

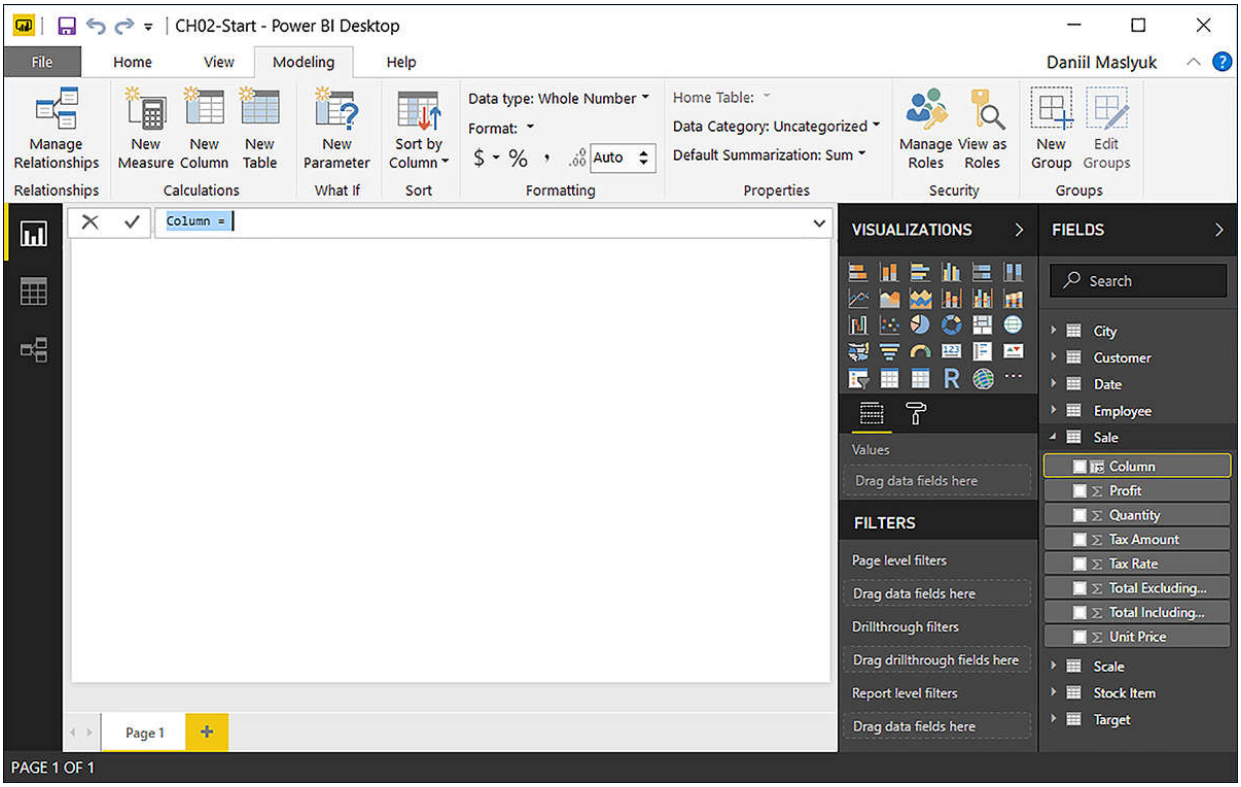


Figure 2.9 Formula bar after clicking New Column

The formula that you write is automatically applied to each row in the new column. You can reference another column in the following way:

```
'Table name'[Column name]
```

For example, calculate Unit Price including Tax by creating a calculated column in the Sale table with the following formula:

Click here to view code image

```
Unit Price Including Tax = Sale[Unit Price] * ( 1 + Sale[Tax
Rate] / 100 )
```

Note that the formula includes both the column name—it precedes the equals operator—and the column formula itself, which follows the equals operator.

The Power BI Desktop formula bar has IntelliSense enabled, and it helps you with selecting tables, columns, and functions after you type a few characters, and it also highlights syntax. Instead of copying the above formula, you can start by specifying the column name, followed by the equals operator, then start typing **uni**. At this stage, IntelliSense will give

you a list of all column and functions that have “uni” appear as part of their names (Figure 2.10). If you select a function from the list, it will also display the function’s description.

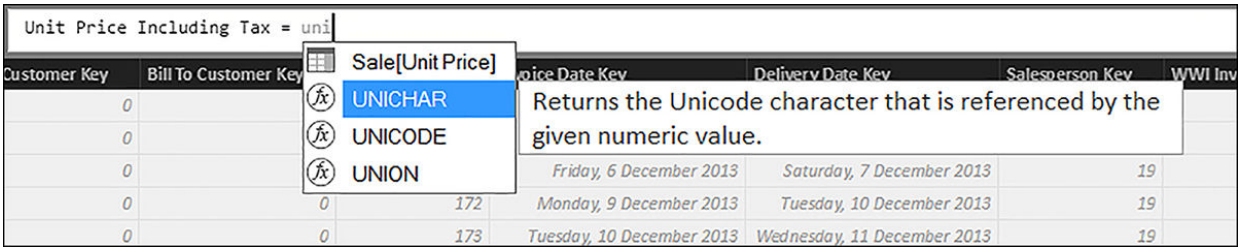


FIGURE 2.10 IntelliSense suggested values

You can navigate in this list with arrow keys on your keyboard and press Tab to auto-complete the statement. Alternatively, you can double-click on a value with your mouse, which has the same effect as pressing the Tab key.

In general, columns should always be referenced using a fully qualified syntax, which is a table name in single quotation marks followed by a column name in square brackets. If a table name does not contain spaces, does not start with a number, and is not a reserved keyword such as Calendar, then you can safely omit single quotation marks. If IntelliSense highlights a word, then it is likely a reserved keyword.

When you a referencing a column in the same table, you can use just a column name in square brackets. While this is syntactically correct, it might be difficult to read, especially because it is best practice to reference measures without table names. Measures are discussed in more detail later in this chapter.

If you want to reference a column from a table that is in a one-to-many relationship with the current table, you need to use the RELATED function. For example, you could add a Bill To Customer column to the Sale table with the following formula:

```
Customer = RELATED ( Customer[Customer] )
```

NOTE **USING RELATED WITH INACTIVE RELATIONSHIPS**

By default, DAX will use the active relationship to get the related value. It is also possible to get a related value while using an

inactive relationship.

RELATED has a companion function, RELATEDTABLE, which works in the opposite direction. For example, you could add a calculated column to the Date table that counts the number of rows in the Sale table. Because it is not possible to store a multi-row table in one row, you would also need to apply an aggregation function to RELATEDTABLE. In this case, we can use COUNTROWS, which counts the number of rows in a table:

```
Sales # = COUNTROWS ( RELATEDTABLE ( Sale ) )
```

Note that RELATEDTABLE only works in one direction by default. If you have not enabled bidirectional relationships, the following calculated column in the Date table will contain the same value for each row of the column, which is the same as the number of rows in the City table:

```
Cities # = COUNTROWS ( RELATEDTABLE ( City ) )
```

If this column is defined in the Date table, changing cross filter direction between the Sale and City tables from Single to Both makes sure that each row shows the number of cities to which we sold on a particular date.

DAX data types

Every column in a Power BI data model has exactly one data type. Currently, DAX supports the following eight data types:

- **Decimal Number** This is the most popular numeric data type. It is designed to hold fractional numbers, and it can handle whole numbers as well.
- **Fixed Decimal Number** This data type is similar to Decimal Number, but the number of decimal places is fixed at four. Internally, numbers of this type are stored as integers divided by 10,000.
- **Whole Number** This data type stores integers.
- **Date/Time** This data type stores dates and times together. Internally, values are stored as decimal numbers.
- **Date** Allows you to store dates without time. If you convert a date/time value to date, the time portion is truncated, not rounded.
- **Time** This data type stores time only, without dates.

- **Text** Stores text strings in Unicode format.
- **True/False** Also known as Boolean, this data type stores True and False values, which, if converted to a number, will be 1 and 0, respectively.

DAX can perform implicit type conversions if needed. For example, you can add TRUE to a text string, "2", and the result will be 3:

```
3 = "2" + TRUE
```

On the other hand, if you concatenate two numbers, you will get a text string as a result:

```
23 = 2 & 3
```

You can perform explicit type conversion with functions such as INT and VALUE, which convert values to integers. For example, the following expression results in 43243:

```
43243 = INT ( "2018-05-23" )
```

Dates in the form of text strings can be converted to dates using the DATEVALUE function:

```
23 May 2018 = DATEVALUE ( "2018-05-23" )
```

You can convert numeric and datetime values to text using the FORMAT function, which takes two arguments: an expression to convert and a format string. FORMAT is an example of a function that is case-sensitive. The following two expressions provide different results:

Click here to view code image

```
// AM or PM, depending on time of the day
Upper = FORMAT ( NOW (), "AM/PM" )

// am or pm, depending on time of the day
Lower = FORMAT ( NOW (), "am/pm" )
```

MORE INFO DAX FORMAT FUNCTION

To learn more about the FORMAT function in DAX, see "FORMAT Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634924.aspx*.

The format strings used in the function are based on the Visual Basic format strings. For more information on the format strings that the FORMAT function accepts, see:

- "Pre-Defined Numeric Formats for the FORMAT Function" at *https://msdn.microsoft.com/en-us/library/ee634561.aspx*.
- "Custom Numeric Formats for the FORMAT Function" at *https://msdn.microsoft.com/en-us/library/ee634206.aspx*.
- "Pre-defined Date and Time formats for the FORMAT Function" at *https://msdn.microsoft.com/en-us/library/ee634813.aspx*.
- "Custom Date and Time formats for the FORMAT Function" at *https://msdn.microsoft.com/en-us/library/ee634398.aspx*.

Blank or null values in DAX act like zeros in many cases; this behavior is very different from SQL nulls and Excel empty cells. For example, a sum of two blanks is blank. In Excel you would get 0; the sum of 1 and blank is 1, whereas in SQL, the sum is NULL. You can generate a blank value using the BLANK function. You can check whether an expression is blank with the ISBLANK function.

***MORE INFO* DATA TYPES IN POWER BI**

For a more detailed overview of data types supported in Power BI, including a table of implicit type conversions and BLANK behavior, see "Data types in Power BI Desktop" at: *https://docs.microsoft.com/en-us/power-bi/desktop-data-types*.

DAX operators

In DAX, you can use the following operators, as shown in Table 2-9.

TABLE 2-9 DAX operators

Type	Operator	Meaning	Example	Result
Arithmetic	+	Addition	2 + 3	5
	-	Subtraction or sign	2 - 3	-1
	*	Multiplication	2 * 3	6
	/	Division	3 / 2	1.5
	^	Exponentiation	2 ^ 3	8
Comparison	=	Equal to	2 = 3	FALSE
	>	Greater than	2 > 3	FALSE
	<	Less than	2 < 3	TRUE
	>=	Greater than or equal to	2 >= 3	FALSE
	<=	Less than or equal to	2 <= 3	TRUE
	<>	Not equal to	2 <> 3	TRUE
Text concatenation	&	Concatenates two text values	“2” & “3”	23
Logical	&&	AND condition between two Boolean expressions	(2 = 3) && (1 = 1)	FALSE
	||	OR condition between two Boolean expressions	(2 = 3) || (1 = 1)	TRUE
	IN	Belonging in a list	2 IN { 1, 2, 3 }	TRUE
	NOT	Negation	NOT 2 = 3	TRUE

Some logical operators are also available as functions. Instead of the double ampersand, you can use the AND function:

```
AND ( 2 = 3, 1 = 1 )
```

Instead of a double pipe, you can use the OR function:

```
OR ( 2 = 3, 1 = 1 )
```

Both functions, AND and OR, take exactly two arguments. If you need to evaluate more than two conditions, you can nest your functions:

```
AND ( 2 = 3, AND ( 1 = 1, 5 = 5 ) )
```

The NOT operator can be used as a function as well:

```
NOT ( 2 = 3 )
```

> ***MORE INFO* DAX OPERATOR REFERENCE**
>
> For more examples and details on DAX operators, including operator precedence, see "DAX Operator Reference" at: *https://msdn.microsoft.com/en-us/library/ee634237.aspx*.

Using DAX functions in calculated columns

DAX has more than 200 functions. Some functions return scalar values, while others return tables. If a function results in a one-column one-row table, it can be implicitly converted to a scalar value.

There are many functions that perform the same tasks as some M functions. For example, the LOWER, UPPER, LEN, and TRIM functions transform text values in the same way as the M Text.Lower, Text.Upper, Text.Length, and Text.Trim functions, respectively.

Unlike M functions, DAX functions can perform implicit type conversion. For instance, in M, the following expression results in the error shown in Figure 2.11:

```
Text.Length ( 100 )
```

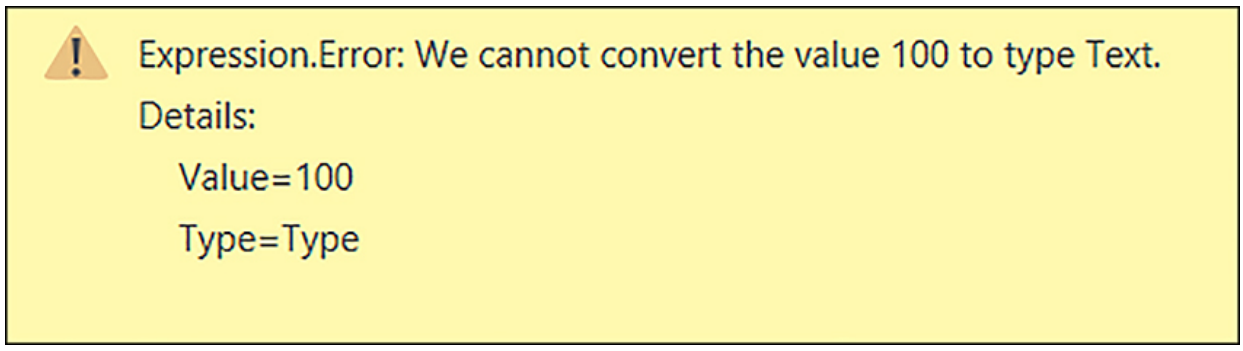


FIGURE 2.11 Error message

In DAX, on the other hand, `LEN ( 100 )` returns 3. Using LEN on non-text values, however, is somewhat unpredictable, and it should be combined with the FORMAT function. For example, if a column 'Date'[Date] contains a date value of 1 January 2018, then a corresponding value in the following calculated column will result in 8:

```
DAX Length = LEN ( 'Date'[Date] )
```

However, if you format values explicitly inside the LEN function, then it is possible to control the results. The following calculated column returns 10:

Click here to view code image

```
DAX Formatted Length = LEN ( FORMAT ( 'Date'[Date], "dd-MM-yyyy"
) )
```

The LEN function, as well as FIND or SEARCH, can be useful when you want to extract substrings of a variable length. For instance, in the Customer table, there is a column called Buying Group, which has three distinct values:

- N/A
- Tailspin Toys
- Wingtip Toys

Let's say you want to extract the first word only, so you are looking to create a column with the following three values:

- N/A
- Tailspin
- Wingtip

Note that each word has a different length. If the number of characters you wanted to extract were fixed, you could use the LEFT function, which gives you the first N characters. This function, along with RIGHT, MID, and LEN, also exists in Excel. To create a calculated column with the first three characters from Buying Group, you would write the following formula:

Click here to view code image

```
Buying Group First Three Characters = LEFT ( Customer[Buying
Group], 3 )
```

To extract the first word, first, calculate the length of the first word. For this, you need to find the position of the space symbol in a string. In this case, use the FIND or SEARCH functions. Both functions have two required arguments: text to find and where to search. The only difference between them is that the FIND is case-sensitive, while SEARCH is not. Because we are looking for a space symbol, we can use either function. We can first try the following formula:

Click here to view code image

```
Buying Group First Space Position = FIND ( " ", Customer[Buying
Group] )
```

Because there is no space in "N/A," we get an error that propagates to the entire column, even though there is only one row in which the space was not found. You can see the error in Figure 2.12.

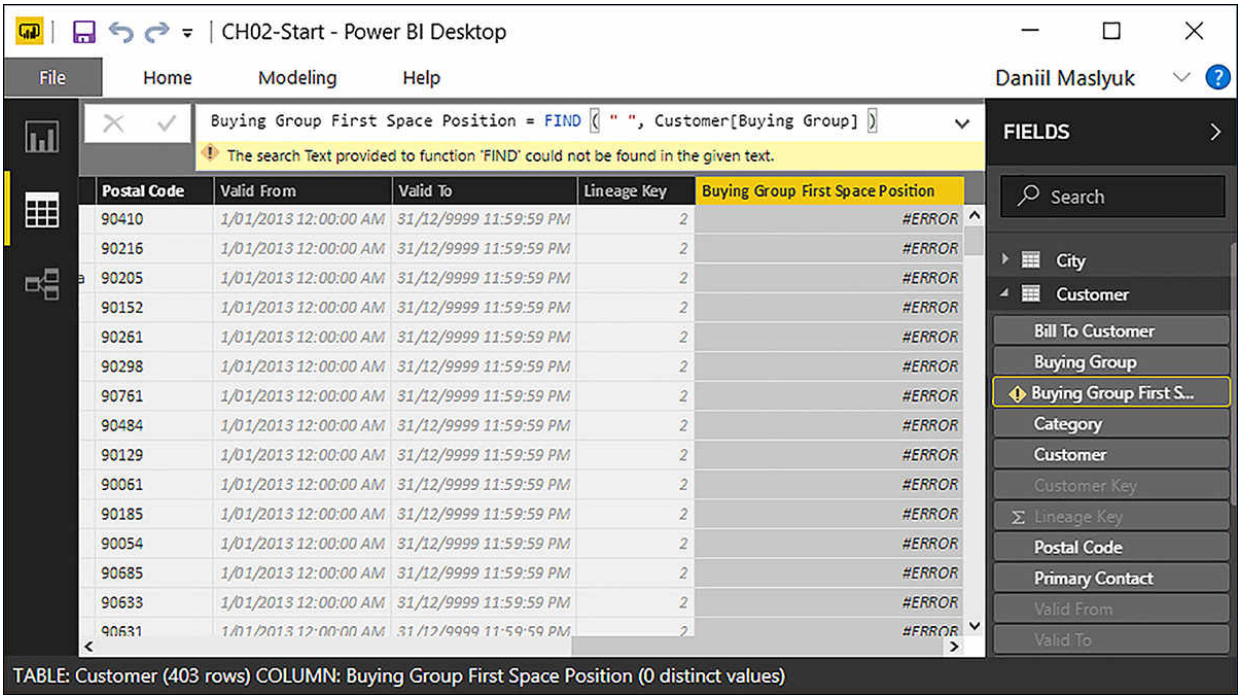


FIGURE 2.12 DAX error message in the whole column

This behavior is typical for DAX calculated columns. One way to solve this problem is to use the optional parameters in FIND. The third parameter specifies the number of character to start the search from; if omitted, it is 1. The fourth parameter specifies the value to return in case nothing is found. For example, return 0 if nothing is found:

Click here to view code image

```
Buying Group First Space Position No Error = FIND ( " ",
Customer[Buying Group], , 0 )
```

In this case, you get no error. An alternative way to solve the same problem is to use the IFERROR function, which takes two arguments: an expression to evaluate and a value to return in case of an error. The following calculated column returns the same result as the previous one:

Click here to view code image

```
Buying Group First Space Position No Error = IFERROR ( FIND ( "
", Customer[Buying Group] ), 0 )
```

To extract the first word from Buying Group, use the following formula:

Click here to view code image

```
Buying Group First Word = IFERROR ( LEFT ( Customer[Buying
Group], FIND ( " ", Customer[Buying Group] ) - 1 ),
Customer[Buying Group] )
```

There are two things to note about this formula. First, subtract 1 from the result of the FIND, because DAX starts counting from 1, which is different from M. Second, this formula is quite long and could benefit from formatting to make the code easier to read. There is a tool by SQLBI called DAX Formatter, which helps you to make your code cleaner and easier to read.

NOTE DAX FORMATTER

It is a good practice to format your code. The DAX Formatter tool, which is updated regularly to work with the newest functions, can be found at *http://www.daxformatter.com/*.

The set of rules according to which DAX Formatter adheres, can be found at *https://www.sqlbi.com/articles/rules-for-dax-code-formatting/*.

The following code has been formatted with DAX Formatter:

Click here to view code image

```
Buying Group First Word =
IFERROR (
    LEFT ( Customer[Buying Group], FIND ( “ “,
```

```
Customer[Buying Group] ) - 1 ),
    Customer[Buying Group]
)
```

The LEN function can also be useful when you want to calculate how many times a text string appears in another text string. For this, use the SUBSTITUTE function.

The SUBSTITUTE function, which is case-sensitive, has three required parameters: text, old text, and new text. For instance, replace all a's with o's in "Alabama." As a result, you get "Alobomo":

```
Alobomo = SUBSTITUTE ( "Alabama", "a", "o" )
```

Because SUBSTITUTE is case-sensitive, the first A is not affected. To count the number of times a character appears in a string, substitute the character with an empty string and calculate the difference in lengths of the old and the new strings. The following expression counts the number of times the capital letter "T" appears in the Buying Group column values:

Click here to view code image

```
Number of T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( Customer[Buying Group], "T", "" ) )
```

To count the number of times the letter "t" appeared regardless of case, you can either use another SUBSTITUTE or use LOWER or UPPER. The following three formulas provide identical results, and it shows that in DAX there is often more than one way to solve the same problem:

Click here to view code image

```
// Using second SUBSTITUTE
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( SUBSTITUTE ( Customer[Buying Group],
"t", "" ), "T", "" ) )

// Using LOWER
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( LOWER ( Customer[Buying Group] ), "t",
"" ) )
```

```
// Using UPPER
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( UPPER ( Customer[Buying Group] ), "T",
"" ) )
```

The Number of T's and Number of all T's columns provide the following results, shown in Table 2-10.

TABLE 2-10 Comparison of the Number of T's and Number of all T's columns

Buying Group	Number of T's	Number of all T's
N/A	0	0
Tailspin Toys	2	2
Wingtip Toys	1	2

***MORE INFO* TEXT DAX FUNCTIONS**

For more information on the available text functions in DAX, see "Text Functions (DAX)" at: *https://msdn.microsoft.com/en-us/library/ee634938.aspx*.

DAX has several mathematical functions available, many of which are similar to the Excel functions with which they share their names. In the following list, you can see how some of the most common mathematical DAX functions work.

- **ABS**(Number) Returns the absolute value of a number.
- **DIVIDE**(Numerator, Denominator, AlternateResult) Safe division function that can handle division by zero.
- **EXP**(Number) Returns *e* raised to the power of a number.
- **EVEN**(Number) Returns a number rounded up to the nearest even number. You can check if a number is even using the ISEVEN function.
- **ODD**(Number) Returns a number rounded up to the nearest odd number. You can check if a number is odd using the ISODD function.
- **FACT**(Number) Returns the factorial of a number.

- **LN**(Number) Returns the natural logarithm of a number.
- **LOG**(Number, Base) Returns the logarithm of a number to the base you specify.
- **MOD**(Number, Divisor) Returns the remained of a number divided by a divisor.
- **PI**() Returns the number Pi, accurate to 15 digits.
- **POWER**(Number, Power) Returns the result of a number raised to a power. This is the function equivalent of the exponentiation (^) operator.
- **QUOTIENT**(Numerator, Denominator) Returns the integer portion of a division.
- **SIGN**(Number) Returns -1 if a number is negative, 1 if it is positive, and 0 if it is zero.
- **ROUNDDOWN**(Number, NumberOfDigits) Rounds a number towards zero to a specified number of decimal places.
- **FLOOR**(Number, Significance) Rounds a number toward zero to the nearest multiple of significance.
- **TRUNC**(Number, NumberOfDigits) Truncates a number, keeping the specified number of decimal places.
- **ROUND**(Number, NumberOfDigits) Rounds a number to a specified number of decimal places.
- **MROUND**(Number, Multiple) Rounds a number to the nearest multiple.
- **ROUNDUP**(Number, NumberOfDigits) Rounds a number towards zero to a specified number of decimal places.
- **CEILING**(Number, Significance) Rounds a number towards zero to the nearest multiple of significance.
- **INT**(Number) Rounds a number down to the nearest integer.
- **RAND**() Returns a random number greater than or equal to 0 and less than 1.
- **RANDBETWEEN**(Bottom, Top) Returns a random integer between two specified numbers.
- **SQRT**(Number) Returns the square root of a number.

***MORE INFO* MATHEMATICAL DAX FUNCTIONS**

For more information on the available mathematical and trigonometric functions in DAX, see "Math and Trig Functions (DAX)" at: *https://msdn.microsoft.com/en-us/library/ee634241.aspx*.

The date and time functions in DAX help you create calculations based on dates and time. The following list shows some of the most common date and time functions.

- **TODAY**() Returns the current system date in datetime format.
- **NOW**() Returns the current system date and time in datetime format.
- **DATE**(Year, Month, Day) Returns the specified date in datetime format.
- **DATEVALUE**(TextDate) Converts a text date to a date in datetime format.
- **YEAR**(Date) Returns the year portion of a date.
- **MONTH**(Date) Returns the month number of a date.
- **DAY**(Date) Returns the day number of a date.
- **TIME**(Hour, Minute, Second) Returns the specified time in datetime format.
- **TIMEVALUE**(TextTime) Converts a text time to time in datetime format.
- **HOUR**(Datetime) Returns the hour of a datetime.
- **MINUTE**(Datetime) Returns the minute of a datetime.
- **SECOND**(Datetime) Returns the second of a datetime.
- **DATEDIFF**(StartDate, EndDate, Interval) Returns the number of intervals between two dates. The interval can be any of the following: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR.
- **EDATE**(Date, Months) Shifts a date back or forward by a specified number of months.
- **EOMONTH**(Date, Months) Returns the end of month date of a specified date, shifted by a specified number of months.

- **WEEKDAY**(Date, ReturnType) Returns the number of the day of the week according to the specified ReturnType.
- **WEEKNUM**(Date, ReturnType) Returns the week number in the year according to the specified ReturnType.

> ***MORE INFO* DATE AND TIME DAX FUNCTIONS**
>
> For more information on the available date and time functions in DAX, see "Date and Time Functions (DAX)" at: *https://msdn.microsoft.com/en-us/library/ee634786.aspx*.

Using LOOKUPVALUE

If there is a many-to-one relationship between two tables, you can bring a related value from the one side to the many side with RELATED, as discussed earlier in the chapter. With LOOKUPVALUE, it is possible to look up values from another table based on one or more conditions. This is especially useful when there are two or more conditions to look up by because DAX allows creating physical relationships based on one column only.

LOOKUPVALUE uses the following syntax: column to retrieve values from, followed by pairs of arguments, in which the first item is a column to search, and the second item is a scalar expression to look for. If there is no match, a blank value is returned. In case there are multiple values that match the same condition, an error is returned.

For review purposes, use the following example. In the Sale table, you can insert values from the Target table using the following formula.

Click here to view code image

```
Target Quantity =
LOOKUPVALUE (
    Target[Target Quantity],
    Target[Calendar Year], RELATED ( 'Date'[Calendar Year] ),
    Target[Bill To Customer], RELATED ( Customer[Bill To
Customer] )
)
```

Note that this particular calculated column does not provide useful results because the granularity of the Sale and Target tables are different, but LOOKUPVALUE can nonetheless be useful in situations where the

granularity of tables is the same or does not matter. Also note that the following formula, that is not specific enough, does not work and results in an error shown in Figure 2.13.

Click here to view code image

```
Target Quantity =
LOOKUPVALUE (
    Target[Target Quantity],
    Target[Calendar Year], RELATED ( 'Date'[Calendar Year] )
)
```

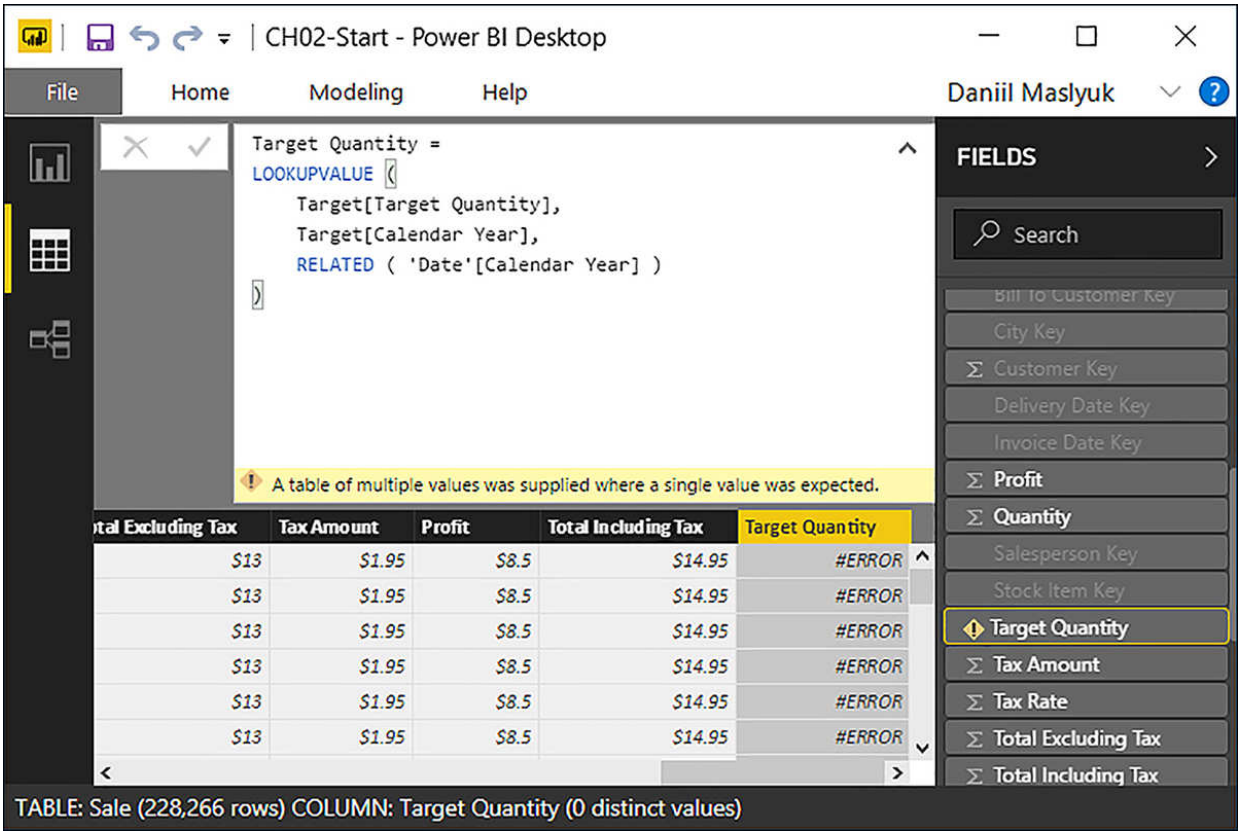


FIGURE 2.13 LOOKUPVALUE that results in error

MORE INFO LOOKING UP VALUES IN DAX

There are other techniques for looking up values in DAX that you may want to consider. For a comprehensive discussion, see an article by Marco Russo, "Lookup multiple values in DAX" at: *https://www.sqlbi.com/articles/lookup-multiple-values-in-dax/*.

Grouping values

Calculated columns can be very useful for grouping values. For instance, if you have a column with unit prices, such as ‘Stock Item’[Unit Price], you may want to put them into Low, Medium, and High categories based on business requirements. The column contains 58 distinct values. Let’s say that items under $100 can be placed in the “Low” category, items from $100 to $1,000 can be placed into the “Medium” category, and the rest can be placed into the “High” category.

One way to do it would be by using the IF function. IF, like many other DAX functions, can be nested. The following calculated column produces the necessary grouping:

Click here to view code image

```
Price Category =
IF (
    'Stock Item'[Unit Price] < 100,
    "Low",
    IF (
        'Stock Item'[Unit Price] < 1000,
        "Medium",
        "High"
    )
)
```

If you use this calculated column in a visual, you will notice that the values are sorted in alphabetic order: High, Low, Medium. To solve this problem, try the following code:

Click here to view code image

```
Price Category Number =
IF (
    'Stock Item'[Price Category] = "Low",
    1,
    IF (
        'Stock Item'[Price Category] = "Medium",
        2,
        3
    )
)
```

An alternative way to produce the same column is to use the SWITCH function. The first parameter in this function is an expression to be evaluated

multiple times. The other parameters come in pairs: a value to evaluate against the expression, and a result to return in case the value and the expression match. The last argument, which is optional, is the result to return in case no value matched the expression.

The following formula returns the same results as the previous formula:

Click here to view code image

```
Price Category Number = SWITCH ( 'Stock Item'[Price Category],
"Low", 1, "Medium", 2, 3 )
```

Once you create the column using either approach and try to sort the Price Category column by Price Category Number, you will get the error shown in Figure 2.14.

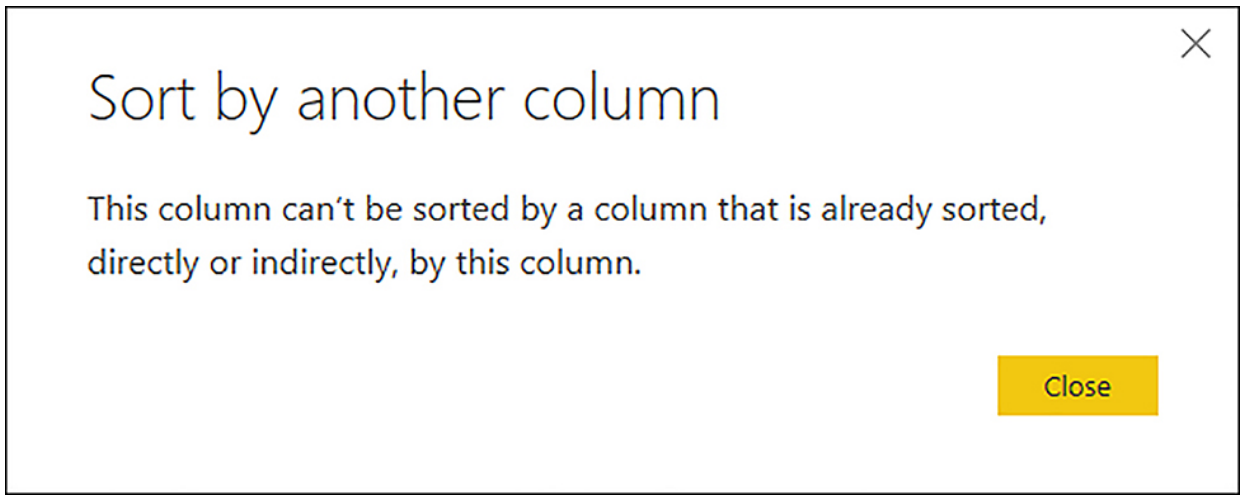


FIGURE 2.14 Sort by another column error

This error message appears because you are trying to sort the Price Category column by a column that derives its values from Price Category. Note that you can sort Price Category Number by Price Category. This problem can be fixed in at least two ways. One way is to create the two calculated columns in reverse order.

Click here to view code image

```
// First, create Price Category Number
Price Category Number =
IF (
    'Stock Item'[Unit Price] < 100,
    1,
    IF (
```

```
        'Stock Item'[Unit Price] < 1000,
        2,
        3
    )
)

// Second, create Price Category
Price Category = SWITCH ( 'Stock Item'[Price Category Number],
1, "Low", 2, "Medium", "High" )
```

Now the Price Category column can be sorted by the Price Category Number column without any errors. An alternative approach is to write the two calculated columns in the same way so that both of them reference the Unit Price column without referencing each other:

Click here to view code image

```
// The order of creation does not matter in this case
Price Category =
IF (
    'Stock Item'[Unit Price] < 100,
    "Low",
    IF (
        'Stock Item'[Unit Price] < 1000,
        "Medium",
        "High"
    )
)

Price Category Number =
IF (
    'Stock Item'[Unit Price] < 100,
    1,
    IF (
        'Stock Item'[Unit Price] < 1000,
        2,
        3
    )
)
```

In this way, you can also sort the Price Category column by the Price Category Number column without any problem.

The SWITCH function is especially useful when there are many conditions to check. For instance, if you decide to group unit prices into five categories, use four IF statements:

Click here to view code image

```
Five Price Categories =
IF (
    'Stock Item'[Unit Price] < 10,
    "Very Low",
    IF (
        'Stock Item'[Unit Price] < 100,
        "Low",
        IF (
            'Stock Item'[Unit Price] < 200,
            "Medium",
            IF (
                'Stock Item'[Unit Price] < 1000,
                "High",
                "Very High"
            )
        )
    )
)
```

With SWITCH, you can use the SWITCH TRUE pattern to check Boolean statements:

Click here to view code image

```
Five Price Categories SWITCH =
SWITCH (
    TRUE (),
    'Stock Item'[Unit Price] < 10, "Very Low",
    'Stock Item'[Unit Price] < 100, "Low",
    'Stock Item'[Unit Price] < 200, "Medium",
    'Stock Item'[Unit Price] < 1000, "High",
    "Very High"
)
```

You can also group values with the user interface. To group the Unit Price values, select the column in the Fields pane, then select **Modeling** > **Groups** > **New Group**. The Groups window, shown in Figure 2.15, will then open, where you can specify the settings for grouping, also called binning.

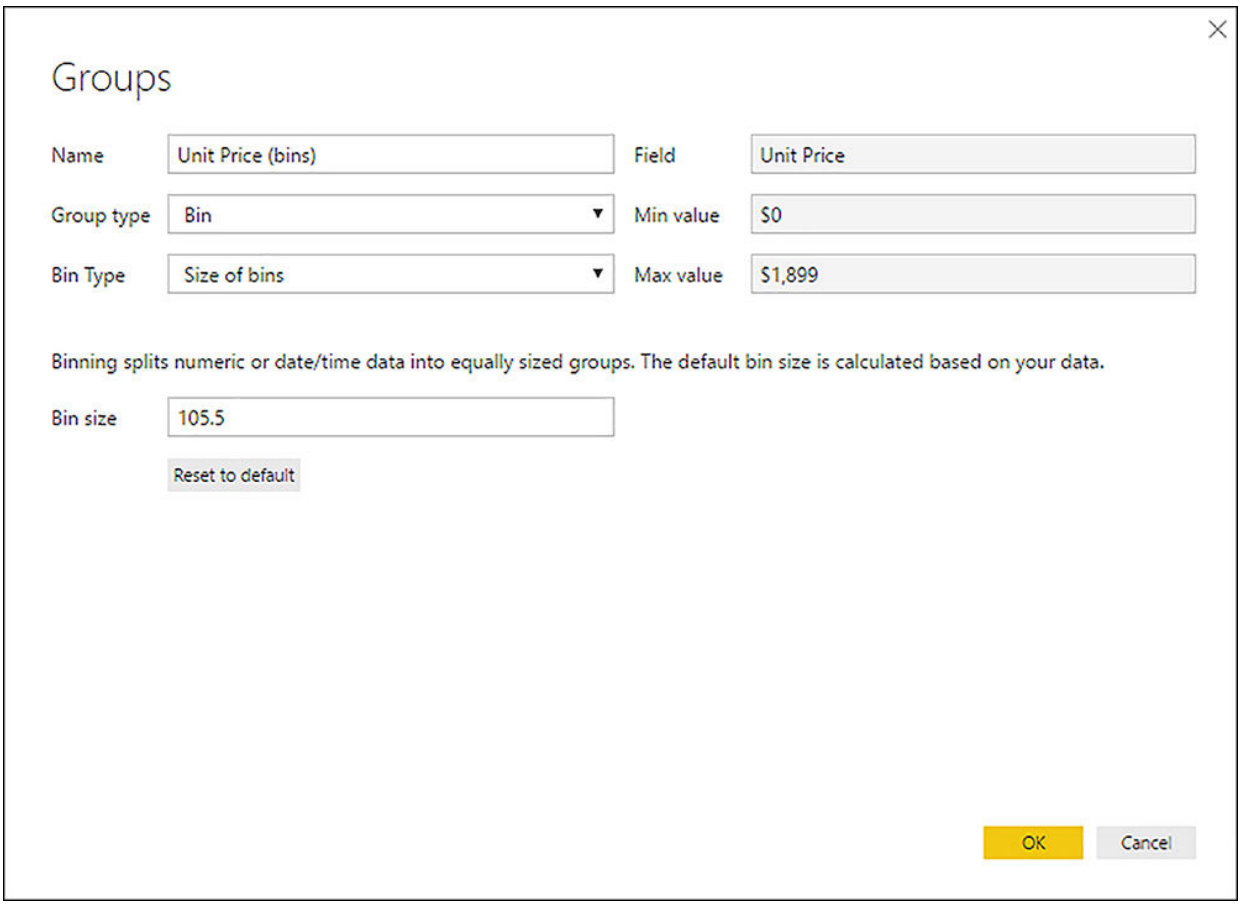


Figure 2.15 The Groups window with Bin selected as group type

Power BI analyzes the values in the column and chooses the group type that it deems best for the selected column. Because we have numeric values in Unit Price, Power BI decided that it's best to group the values into bins. Power BI gave the new column a default name, Unit Price (bins), which can be changed.

There are two bin types available: Size of Bins and Number of Bins. For size of bins, Power BI determines the best bin size, which can be adjusted. In our case, Power BI deemed 105.5 to be the best bin size. If necessary, you can reset the bin size to its default. If you click OK with default settings, Power BI will create a calculated column in which unit prices are rounded down to the nearest multiple of 105.5.

In this specific case, the new column will contain five distinct values:

- $0
- $105.50

- $211
- $316.50
- $1,899

Internally, Power BI uses the following formula:

Click here to view code image

```
Unit Price (bins) =
IF (
    ISBLANK ( 'Stock Item'[Unit Price] ),
    BLANK (),
    IF (
        'Stock Item'[Unit Price] >= 0,
        ROUNDDOWN ( 'Stock Item'[Unit Price] / 105.5, 0 ) *
105.5,
        ROUNDUP ( 'Stock Item'[Unit Price] / 105.5, 0 ) * 105.5
    )
)
```

Note that this calculated column has a special icon next to it: two overlapping squares. This is different from the icons shown with other calculated columns that we have created so far. You can see the Unit Price (bins) icon in Figure 2.16.

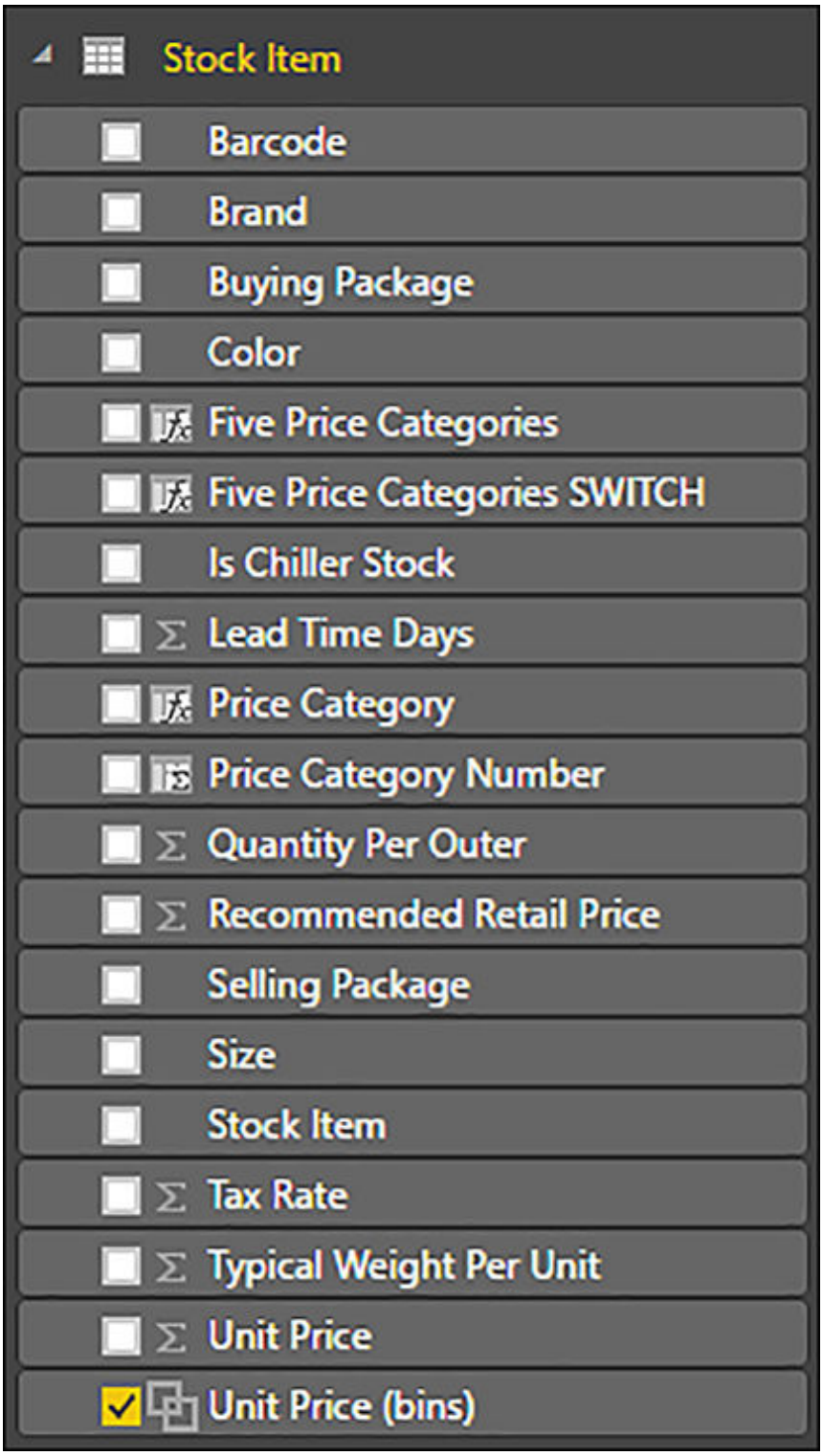


FIGURE 2.16 The Unit Price (bins) column in the fields list

The other calculated columns have either of two icons:

- When a column is of a numeric data type, such as the Price Category Number column, it has a calculated column icon with a capital sigma.
- When a column is of other data type text, like the Price Category column, it has a calculated column icon with *fx* written on it.

Even though technically Unit Price (bins) is a calculated column, it is not possible to see and modify its formula with Power BI Desktop. Instead, you can edit the group by selecting the column and selecting **Modeling** > **Groups** > **Edit Groups**. The Groups window will then open where you make changes, but you will not be able to change the group type.

If you choose to change the Bin Type to Number of Bins, you will need to specify the number of bins, which, for the Unit Price column, Power BI set to 18 by default. When you edit the Bin Count field, Power BI will show you the approximate bin size; for the default 18 bins, the bin size is 105.5. You can see the Groups dialog box in Figure 2.17.

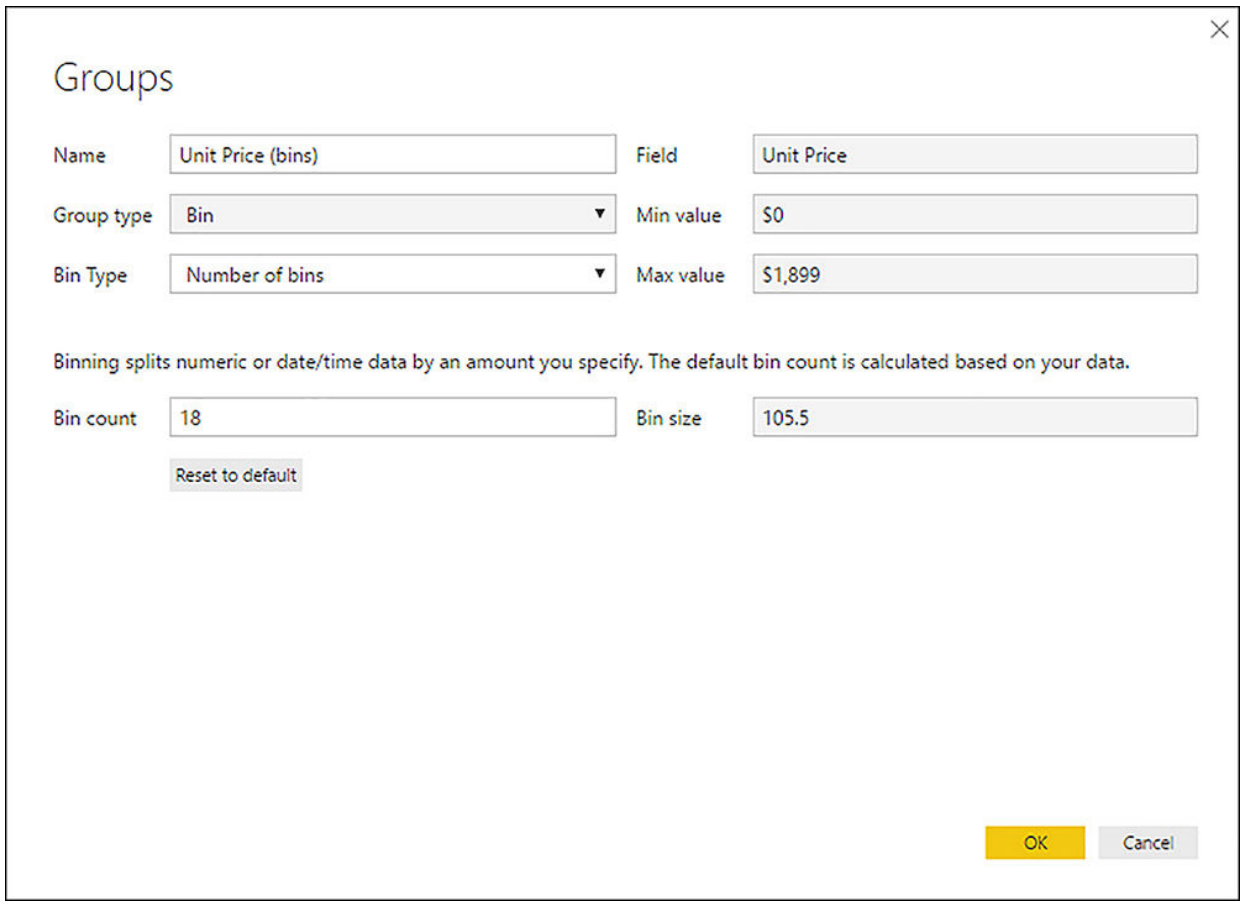


Figure 2.17 Groups window with Number of Bins selected as Bin Type

Grouping Unit Price by number of bins results in the following five distinct values, which are different from groups based on bin size:

- $0
- $105.50
- $211
- $316.50
- $1,793.50

When you create a new group, you can also choose the List group type. If you choose List in the Group type drop-down list, the Groups window interface will change, allowing you to pick values and group them. The Groups dialog box with List group type selected can be seen in Figure 2.18.

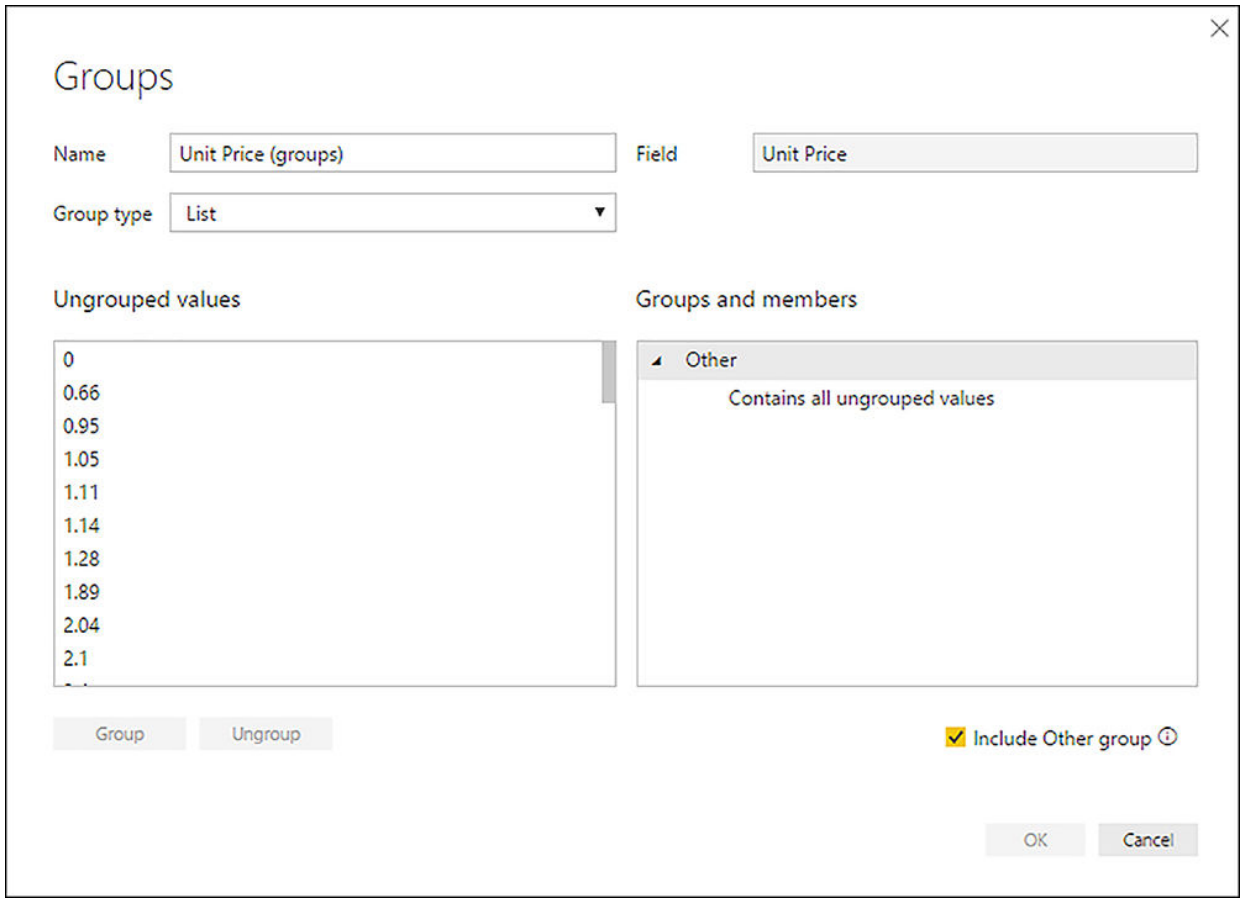


FIGURE 2.18 Groups window with List group type

In this dialog box, you can either group items individually, resulting in single-item groups, or you can select multiple values at once by holding the

Ctrl key. Holding the Shift key allows you to select a range of values. Once you select some values and click the Group button, the values are transferred to the list on the right, Groups and members, and you can specify a new name for each group. By default, every group is given a name that is the list of values separated by ampersands. Below the Groups and members list, the Include Other group check box enables you to create the Other group, which contains all ungrouped values. If you choose not to include the Other group, ungrouped values will be left as-is.

> ***MORE INFO* GROUPING AND BINNING IN POWER BI DESKTOP**
>
> For more information and examples on the grouping and binning feature in Power BI Desktop, see "Use grouping and binning in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-grouping-and-binning*.

Using variables in calculated columns

By using variables in DAX, you can write more readable and concise code. For example, let's say you want to define a discounted unit price that includes tax, but if the new price is less than $20, then you keep the old price, including tax. The discount rate is 20 percent. One way to write this formula is as follows.

Click here to view code image

```
New Price =
IF (
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 )
        * 0.8
        < 20,
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 ),
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 )
        * 0.8
)
```

Note that the formula contains repeated code. If we rewrite the formula with variables, it will become easier to read. The general pattern for using

variables is as follows:

Click here to view code image

```
Measure name =
VAR FirstVariable = … // DAX expression
VAR NthVariable = … // DAX expression
RETURN = … // DAX expression
```

The names of variables cannot include spaces or be reserved keywords, such as Measure. You can reference previously declared variables inside variables. The variables can only be accessed in the expression they are defined in; you cannot access variables defined in column A from column B, for example.

If we rewrite the New Price formula with variables, it may look as follows:

Click here to view code image

```
New Price =
VAR TaxPct = Sale[Tax Rate] / 100
VAR PriceInclTax = Sale[Unit Price] * ( 1 + TaxPct )
VAR DiscountedPriceInclTax = PriceInclTax * 0.8
RETURN
    IF (
        DiscountedPriceInclTax < 20,
        PriceInclTax,
        DiscountedPriceInclTax
    )
```

***MORE INFO* DAX VARIABLES**

For more examples and information on using variables in DAX, see the official documentation articles: "VAR (DAX)" at: *https://msdn.microsoft.com/en-us/library/mt243785.aspx*, and "Using variables in DAX expressions" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/introductiontodax#step-4*.

See also an article by Matt Allington, "Using Variables in DAX" at *https://exceleratorbi.com.au/using-variables-dax/*.

Another way to solve the problem of repetitive code in calculated columns is to split intermediate calculations in different columns. For instance, you

can instead create four calculated columns as follows.

LISTING 2-8 Calculated columns

Click here to view code image

```
TaxPct = Sale[Tax Rate] / 100

PriceInclTax = Sale[Unit Price] * ( 1 + Sale[TaxPct] )

DiscountedPriceInclTax = Sale[PriceInclTax] * 0.8

New Price =
IF (
    Sale[DiscountedPriceInclTax] < 20,
    Sale[PriceInclTax],
    Sale[DiscountedPriceInclTax]
)
```

While this approach results in the same values as the previous New Price formulas, it has an important drawback: increased data model size. Every calculated column is materialized, resulting in greater file size. Because Power BI uses VertiPaq, an in-memory engine, calculated columns also occupy RAM space. The size of a calculated column depends on the number of its distinct values, so some columns may take more space than others.

Evaluation context

Most functions that we have reviewed so far were executed in so-called row context. Row context can be thought of as the current row. When you define a calculated column, its formula is evaluated for each row of a table to which the column is added. Some table and aggregation functions, such as FILTER and SUMX, iterate over tables and, as a result, also use row context.

Some functions ignore row context, and they use filter context instead. Filter context can be thought of as all the filters that are applied to a calculation. Filters can come from several places: visual-, page-, and report-level filters; axes in a visual; slicers; rows and columns in the Matrix visual; and more. For example, when you create a bar chart with Calendar Year Label on the axis and Profit in values, like in Figure 2.19, then each Calendar Year Label provides filter context for its Profit values, and as a

result, each bar is different. Filters can also be applied programmatically with DAX.

IMPORTANT SORT BY COLUMN AND FILTER CONTEXT

Columns used for sorting are part of filter context. For example, if you show Profit by Month, and you sort Month by Month Number, then Month Number is also part of filter context.

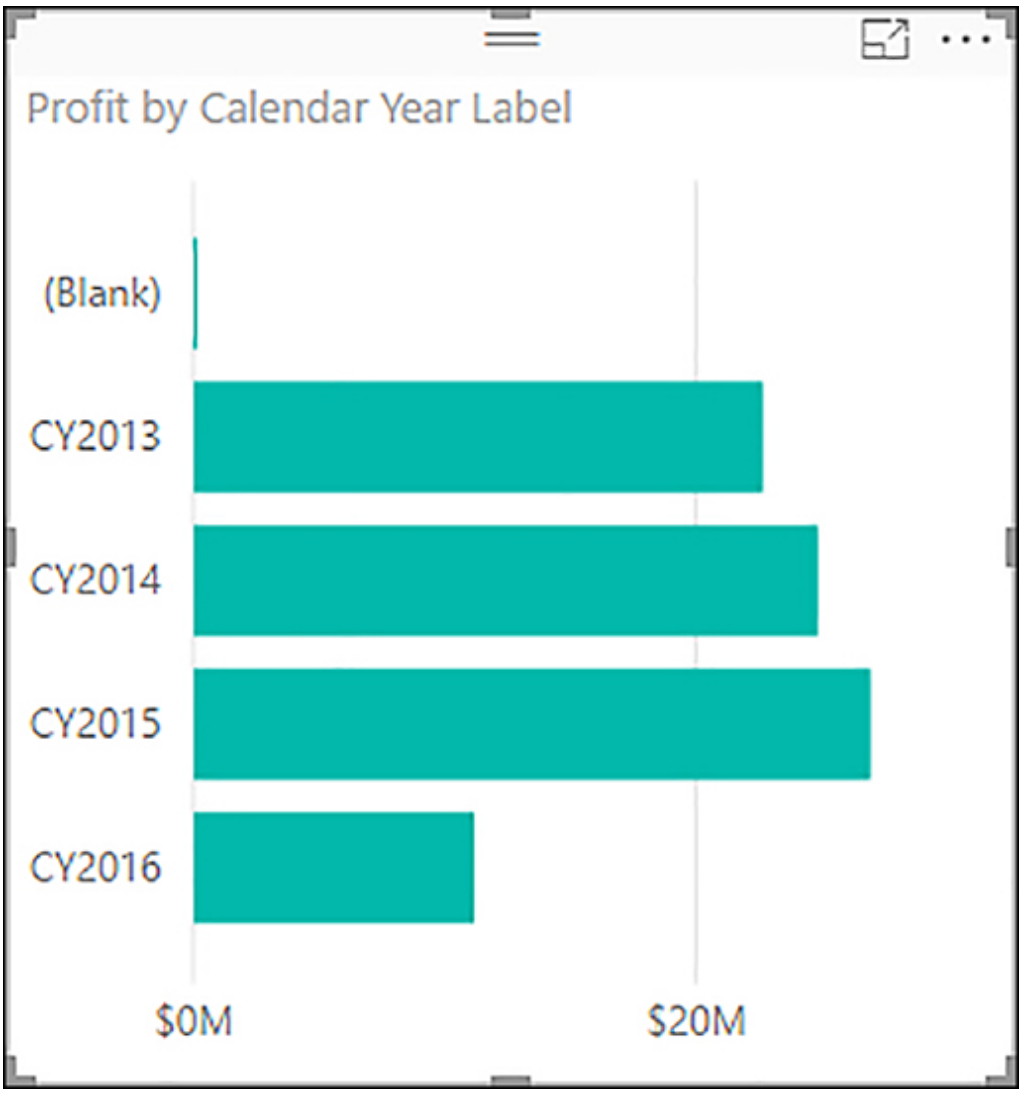


FIGURE 2.19 Bar chart showing Profit by Calendar Year Label

The two contexts always co-exist at the same time, though either of them can be empty at certain times. For instance, when you define a calculated

column in a physical table with a formula that uses no functions, the filter context is empty.

Functions that ignore row context include, but not limited to, SUM, AVERAGE, and COUNTROWS. To review the effect of a using a formula that ignores the row context, we can create a calculated column in the Scale table with the following formula:

```
Sum of Scale = SUM ( Scale[Scale] )
```

The resulting column can be seen in Figure 2.20.

FIGURE 2.20 Sum of Scale calculated column

Note that the new column has the same value for each row because SUM works in filter context and ignores row context. In other words, SUM calculates its values irrespective of what the current row's value in the Scale[Scale] column is.

By default, row context ignores any relationships that are in place unless you use the RELATED or RELATEDTABLE functions. You can only leverage the effect of relationships between tables using filter context. It is possible to transform row context into equivalent filter context. For this purpose, we can use the CALCULATE function, which has one required parameter: an expression that works in filter context. This function can also take optional filter arguments, which we are going to review later in this chapter. At this time, we are going to focus on context transition capability of CALCULATE. To see the effect of context transition, create the following calculated column:

```
Sum of Scale Calculate = CALCULATE ( SUM ( Scale[Scale] ) )
```

Note that writing `CALCULATE ( Scale[Scale] )` results in an error because `Scale[Scale]` can only work in row context. The reason why you cannot use `Scale[Scale]` in filter context is that DAX does not know what you want to do with values in the column. Do you want to sum the value, take an average of them, or something else? The Sum of Scale Calculate column is shown in Figure 2.21.

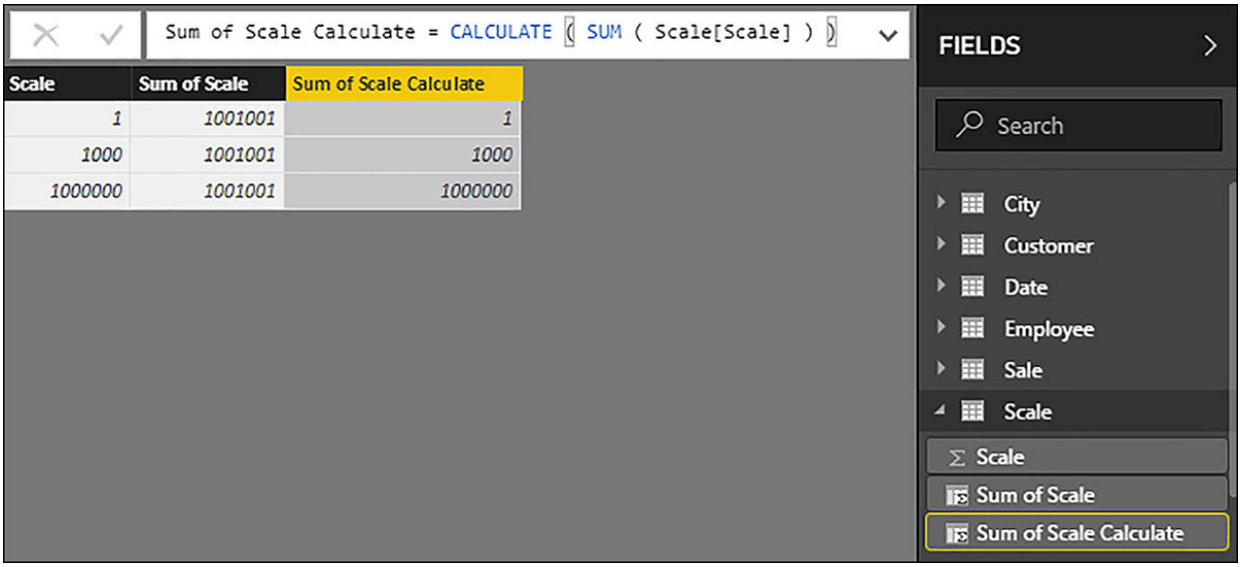


FIGURE 2.21 Sum of Scale Calculate column

When context transition happens, the row context is transformed into equivalent filter context. This means that for each row, a table is filtered to contain only those rows where the values are the same as in the current row. For example, when we define the Sum of Scale Calculate calculated column with CALCULATE and context transition happens, in the first row the following filter context is applied: Scale[Scale] = 1 and Scale[Sum of Scale] = 1001001. For the Scale rows that remain after filtering, SUM (Scale[Scale]) is performed. Therefore, every row contains a different value in the Sum of Scale Calculate column.

***MORE INFO* CONTEXT TRANSITION**

For more examples and details on context transition in DAX, see an article by Alberto Ferrari: "Understanding Context Transition" at: *https://www.sqlbi.com/articles/understanding-context-transition/*.

It's important to remember that the current row and the equivalent filter context are not the same thing. To review an example that highlights the difference, add a duplicate row to the Scale table. To do that, follow these steps:

1. Right-click on the Scale table in the Fields list and select Edit Query.
2. In Power Query Editor, click on the gear icon next to the Source step.
3. Type **1** in the fourth row and click OK.
4. Click Close & Apply.

You can see the results in Figure 2.22.

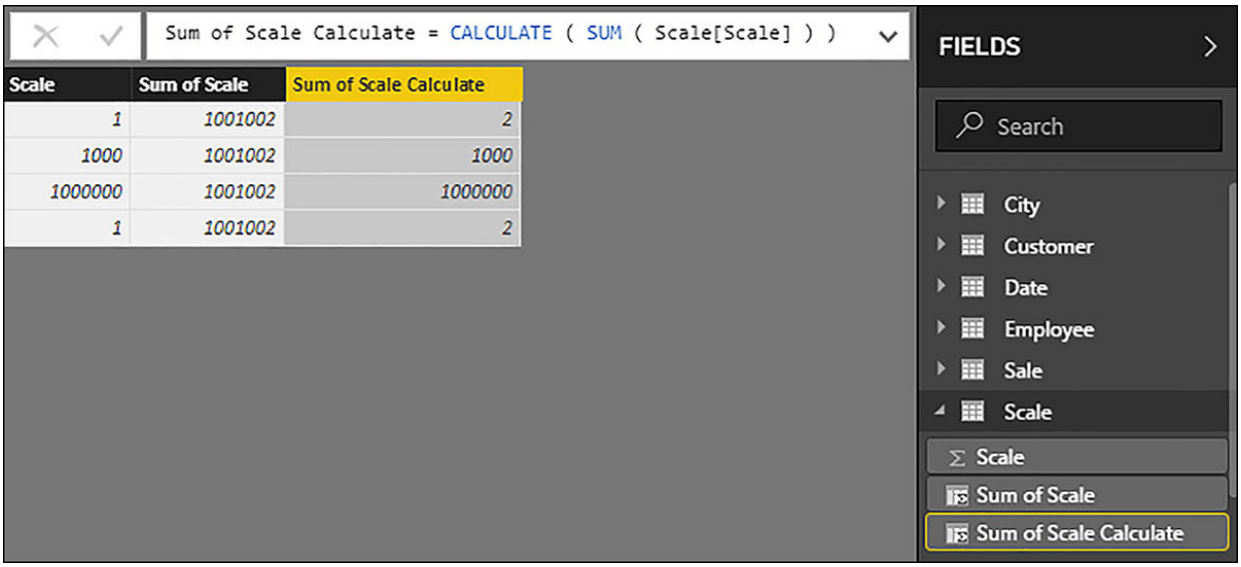


FIGURE 2.22 Resulting Sum of Scale Calculate values with duplicate rows

Note that the rows in which Scale = 1 has the Sum of Scale Calculate value of 2. For these rows, here are the steps that DAX followed to arrive at these figures:

1. Identify the row context: Scale = 1, Sum of Scale = 1001002.
2. Convert the row context into the equivalent filter context: filter the Scale table and keep only those rows where Scale = 1 and Sum of Scale = 1001002.
3. For the two rows that remain, sum the values in the Scale column: 1 + 1 = 2.
4. Return the result of the summation, which is 2.

Because there are two identical rows, DAX follows the same procedure twice. The filter context cannot distinguish between identical rows, and as a result, the Sum of Scale Calculate values are not the same as the Scale column values.

Earlier in this chapter, we defined the following two columns in the Date table:

Click here to view code image

```
// Returns different values for each row
Sales # = COUNTROWS ( RELATEDTABLE ( Sale ) )

// Returns different values only with bidirectional filtering
enabled
Cities # = COUNTROWS ( RELATEDTABLE ( City ) )
```

The reason why these calculated columns returned different values is that RELATEDTABLE is an alias for CALCULATETABLE, a sister function of CALCULATE, which works similarly, but receives a table expression as the first parameter instead of a scalar expression. Therefore, for each row in the Date table, context transition occurred, which filtered the Sale and City tables to only those rows that were related to the current row.

***MORE INFO* DAX EVALUATION CONTEXTS**

For a general overview of evaluation contexts in DAX, see an article in the official documentation, "DAX basics in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-quickstart-learn-dax-basics*.

Full details on evaluation contexts in DAX are outside of the scope of this book, but it's important to understand this topic well to write correct DAX formulas. For more details on the topic, see a sample chapter from "The Definitive Guide to DAX: Business intelligence with Microsoft Excel, SQL Server Analysis Services, and Power BI," a book by Alberto Ferrari and Marco Russo, "Understanding Evaluation Contexts in DAX" at *https://www.microsoftpressstore.com/articles/article.aspx?p=2449191*.

Circular dependencies in calculated columns

DAX evaluates every expression and does so in the order that respects every dependency of one expression on another. To understand circular dependencies, first review with the following calculated column in the Stock Item table:

Click here to view code image

```
Profit $ = 'Stock Item'[Recommended Retail Price] - 'Stock
Item'[Unit Price]

Profit % = DIVIDE ( 'Stock Item'[Profit $], 'Stock Item'[Unit
Price] )
```

Note that the Profit % column references and depends on the Profit $ column. At this point, we can attempt to change the Profit $ column formula to the following one:

Click here to view code image

```
Profit $ = 'Stock Item'[Profit %] * 'Stock Item'[Unit Price]
```

Using this formula results in a circular dependency error: “A circular dependency was detected: Stock Item[Profit %], Stock Item[Profit $], Stock Item[Profit %].” This is because Profit % depends on Profit $, which also depends on Profit % to calculate its value. As a result, neither column can be calculated.

You can now remove both columns and go to the Scale table, where you have previously added the following column:

Click here to view code image

```
Sum of Scale Calculate = CALCULATE ( SUM ( Scale[Scale] ) )
```

For clarity purposes, rename it to Calculate1 and remove the Sum of Scale column, keeping only Scale and Calculate1 columns. Now, try to add the following column to the Scale table:

```
Calculate2 = CALCULATE ( SUM ( Scale[Scale] ) )
```

Note that this formula is the same as the Calculate1 column formula, and neither column references the other. At the same time, we get the circular dependency error: “A circular dependency was detected: Scale[Calculate1], Scale[Column], Scale[Calculate1].”

The reason why this happens lies in context transition. When DAX evaluates Calculate1, it converts the row context into equivalent filter context: it filters the Scale table to those rows where Scale and Calculate2 column values are the same as the values of the current row. For Calculate2, it keeps those Scale rows where Scale and Calculate1 column values are the same as current row values. Therefore, Calculate1 implicitly depends on Calculate2 and vice versa.

This situation happens in tables in which there is no column that is used as a primary key. In our case, the Scale column can be used as a primary key, because it contains unique values only. When a column is used as a primary key, DAX performs context transition differently: because it knows it can rely on the column having unique values, it uses this column to filter the table during context transition, without using values from other columns. For the Scale table, it means that during context transition DAX will only look at the Scale column values to filter the table.

The circular dependency issue can be fixed by creating a relationship between the Scale table and any other table with Scale being on the one side of the relationship. For review purposes, follow a technique developed by Marco Russo: introduce a dummy fact table and relate it to the Scale table. To fix the error, follow these steps:

1. Select **Home** > **External data** > **Get Data** > **Blank Query**.
2. Rename the query to ScaleDummyFact.
3. Open Advanced Editor by clicking **Home** > **Query** > **Advanced Editor**.
4. Replace all code with the following:

   Click here to view code image

   ```
   #table (
       type table [ Scale = Int64.Type ],
       {}
   )
   ```

5. Click **Done** >**Close & Apply**.
6. Create a relationship between Scale[Scale] and ScaleDummyFact[Scale]. The type of relationship does not matter as long as Scale[Scale] is on the one side of the relationship.

7. If necessary, go to Calculate2 formula and press Enter, prompting Power BI to re-evaluate the column.

> ***MORE INFO*** **CIRCULAR DEPENDENCIES IN DAX**
>
> For more details on circular dependencies in DAX, see two articles by Alberto Ferrari: "Understanding Circular Dependencies in Tabular and PowerPivot" at *https://www.sqlbi.com/articles/understanding-circular-dependencies/*, and "Avoiding circular dependency errors in DAX" at *https://www.sqlbi.com/articles/avoiding-circular-dependency-errors-in-dax/*.

At this stage, delete the calculated columns in the Scale table and remove the ScaleDummyFact table.

> ***MORE INFO*** **CALCULATED COLUMNS IN POWER BI DESKTOP**
>
> Calculated columns can help you to enrich your data models by using DAX, which is better suited for certain tasks compared to M. To learn more about calculated columns, you can refer to the official documentation page, "Using calculated columns in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-calculated-columns*, and "Tutorial: Create calculated columns in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-tutorial-create-calculated-columns*.
>
> For a video overview on creating calculated columns, see "Create calculated columns" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/modeling#step-3*.

Calculated tables

As previously mentioned, some DAX functions return tables. Table expressions can be used inside formulas of calculated columns and measures, as well as by themselves to materialize calculated tables. You can create a calculated table by selecting **Modeling** > **Calculations** > **New**

Table. You will then need to write a DAX formula for a table in the formula bar.

> ***MORE INFO*** **CREATING CALCULATED TABLES**
>
> For a video overview on how to create calculated tables, see "Create calculated tables" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/modeling#step-6*.

One way to create a calculated table is to duplicate an existing one. For example, you can duplicate the Date table:

```
Date Duplicate = 'Date'
```

Because Date is a reserved keyword, you need to enclose it in single quotation marks.

The technique of duplicating tables can be useful if you want to separate multiple relationships between two tables—for example, between Sale and Date.

FILTER

By using FILTER, you can filter a table based on specified condition. The FILTER function takes two arguments: a table expression and a filter condition. The filter condition is evaluated in row context for each row of the table. For example, create a calculated table for stock items in which unit price is greater than $300. Use the following formula shown in Listing 2-9.

LISTING 2-9 Expensive Stock Item calculated table with FILTER

Click here to view code image

```
Expensive Stock Items =
FILTER (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)
```

This formula creates a table with six rows. Note that if you had calculated columns in the Stock Item table, those columns would appear as native

columns in the Expensive Stock Items table.

Although FILTER takes only one condition, you can combine the conditions into a single Boolean condition with AND or OR logic. For instance, if you want to select only those stock items that are more expensive than $300 or are gray, so you can write the following formula shown in Listing 2-10.

LISTING 2-10 Expensive or Gray Stock Item calculated table

Click here to view code image

```
Expensive or Gray Stock Items =
FILTER (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] > 300,
        'Stock Item'[Color] = "Gray"
    )
)
```

We would get 15 rows in this case.

Unlike CALCULATETABLE, FILTER does not trigger context transition. The following two calculated columns, created in the Date table, produce different results:

Click here to view code image

```
// Different value for each row
Countrows Calculatetable = COUNTROWS ( CALCULATETABLE ( Sale ) )


// Same number for each row
Countrows Filter = COUNTROWS ( FILTER ( Sale, TRUE () ) )
```

Note that while CALCULATETABLE has only one mandatory parameter, FILTER always uses two parameters.

When you use FILTER in context transition—for example, in a calculated column—it generates new row context. This means that in each row of the table where we create a calculated column, DAX iterates over each row in the table used inside FILTER. It is possible to access the original row context from the new one with the EARLIER function. Doing so allows you to perform calculations in a way similar to SUMIF in Excel. For instance,

you can count the number of rows in the Date table for each month in the following calculated column formula shown in Listing 2-11.

LISTING 2-11 Days in Month calculated column with EARLIER

Click here to view code image

```
Days in Month EARLIER =
COUNTROWS (
    FILTER (
        'Date',
        'Date'[Calendar Month Label]
            = EARLIER ( 'Date'[Calendar Month Label] )
    )
)
```

The same effect can be achieved by using a variable instead of EARLIER. Because variables always stay constant after being evaluated, they are not affected by the new context. In a way, variables behave like constants. This behavior will be reviewed again later in the chapter. You can see an example of using a variable in Listing 2-12.

LISTING 2-12 Days in Month calculated column with VAR

Click here to view code image

```
Days in Month VAR =
VAR CurrentMonth = 'Date'[Calendar Month Label]
RETURN
    COUNTROWS (
        FILTER (
            'Date',
            'Date'[Calendar Month Label] = CurrentMonth
        )
    )
```

If needed, you can perform context transition inside FILTER. For example, you can create a calculated table for salespeople who have made over 5,000 sales:

Click here to view code image

```
Productive Salespeople =
FILTER (
    Employee,
    CALCULATE ( COUNTROWS ( Sale ) ) > 5000
)
```

IMPORTANT CONTEXT TRANSITION AND FILTERING

It is important to remember that when you perform context transition in large tables, the operation is much slower than filtering column values. Therefore, instead of filtering a table and doing context transition, it is advisable to pre-calculate the results in a calculated column and then filter by the column.

ALL

The ALL function removes any filters that were placed on a table or columns. This function can also be used to create calculated tables. The function accepts either a table as an argument or one or more columns from one table. The ALL function cannot accept another function as an argument. When you use a table as an argument for ALL, every row, including duplicated rows, is returned in the new table. When you use one column in ALL, a one-column table containing distinct values from the column is returned. When you use multiple columns from the same table in ALL, a table containing all existing combinations of the column values is returned. It is not possible to use columns from different tables inside one ALL statement. Note that if a table contains duplicate rows, then ALL with a table as the only argument and ALL with all columns listed will return tables with a different number of rows.

For examples of using ALL to create calculated tables, create the following three tables:

Click here to view code image

```
All Stock Item = ALL ( 'Stock Item' )

All Color = ALL ( 'Stock Item'[Color] )

All Color, Buying Package = ALL ( 'Stock Item'[Color], 'Stock
Item'[Buying Package] )
```

- The first table, All Stock Item, is a duplicate of the Stock Item table.

- The second table, All Color, contains distinct values from the Color column from the Stock Item table.
- The third table, All Color, Buying Package, contains all existing distinct combinations of the Color and Buying Package columns from the Stock Item table. This table is shown in Figure 2.23.

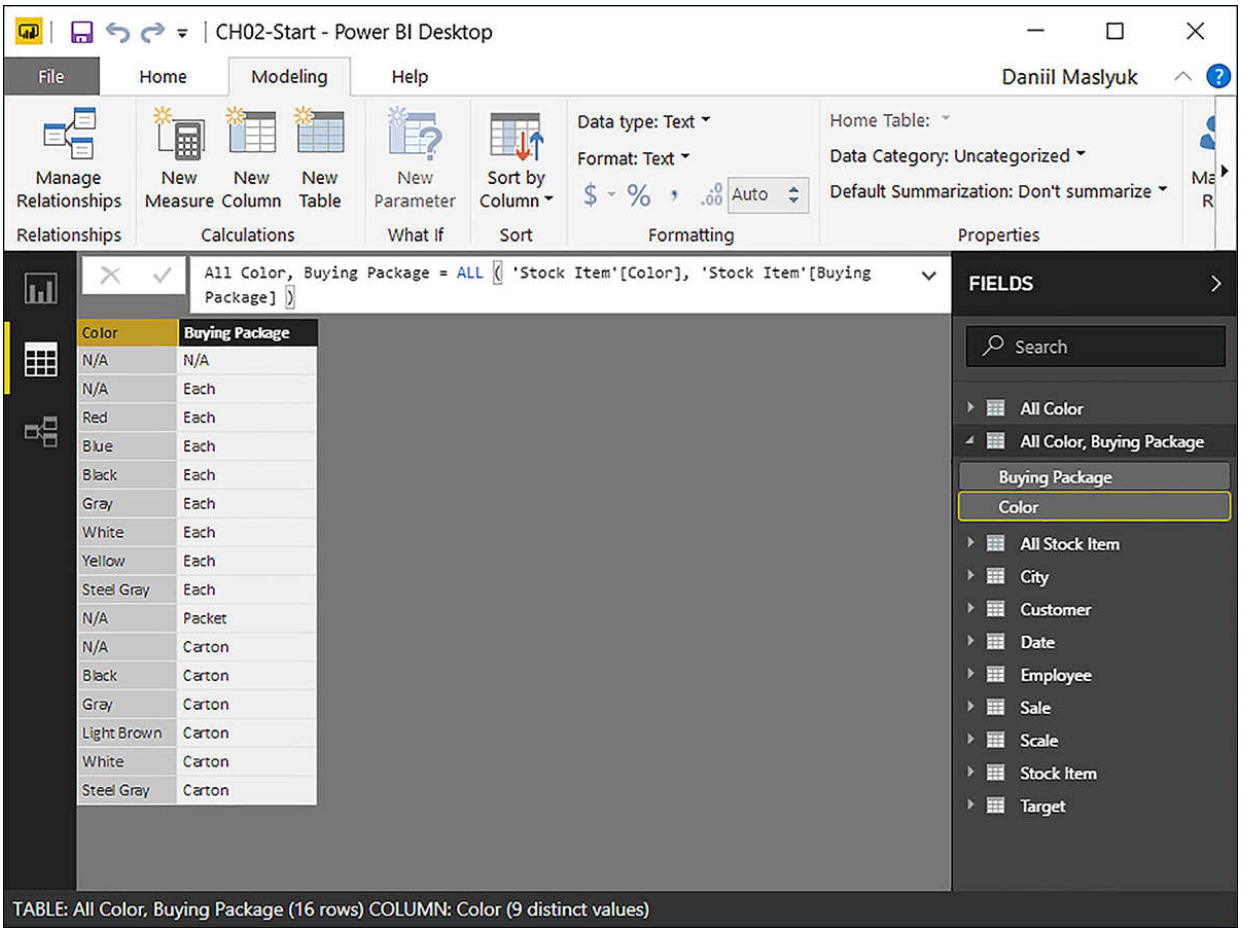


FIGURE 2.23 All Color, Buying Package

To see the difference that ALL makes in filter context, create the following calculated columns in the Customer table:

Click here to view code image

```
Customer Rows = COUNTROWS ( Customer )

Customer Rows Calculate = CALCULATE ( COUNTROWS ( Customer ) )

Customer Rows Calculate All = CALCULATE ( COUNTROWS ( ALL (
Customer ) ) )
```

- The first calculated column, Customer Rows, returns the total number of rows in the Customer table. Because there is no context transition, you get the same value, the total number of rows, for every row in the calculated column.
- The second calculated column, Customer Rows Calculate, includes CALCULATE, which triggers context transition. This means that for every row of the table, DAX counts only those rows that have the same column values as the current row. Because there is a primary key in this table, the result is always 1.
- The third calculated column, Customer Rows Calculate All, also includes CALCULATE, but it has Customer wrapped in ALL. Here is what happens: first, context transition transforms the row context into equivalent filter context. Because we have a primary key in the table, the Customer table for each row, is filtered to include that row only. Next, the changes produced by filter placed on the Customer table are undone with ALL.

Note that the calculated columns, Customer Rows and Customer Rows Calculate All, return the same value, 403, which is the total number of rows in the Customer table. These two values are not always equivalent. As discussed earlier in this chapter, when a table on the many side of a relationship includes some values that are not present in the table on the one side of a relationship, DAX adds a virtual row to the table on the one side. This row is blank and not visible by default. To review this effect on our formulas, follow these steps:

1. In the Fields pane, right-click on the Customer table and select Edit Query.
2. Click on the AutoFilter button of the Customer Key column.
3. If necessary, click Load More in the bottom-right corner of the filter list.
4. De-select values 1, 2, and 3, and click OK.
5. Click Close & Apply.

Once the Customer table is updated, you can see different values in the Customer Rows and Customer Rows Calculate All calculated columns: 400 and 401, respectively, as shown in Figure 2.24.

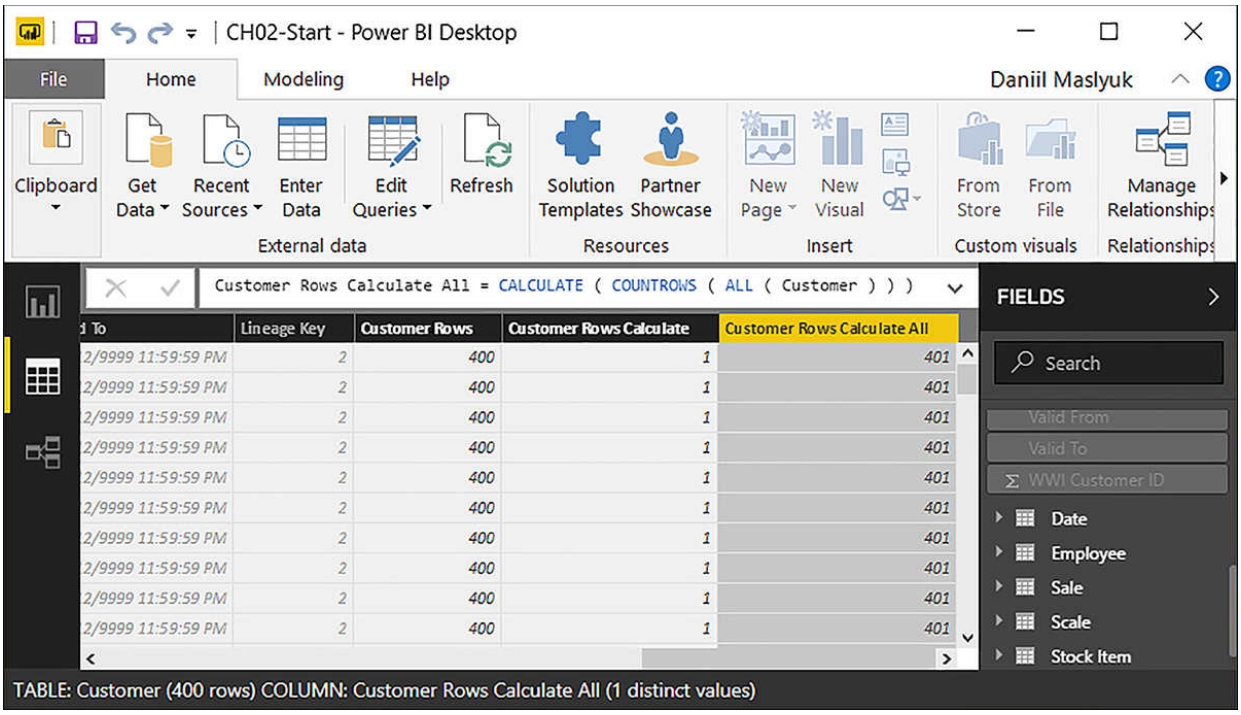


Figure 2.24 Customer Rows calculated columns

Note that the total number of rows in the Customer table is now 400, which can be seen at the bottom of the screen. The Customer Rows Calculate All, however, displays 401, which includes the virtual row that DAX included for those Customer Key values in the Sale table that do not have a corresponding value in the Customer table.

You can see this row materialized if you create the following calculated table:

```
All Customer = ALL ( Customer )
```

Note that this table has 401 rows, which includes the special blank row. Create a duplicate of the Customer table without the blank row in at least three ways. First, reference the original table without using any functions:

```
Duplicate Customer = Customer
```

Second, filter out the blank row:

Click here to view code image

```
Filter All Customer =
FILTER (
    ALL ( Customer ),
```

```
    NOT ISBLANK ( Customer[Customer Key] )
)
```

Third, use a variation of ALL – ALLNOBLANKROW:

```
AllNoBlankRow Customer = ALLNOBLANKROW ( Customer )
```

The ALLNOBLANKROW function returns a table without the virtual blank row that is added in cases in which a table on the many side contains values that are not in the table on the one side. Note that if a table or a column contains a genuine blank row or value, the ALLNOBLANKROW function will not filter it out. This function, like ALL, also removes any filters on a specified table or columns.

At this stage, go back to Power Query Editor and remove the Filtered Rows step from the Customer query.

The third variation of the ALL function, ALLEXCEPT, has a different syntax from ALL and ALLNOBLANKROW: it receives a table as the first argument, followed by at least one column to exclude. ALLEXCEPT returns all columns from a specified table except the excluded columns. This function can be useful when you want to include more columns than you want to exclude.

Another application of the ALLEXCEPT function can be inside calculated columns to calculate subtotals without using the combination of FILTER and EARLIER or VAR. Earlier in this chapter, we created two calculated columns that calculated the number of rows in the Date table for each month. The formulas of the columns can be seen in Listing 2-11 and Listing 2-12. The following formula returns the equivalent results.

Click here to view code image

```
Days in Month ALLEXCEPT =
CALCULATE (
    COUNTROWS ( 'Date' ),
    ALLEXCEPT (
        'Date',
        'Date'[Calendar Month Label]
    )
)
```

MORE INFO ALLEXCEPT IN DAX

To learn more about ALLEXCEPT and when you should use it, see "Using ALLEXCEPT versus ALL and VALUES" by Alberto Ferrari at *https://www.sqlbi.com/articles/using-allexcept-versus-all-and-values/*.

CALCULATETABLE

Earlier in the chapter, we encountered CALCULATETABLE when reviewing the RELATEDTABLE function. The latter is an alias for the former when only one argument is used. In CALCULATETABLE, you can specify optional conditions, which are combined with AND logic. CALCULATETABLE accepts either tables or Boolean statements as filter conditions. Any table expression can be used in place of filters, including functions that return tables. The following is an example of a Boolean filter condition:

```
'Stock Item'[Color] = "Black"
```

In the following example, we are reducing the number of rows in the Stock Item table, keeping only those in which Unit Price is greater than $300:

Click here to view code image

```
Expensive Stock Items =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)
```

Same as in Listing 2-9, we receive a table with six rows. Unlike FILTER, CALCULATETABLE can accept more than one filter parameter. The following calculated table contains stock items that are both black and priced higher than $300:

Click here to view code image

```
Expensive and Black Stock Items =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300,
    'Stock Item'[Color] = "Black"
)
```

It is possible to combine filter conditions with OR logic instead of AND logic using either the OR function or double pipe operator. For example, the following calculated table will filter the Stock Item table to contain only those items that are priced either below $1 or above $1000, but not at $0:

Click here to view code image

```
Stock Items below $1 or above $1000 =
CALCULATETABLE (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] < 1,
        'Stock Item'[Unit Price] > 1000
    ),
    'Stock Item'[Unit Price] <> 0
)
```

When you combine filters using the OR logic, the filters that you specify must be applied to one column only. The following calculated table, which is an attempt to reproduce the Expensive or Gray Stock Item calculated table from Listing 2-10, will result in an error:

Click here to view code image

```
Expensive or Gray Stock Item Wrong =
CALCULATETABLE (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] > 300,
        'Stock Item'[Color] = "Gray"
    )
)
```

The error message can be seen in Figure 2.25.

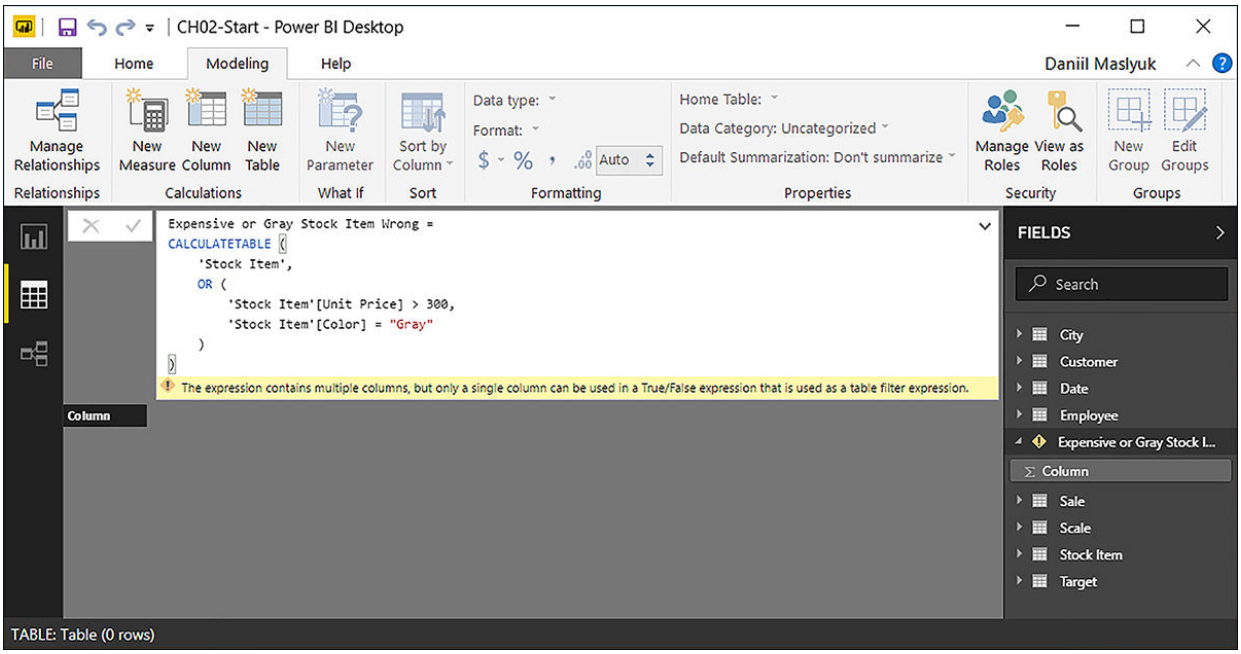


Figure 2.25 Error resulting from combining different columns in one OR statement in a CALCULATETABLE filter

The reason expression fails because internally, every Boolean filter condition in CALCULATETABLE and CALCULATE is transformed into a table. The following two table expressions are equivalent:

Click here to view code image

```
// Boolean filter condition
Expensive Stock Item Boolean =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)


// Table filter expression
Expensive Stock Item Filter All =
CALCULATETABLE (
    'Stock Item',
    FILTER (
        ALL ( 'Stock Item'[Unit Price] ),
        'Stock Item'[Unit Price] > 300
    )
)
```

When you combine different columns in one Boolean filter, DAX is unable to convert the Boolean filter to an equivalent FILTER ... ALL table. Instead, you need to write an equivalent filter table expression. The following expression returns the same table as the one shown in Listing 2-10.

Click here to view code image

```
Expensive or Gray Stock Items CalculateTable =
CALCULATETABLE (
    'Stock Item',
    FILTER (
        ALL (
            'Stock Item'[Unit Price],
            'Stock Item'[Color]
        ),
        OR (
            'Stock Item'[Unit Price] > 300,
            'Stock Item'[Color] = "Gray"
        )
    )
)
```

***MORE INFO* FILTER ARGUMENTS IN CALCULATE**

The approach described above only works for columns from one table. If you need to use the OR logic with columns from different tables, you need to follow a different approach. For a more detailed discussion, see the article by SQLBI, "Filter Arguments in CALCULATE" at: *https://www.sqlbi.com/articles/filter-arguments-in-calculate/*.

VALUES and DISTINCT

The VALUES and DISTINCT functions work similarly: they return tables that contain only distinct rows. Both functions can receive a column reference as a parameter. For example, the following table returns 12 month names from the Date table:

```
Months = DISTINCT ( 'Date'[Month] )
```

Both functions can also receive a table as a parameter. While the VALUES function can only receive a physical table as a parameter, the

DISTINCT function can also work with table expressions, which means that the table can be either a physical table or a table returned by another function.

Unlike the ALL function, which also returns a table with distinct rows, the VALUES and DISTINCT functions do not remove filters from their tables. In practice, this means that these two calculated columns in the Employee table return the same results:

Click here to view code image

```
Sale Rows Calculate = CALCULATE ( COUNTROWS ( Sale ) )

Sale Rows Calculate VALUES = CALCULATE ( COUNTROWS ( VALUES (
Sale ) ) )
```

As previously mentioned, if a table expression returns a table with one row and one column, it can be converted to a scalar value. The following calculated column in the Employee table works, too; it contains the same values as the Employee column:

Click here to view code image

```
Employee Calculate = CALCULATE ( VALUES ( Employee[Employee] ) )
```

Another major difference between the VALUES and DISTINCT functions is that the former might include a special blank row that is added when some values on the many side of a relationship do not have a matching value on the one side. The DISTINCT function never includes a blank row unless a blank row physically exists in the data. To illustrate this difference, use the Date table, which has an active relationship with the Sale table using the Delivery Date Key column. There are null values in the column, which causes a special blank row to be added. The following two tables return a different number of rows:

Click here to view code image

```
// 13 rows
Month Values = VALUES ( 'Date'[Month] )


// 12 rows
Month Distinct = DISTINCT ( 'Date'[Month] )
```

In this regard, the behavior of VALUES corresponds to ALL, while the behavior of DISTINCT corresponds to ALLNOBLANKROW.

SUMMARIZE and SUMMARIZECOLUMNS

The SUMMARIZE function allows you to group a table by one or more columns and add new columns, if necessary. The general syntax is as follows:

```
SUMMARIZE ( Table, OldColumns, NewColumns )
```

The new columns are defined by specifying a name and formula for each column. SUMMARIZE expects at least two arguments: a table to summarize and a column to group by.

The following formula produces a table with month names and numbers:

Click here to view code image

```
Month = SUMMARIZE ( 'Date', 'Date'[Month], 'Date'[Calendar Month
Number] )
```

In this case, the table does not include the special blank row that results from null values being present in the Delivery Date Key column of the Sale table. If we decide to add a column that contains the number of rows in the Sale table for each month, the blank row will still not be included. The summarized table can be seen in Figure 2.26.

Click here to view code image

```
Month Sale =
SUMMARIZE (
    'Date',
    'Date'[Month],
    'Date'[Calendar Month Number],
    "Sale Rows", COUNTROWS ( Sale )
)
```

Month	Calendar Month Number	Sale Rows
January	1	22620
February	2	18984
March	3	22485
April	4	22937
May	5	24165
June	6	17178
July	7	18735
August	8	15549
September	9	16001
October	10	17235
November	11	15970
December	12	16123

Figure 2.26 Date table summarized

Note that we summarized the Date table. If instead, we summarize the Sale table with the same columns, the blank row will appear, and it does not matter whether we include the Sale Rows column (see Listing 2-13). The table can be seen in Figure 2.27.

LISTING 2-13 Summarizing the Sale table

Click here to view code image

```
Month Sale =
SUMMARIZE (
    Sale,
    'Date'[Month],
    'Date'[Calendar Month Number],
    "Sale Rows", COUNTROWS ( Sale )
)
```

Month	Calendar Month Number	Sale Rows
		284
January	1	22620
February	2	18984
March	3	22485
April	4	22937
May	5	24165
June	6	17178
July	7	18735
August	8	15549
September	9	16001
October	10	17235
November	11	15970
December	12	16123

FIGURE 2.27 Sale table summarized

The columns to group by are optional. You can specify a table to summarize, new column names, and expressions. If you specify only one new column name and expression, it will result in a one-row and one-column table with a single value. As in the previous examples, the table you summarize can make a difference.

Click here to view code image

```
// Returns 198043439.450001
Summarize Sale Single =
SUMMARIZE (
    Sale,
    "Total Sales", SUM ( Sale[Total Including Tax] )
)

// Returns 197776428.010001
Summarize Date Single =
```

```
SUMMARIZE (
    'Date',
    "Total Sales", SUM ( Sale[Total Including Tax] )
)
```

An example of such a table can be seen in Figure 2.28.

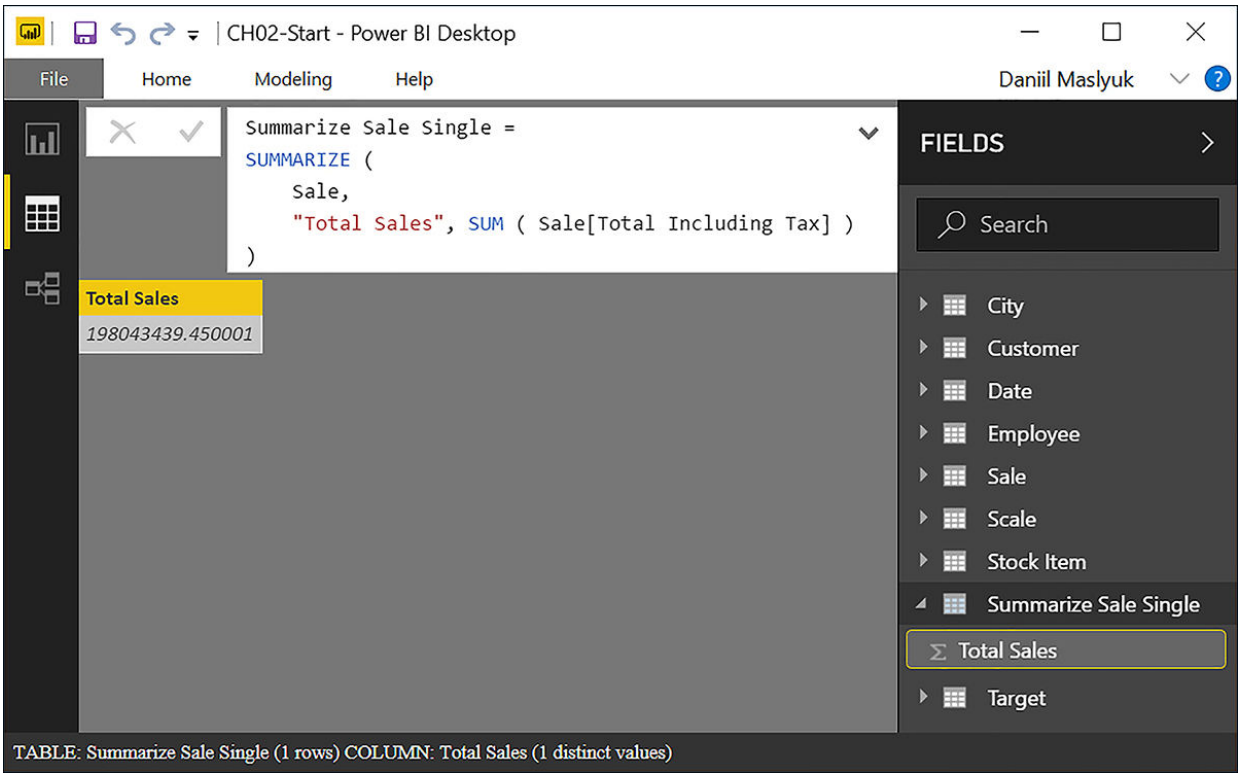


FIGURE 2.28 Sale table summarized to one row

SUMMARIZE only returns rows that have data. For example, if you summarize the Date table by Calendar Year and Month, you will get 48 rows. If you summarize the Sale table instead, you will get 42 rows: 41 Calendar Year and Month combinations plus the special blank row.

Click here to view code image

```
// Returns 48 rows
Date Year Month =
SUMMARIZE (
    'Date',
    'Date'[Calendar Year],
    'Date'[Month],
    'Date'[Calendar Month Number]
)
```

```
// Returns 42 rows
Sale Year Month =
SUMMARIZE (
    Sale,
    'Date'[Calendar Year],
    'Date'[Month],
    'Date'[Calendar Month Number]
)
```

In SUMMARIZE, you do not have access to row context of the table you are summarizing. Instead, SUMMARIZE divides the table into parts, grouping them by the columns you select, with each part of the original table having its own filter context. This is why you don't need to wrap COUNTROWS (Sales) in CALCULATE to trigger context transition, as no context transition is necessary.

There is a function similar to SUMMARIZE: SUMMARIZECOLUMNS. It performs similar operations, except you do not need to specify the table you want to summarize. It also works in a slightly different way from SUMMARIZE. For instance, if you use the function to create a table with Calendar Year and Month combinations, you will get a table with 49 rows: 48 existing Calendar Year and Month combinations, plus the special blank row. If you add a count of Sale rows, the table will have 42 rows, which is the same as using SUMMARIZE with the Sale table:

Click here to view code image

```
// 49 rows
Month Year = SUMMARIZECOLUMNS ( 'Date'[Calendar Year],
'Date'[Month] )

// 42 rows
Month Year =
SUMMARIZECOLUMNS (
    'Date'[Calendar Year],
    'Date'[Month],
    "Sale Rows", COUNTROWS ( Sale )
)
```

MORE INFO SUMMARIZE AND SUMMARIZECOLUMNS

Both SUMMARIZE and SUMMARIZECOLUMNS can be useful when creating aggregated tables. There are a few more nuances about the functions, which are outside of the scope of this book. To

read more about the functions, see “SUMMARIZE Function (DAX)” at *https://msdn.microsoft.com/en-us/library/gg492171.aspx*, and “SUMMARIZECOLUMNS Function (DAX)” at *https://msdn.microsoft.com/en-us/library/mt163696.aspx*.

Additionally, it is worth reading the following articles by Alberto Ferrari and Marco Russo: “All the secrets of SUMMARIZE” at *https://www.sqlbi.com/articles/all-the-secrets-of-summarize/*, and “Introducing SUMMARIZECOLUMNS” at *https://www.sqlbi.com/articles/introducing-summarizecolumns/*.

ADDCOLUMNS and SELECTCOLUMNS

The ADDCOLUMNS function adds new columns to a table, generating row context in the process. The new columns are also known as extension columns. The function expects at least three arguments: a table to add columns to, a new column name, and a new column expression. For example, the following calculated table is equivalent to the one from Listing 2-13:

Click here to view code image

```
AddColumns Month Sale =
ADDCOLUMNS (
    ALL ( 'Date'[Month], 'Date'[Calendar Month Number] ),
    "Sale Rows", CALCULATE ( COUNTROWS ( Sale ) )
)
```

The row context in ADDCOLUMNS makes this function different from SUMMARIZE in two ways:

- First, you need to perform context transition to get different values for each row (you can see it in the preceding formula). Without CALCULATE, we would get the same value for each row, which would be the total number of rows in the Sale table.
- Second, you can reference columns in the table to which you add columns. For instance, you can take all the Calendar Year and Calendar Month Number combinations from the Date table and create a column that puts them in a sequential order:

Click here to view code image

```
AddColumns YearMonthSequential =
ADDCOLUMNS (
    ALL (
        'Date'[Calendar Year],
        'Date'[Calendar Month Number]
    ),
    "Year Month Sequential",
    'Date'[Calendar Year] * 12 + 'Date'[Calendar Month Number]
)
```

To achieve the same effect using SUMMARIZE or SUMMARIZECOLUMNS, which use filter context, you must use VALUES to convert multiple column values into scalar values:

Click here to view code image

```
SummarizeColumns YearMonthSequential =
SUMMARIZECOLUMNS (
    'Date'[Calendar Year],
    'Date'[Calendar Month Number],
    "Year Month Sequential",
    VALUES ( 'Date'[Calendar Year] ) * 12
        + VALUES ( 'Date'[Calendar Month Number] )
)
```

The SELECTCOLUMNS function works like ADDCOLUMNS, except it does not keep the original columns. If you want to keep some of them using this function, you will have to create new columns that reference them. For example, you can take a table that contains all Calendar Year and Month combinations and keep only the Calendar Year column:

Click here to view code image

```
SelectColumns Calendar Year =
SELECTCOLUMNS (
    ALL ( 'Date'[Calendar Year], 'Date'[Month] ),
    "Year", 'Date'[Calendar Year]
)
```

The first few rows of the resulting table can be seen in Figure 2.29. Note that SELECTCOLUMNS does not produce distinct values in its columns:

Year
2013
2013
2013
2013
2013
2013
2013
2013
2013
2013
2013
2013
2014
2014
2014
2014
2014

FIGURE 2.29 Partial results of the SelectColumns Calendar Year calculated table

It is possible to group the results by extension columns using SUMMARIZE. For example, you can create an extension column that returns a quarter label for each date and then get the distinct values of this column. The results can be seen in Figure 2.30.

Click here to view code image

```
Summarize SelectColumns Quarter =
SUMMARIZE (
```

```
    SELECTCOLUMNS (
        'Date',
        "Quarter", FORMAT ( 'Date'[Date], "\QQ" )
    ),
    [Quarter]
)
```

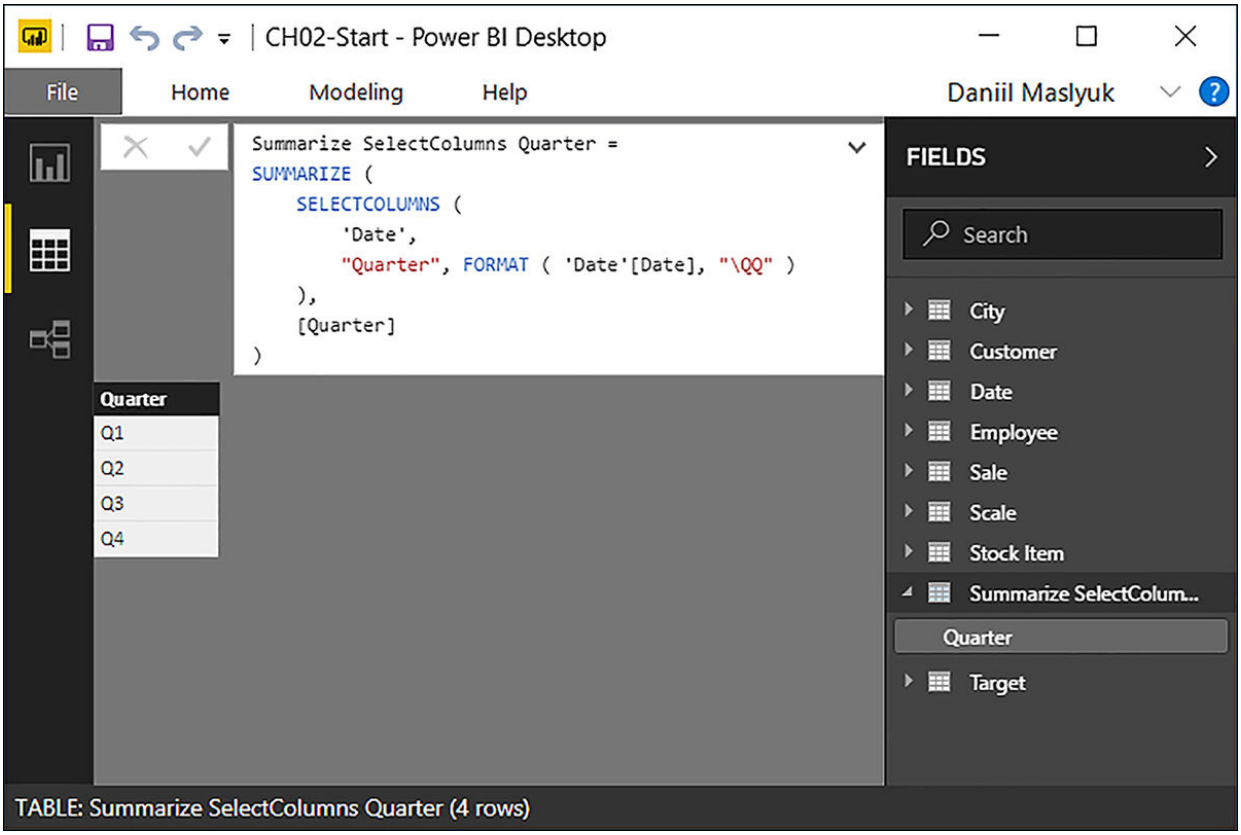


FIGURE 2.30 Date summarized by extension column, Quarter

Note that you must reference the extension column without a table name because it is not a physical column.

> ***NOTE*** **ALIASING COLUMNS WITH SELECTCOLUMNS**
> SELECTCOLUMNS can be used to rename columns. For more information on this technique, see a blog post by Chris Webb, "Using SelectColumns() To Alias Columns In DAX" at *https://blog.crossjoin.co.uk/2015/06/01/using-selectcolumns-to-alias-columns-in-dax/*.

***MORE INFO* ADDCOLUMNS AND SELECTCOLUMNS**

For more information on and examples of using ADDCOLUMNS and SELECTCOLUMNS, see "ADDCOLUMNS Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492204.aspx*, and "SELECTCOLUMNS Function (DAX)" at *https://msdn.microsoft.com/en-us/library/mt761759.aspx*.

TOPN

The TOPN function ranks table rows by specified criteria and takes the top N rows. There are three required arguments: N number, table expression, and expression to order rows by. The fourth argument, which is optional, defines the order: ASC for ascending and DESC for descending. If omitted, the default value is DESC. The order of rows is not guaranteed.

For example, you can rank salespeople by sales amount and take the top three. The result is a subset of the original table and does not include the values you are ranking by:

Click here to view code image

```
Top 3 Employees by Sales =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
)
```

Employee
Hudson Onslow
Kayla Woodcock
Archer Lamble

Note that TOPN uses row context, so to rank employees properly you need to perform context transition. Without CALCULATE, you would get incorrect results. Because we can use row context, you can also get the first three employees alphabetically if you order them by name:

Click here to view code image

```
Top 3 Employees by Name =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    Employee[Employee],
    ASC
)
```

Employee
Anthony Grosse
Alica Fatnowna
Amy Trefl

In the case of ties, you get more rows than expected. For instance, if you order Wide World Importers employees by sales in ascending order and take the top three, you end up with nine rows because nine employees did not sell anything. As a result, they all tie for the first place with zero (more precisely, blank) sales.

Click here to view code image

```
Bottom 3 Employees by Sales =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    CALCULATE ( SUM ( Sale[Total Excluding Tax] ) ),
    ASC
)
```

Employee
Jai Shand
Ethan Onslow
Isabella Rupp
Piper Koch
Henry Forlonge
Stella Rosenhain
Katie Darwin
Alica Fatnowna
Eva Muirden

> ***NOTE*** **USING TOPN**
>
> With TOPN, it is also possible to order by more than one expression: in this case, expressions and orders come in pairs after the first expression and order. For more information about the TOPN function, see "TOPN Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492198.aspx*.

CROSSJOIN, GENERATE, and GENERATEALL

The CROSSJOIN function allows you to create a Cartesian product between two or more tables. For example, you can create a table with all possible Brand and Buying Package combinations, not just those that exist in your data:

Click here to view code image

```
Buying Package, Brand =
CROSSJOIN (
    VALUES ( 'Stock Item'[Buying Package] ),
    VALUES ( 'Stock Item'[Brand] )
)
```

The resulting table has eight rows: four Buying Package values multiplied by two Brand values. If you wrote `ALL ( 'Stock Item'[Buying Package], 'Stock Item'[Brand] )`, you would get only five

rows. All columns in the resulting tables must be unique. This means that if you want to create a Cartesian product of a table with itself, you must rename the columns of one of the tables in advance. For example, you could rename one of the columns using SELECTCOLUMNS:

Click here to view code image

```
Buying Package CrossJoin =
CROSSJOIN (
    VALUES ( 'Stock Item'[Buying Package] ),
    SELECTCOLUMNS (
        VALUES ( 'Stock Item'[Buying Package] ),
        "Buying Package 2", 'Stock Item'[Buying Package]
    )
)
```

With CROSSJOIN, there is no row context that you can use when writing the expression of the second and subsequent tables. The GENERATE function, which always receives two table expressions as parameters, allows you to reference the current row in the first table when writing the second table expression. For instance, create a table with calendar years and top three employees in each year:

Click here to view code image

```
Top 3 Employees per Calendar Year =
GENERATE (
    DISTINCT ( 'Date'[Calendar Year] ),
    TOPN (
        3,
        VALUES ( Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)
```

Calendar Year	Employee
2013	Archer Lamble
2013	Kayla Woodcock
2013	Hudson Onslow
2014	Hudson Hollinworth
2014	Archer Lamble
2014	Kayla Woodcock
2015	Hudson Hollinworth
2015	Lily Code
2015	Jack Potter
2016	Hudson Hollinworth
2016	Taj Shand
2016	Archer Lamble

The resulting table includes only those years and employees that actually had sales. If you wanted to include all years and employees that meet our criteria, even those that did not make any sales, use the GENERATEALL function. It works in the same way as GENERATE, except it includes all possible combinations:

Click here to view code image

```
// Returns 123 rows
Top 3 Employees per Calendar Year Month =
GENERATE (
    ALLNOBLANKROW ( 'Date'[Calendar Year], 'Date'[Month] ),
    TOPN (
        3,
        SUMMARIZE ( RELATEDTABLE ( Sale ), Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)

// Returns 130 rows
Top 3 Employees per Calendar Year Month =
GENERATEALL (
    ALLNOBLANKROW ( 'Date'[Calendar Year], 'Date'[Month] ),
    TOPN (
```

```
        3,
        SUMMARIZE ( RELATEDTABLE ( Sale ), Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)
```

In the above expressions, the seven-row difference comes from months from June to December 2016, which have no deliveries, yet we have these combinations in our Date table.

> ***MORE INFO* CROSSJOIN, GENERATE, AND GENERATEALL**
>
> For more information and examples of using the three table functions discussed above, see "CROSSJOIN Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492168.aspx*, "GENERATE Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492196.aspx*, and "GENERATEALL Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492206.aspx*.

GENERATESERIES

With GENERATESERIES, you can generate a table with one column, called Value, containing a list of numbers with predefined increment. These values need not exist in the data model. The function expects at least two arguments: start value and end value. If the start value is greater than the end value, the result will be a table with no rows. The optional third parameter specifies the increment; if omitted, it is 1 by default. The following expression outputs a list of numbers from 1 to 5:

```
1 to 5 = GENERATESERIES ( 1, 5 )
```

Value
1
2
3
4
5

Using the optional third parameter, you can create consecutive lists that increment by a number other than 1. For example, generate a list of odd numbers from 1 to 9 inclusive:

```
Odd 1 to 9 = GENERATESERIES ( 1, 10, 2 )
```

Value
1
3
5
7
9

Note that even though we specified 10 as the end value, the table only goes up to 9 because the next value in the sequence, 11, falls outside of the specified range.

The GENERATESERIES function automatically detects the data type. The only column in the above table has the Whole Number data type. In the following expression, the data type is set as Decimal Number:

```
0.1 to 0.5 Decimal = GENERATESERIES ( 0.1, 0.5, 0.1 )
```

Value
0.1
0.2
0.3
0.4
0.5

It is also possible to specify data types explicitly. For instance, you can use the CURRENCY function to convert values to the Fixed Decimal Number type:

Click here to view code image

```
1 to 5 Fixed Decimal = GENERATESERIES ( CURRENCY ( 1 ), CURRENCY
( 5 ) )
```

GENERATESERIES can be used to generate lists of datetime values as well. The following table expression creates a one-column table of Date/Time data type that starts at 12 am on 1 May 2018 and finishes at 12 am on 3 May 2018, incrementing by 12 hours:

Click here to view code image

```
1 to 3 May 2018 at 12-hour intervals =
GENERATESERIES (
    DATE ( 2018, 5, 1 ),
    DATE ( 2018, 5, 3 ),
    TIME ( 12, 0, 0 )
)
```

Value
1/05/2018 12:00:00 AM
1/05/2018 12:00:00 PM
2/05/2018 12:00:00 AM
2/05/2018 12:00:00 PM
3/05/2018 12:00:00 AM

Though GENERATESERIES can only generate numeric or datetime value lists, it is possible to generate lists of letters by combining SELECTCOLUMNS and UNICHAR:

Click here to view code image

```
Uppercase Latin alphabet: =
SELECTCOLUMNS (
    GENERATESERIES ( 65, 90 ),
    "Letter", UNICHAR ( [Value] )
)
```

Letter
A
B
C
…
Z

***NOTE* USING THE UNICHAR FUNCTION**

The UNICHAR function takes a positive integer as its only parameter and returns a Unicode character with the code. A list of Unicode characters and their corresponding decimal numbers can be found online on websites such as Wikipedia at: *https://en.wikipedia.org/wiki/List_of_Unicode_characters*.

For more examples of using the function, see Chris Webb's blog post, "The DAX Unichar() Function And How To Use It In Measures For Data Visualization" at *https://blog.crossjoin.co.uk/2017/04/11/the-dax-unichar-function-and-how-to-use-it-in-measures-for-data-visualisation/*.

The GENERATESERIES function is used when you create a parameter with the What if feature in Power BI Desktop. This feature is covered in this section under "Use What If parameters".

***MORE INFO* USING GENERATESERIES**

For more information and examples on using the GENERATESERIES function, see “GENERATESERIES Function” at *https://msdn.microsoft.com/en-us/library/mt842624.aspx*, and Marco Russo’s article, “Generating a series of numbers in DAX” at *https://www.sqlbi.com/articles/generating-a-series-of-numbers-in-dax/*.

CALENDAR and CALENDARAUTO

The CALENDAR function works like GENERATESERIES when you work with dates only: it generates a one-column table of datetime data type called Date. This function can be especially useful when your data model does not have a Date table and you want to create your own calendar.

CALENDAR has two required parameters: start date and end date. By default, the values increment by one day. The following expression generates a table with 365 rows, starting with 1 January 2018 and ending with 31 December 2018:

```
Year 2018 = CALENDAR ( DATE ( 2018, 1, 1 ), DATE ( 2018, 12, 31 ) )
```

Date
1/01/2018 12:00:00 AM
2/01/2018 12:00:00 AM
3/01/2018 12:00:00 AM
…
29/12/2018 12:00:00 AM
30/12/2018 12:00:00 AM
31/12/2018 12:00:00 AM

The CALENDARAUTO function generates a list of dates, taking into account all date and datetime type columns in the data model: it takes the minimum and maximum of all dates found and extracts years; then it

generates a list of dates starting with 1 January of the minimum year and ending with 31 December of the maximum year.

CALENDARAUTO has one optional parameter, which is the fiscal year end month. If omitted, the default value is 12, which corresponds to the year ending on 31 December. If your data model contains only dates from 1 January 2018 to 31 December 2018, then the following calculated table returns all dates from 1 July 2017 to 30 June 2019:

```
Fiscal Date = CALENDARAUTO ( 6 )
```

Because CALENDARAUTO considers all the date and datetime columns in a data model, sometimes this can lead to tables having more rows than necessary. For example, creating the following, calculated in the Wide World Importers data model, results in a table with close to three million rows:

```
All Dates = CALENDARAUTO ()
```

You can see the maximum date in this table by right-clicking on the resulting Date column and selecting Sort Descending. Note that the maximum date is 31 December 9999. The reason why this happens is that we have Valid From and Valid To columns in some tables of our data model, such as City. The values of the Valid To column are often 31 December 9999. Therefore, CALENDARAUTO generates a table with values for every date up to that date.

Unless your data model contains many meaningful date or datetime columns, it is preferable to use the CALENDAR function, referencing the relevant date columns. If we did not have the Date table in our data model, and the only date column we had was the Invoice Date Key column in the Sale table, we could create the following date table:

Click here to view code image

```
Calendar = CALENDAR ( MIN ( Sale[Invoice Date Key] ), MAX (
Sale[Invoice Date Key] ) )
```

Because we also have the Delivery Date Key column in our data model, you can use the alternative MIN and MAX syntax, which allows the comparison of two scalar values:

Click here to view code image

```
Calendar =
CALENDAR (
    MIN (
        MIN ( Sale[Delivery Date Key] ),
        MIN ( Sale[Invoice Date Key] )
    ),
    MAX (
        MAX ( Sale[Delivery Date Key] ),
        MAX ( Sale[Invoice Date Key] )
    )
)
```

If you prefer having complete years in your calendar table, you can use the DATE/YEAR combination:

Click here to view code image

```
Calendar =
CALENDAR (
    DATE (
        YEAR (
            MIN (
                MIN ( Sale[Delivery Date Key] ),
                MIN ( Sale[Invoice Date Key] )
            )
        ),
        1,
        1
    ),
    DATE (
        YEAR (
            MAX (
                MAX ( Sale[Delivery Date Key] ),
                MAX ( Sale[Invoice Date Key] )
            )
        ),
        12,
        31
    )
)
```

You can then add calculated columns, such as month name and calendar year number, to the newly created table to make it a proper calendar table.

MORE INFO USING CALENDAR AND CALENDARAUTO

For more information on the CALENDAR and CALENDARAUTO functions, see “CALENDAR Function (DAX)” at *https://msdn.microsoft.com/en-us/library/dn802546.aspx*, and “CALENDARAUTO Function (DAX)” at *https://msdn.microsoft.com/en-us/library/dn802534.aspx*.

ROW

With the ROW function, you can create one-row tables with several columns at once. The arguments come in pairs: a column name goes first, then an expression. Only one pair of arguments is required. For example, you can create a one-row table with two columns: number of rows in the Sale table and number of rows in the Date table:

Click here to view code image

```
One Row =
ROW (
    "Sale Rows", COUNTROWS ( Sale ),
    "Date Rows", COUNTROWS ( 'Date' )
)
```

Sale Rows	Date Rows
228266	1461

The ROW function can be useful when you want to add a row to another table using the UNION function, covered next.

***MORE INFO* THE ROW FUNCTION**

For more information on the ROW function, see “ROW Function (DAX)” at *https://msdn.microsoft.com/en-us/library/gg492184.aspx*.

UNION

The UNION function works similarly to the Append feature in Power Query Editor: it combines two or more tables vertically. If the tables you are combining have the same rows among them, duplicate rows will be retained. The names of the columns do not have to match, but the number of columns

in tables must be the same because tables are combined by the position of columns, not the names. The output table will have the same column names as the first table. If tables have different data types, they will be combined in accordance with DAX data type coercion. The following is an example of UNION usage:

Click here to view code image

```
Table1 =
UNION (
    ROW ( "Color", "Red", "Value 1", 1 ),
    ROW ( "Color", "Red", "Value 1", 2 ),
    ROW ( "Color", "Green", "Value 1", 3 ),
    ROW ( "Color", "Blue", "Value 1", CURRENCY ( 1 ) )
)
```

Color	Value 1
Red	1
Red	2
Green	3
Blue	1

Note that in this case, the second column will have the Fixed Decimal Number data type.

UNION can be used to create common dimensions from several different tables. For instance, the Calendar Year column in the Target table and the Date table are examples. In addition to that, we have the Bill To Customer column in both Target and Customer tables. To create bridging tables to pass filters from Date and Customer tables to the Target table, create the following calculated tables:

Click here to view code image

```
// Bridging table between Date and Target

Calendar Year =
DISTINCT (
    UNION (
        ALLNOBLANKROW ( Target[Calendar Year] ),
        ALLNOBLANKROW ( 'Date'[Calendar Year] )
    )
)
```

```
// Bridging table between Customer and Target

Bill To Customer =
DISTINCT (
    UNION (
        ALLNOBLANKROW ( Target[Bill To Customer] ),
        ALLNOBLANKROW ( Customer[Bill To Customer] )
    )
)
```

Hide both bridging tables and create the following four relationships:

- From ‘Target’[Bill To Customer] to ‘Bill To Customer’[Bill To Customer]
- From ‘Target’[Calendar Year] to ‘Calendar Year’[Calendar Year]
- Bidirectional from ‘Customer’[Bill To Customer] to ‘Bill To Customer’[Bill To Customer]
- Bidirectional from ‘Date’[Calendar Year] to ‘Calendar Year’[Calendar Year]

At this point, if you go to the relationships view, your data model should look similar to Figure 2.31.

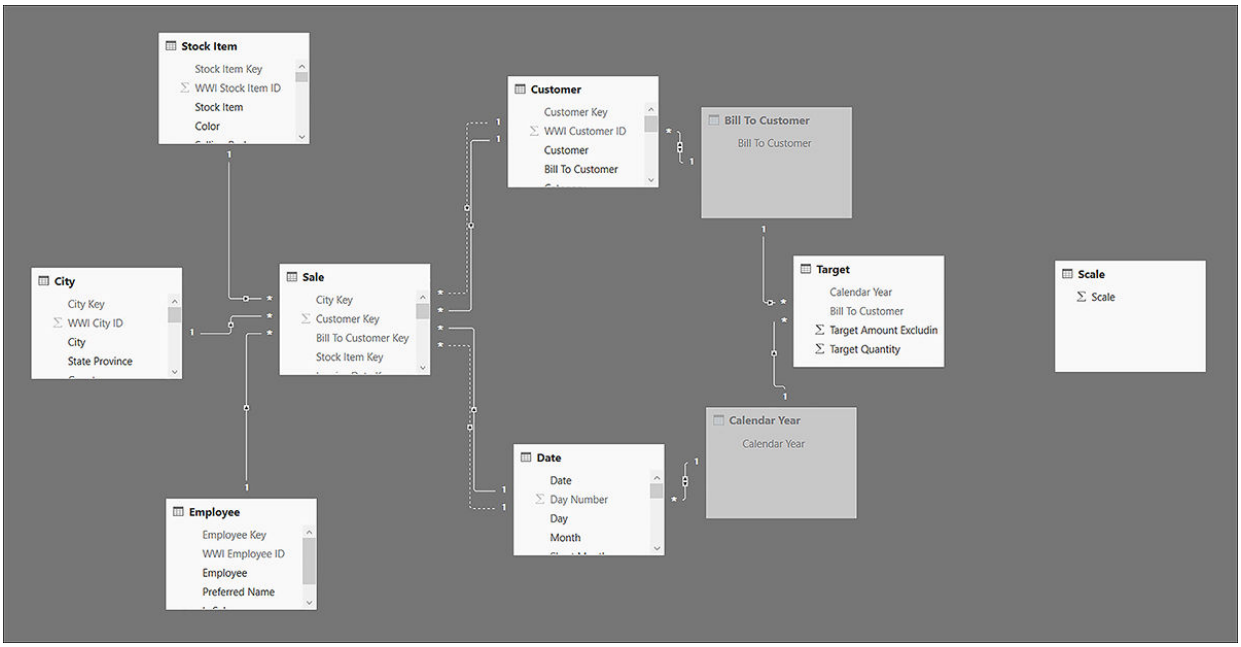


Figure 2.31 Relationships view after adding and relating bridging tables

MORE INFO USING UNION

For more information on the UNION function, see “UNION Function (DAX)” at *https://msdn.microsoft.com/en-us/library/dn802530.aspx*.

INTERSECT

The INTERSECT function creates a table that consists of rows that are present in both tables that are used as arguments in INTERSECT. In the following examples, we are going to use the function on these two tables, called TableOne and TableTwo.

TABLE 2-11 TableOne

Color	Value 1
Red	1
Red	2
Green	3
Blue	1

TABLE 2-12 TableTwo

Color	Value 2
Green	3
Blue	1
Blue	1
Yellow	2

Note that there is a common column name, Color, which differs from the name of the second column. Also, these tables have common rows. INTERSECT has the same requirements for tables as UNION: both tables must have the same number of columns, and the tables are combined based on the position of columns. The result of the following expression is a two-row table that has the same column names as TableOne:

Click here to view code image

```
IntersectOneTwo =
INTERSECT (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

Color	Value 1
Green	3
Blue	1

The order in which two tables are used in INTERSECT matters. In the following calculated table, we are also using TableOne and TableTwo, but in reverse order:

Click here to view code image

```
IntersectTwoOne =
INTERSECT (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

Color	Value 2
Green	3
Blue	1
Blue	1

Note that the second column name is now Value 2 instead of Value 1. Also, the output table retains duplicate rows from the first table but not from the second table.

> ***MORE INFO* USING INTERSECT**
>
> For more information on the INTERSECT function, see "INTERSECT Function (DAX)" at *https://msdn.microsoft.com/en-us/library/mt243783.aspx*.

EXCEPT

The EXCEPT function takes two tables as arguments and outputs all rows that are in the first table but not in the second table. Columns are compared based on their positions, so the number of columns in both tables must be the same, which is the same behavior as with the UNION and INTERSECT functions.

As with INTERSECT, the order of tables used as arguments influences the results. Note the difference in the results when EXCEPT is used with TableOne and TableTwo versus TableTwo and Table one:

Click here to view code image

```
ExceptOneTwo =
EXCEPT (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
```

```
    )
)
```

Color	Value 1
Red	1
Red	2

Click here to view code image

```
ExceptTwoOne =
EXCEPT (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

Color	Value 2
Yellow	2

Like with INTERSECT, the output table has the same column names as the first table. Duplicate rows from the first table, if any, are retained.

> ***MORE INFO* USING EXCEPT**
>
> For more information on the EXCEPT function, see "EXCEPT Function (DAX)" at *https://msdn.microsoft.com/en-us/library/mt243784.aspx*.

NATURALINNERJOIN

The NATURALINNERJOIN function works like the Merge feature in Power Query Editor: it receives two tables as arguments and joins them

based on common column names. The columns that are used for joining must have the same data types. NATURALINNERJOIN joins two tables and outputs a table that has the same values present in join columns of both tables.

In the following examples, we are again using TableOne and TableTwo:

Click here to view code image

```
NaturalInnerJoinOneTwo =
NATURALINNERJOIN (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

Color	Value 1	Value 2
Green	3	3
Blue	1	1
Blue	1	1

Note that in this case, the order in which you join tables only matters when considering the order of columns:

Click here to view code image

```
NaturalInnerJoinTwoOne =
NATURALINNERJOIN (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
```

```
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

Color	Value 2	Value 1
Green	3	3
Blue	1	1
Blue	1	1

NATURALINNERJOIN can also join physical tables that have a relationship between them. For instance, create the following two calculated tables:

Click here to view code image

```
// Many side of a relationship
TableOne =
UNION (
    ROW ( "ColorOne", "Red", "Value 1", 1 ),
    ROW ( "ColorOne", "Red", "Value 1", 2 ),
    ROW ( "ColorOne", "Green", "Value 1", 3 ),
    ROW ( "ColorOne", "Blue", "Value 1", 1 )
)
```

ColorOne	Value 1
Red	1
Red	2
Green	3
Blue	1

Click here to view code image

```
// One side of a relationship
TableThree =
UNION (
    ROW ( "ColorThree", "Green", "Value 3", 3 ),
    ROW ( "ColorThree", "Blue", "Value 3", 1 ),
    ROW ( "ColorThree", "Yellow", "Value 3", 2 )
)
```

ColorThree	Value 3
Green	3
Blue	1
Yellow	2

Once you create a relationship between ColorOne and ColorThree, you can create the following calculated table:

Click here to view code image

```
NaturalInnerJoin One Three = NATURALINNERJOIN ( TableOne,
TableThree )
```

ColorThree	Value 3	ColorOne	Value 1
Blue	1	Blue	1
Green	3	Green	3

In this case, the order of arguments make no difference. Note that we are only able to create this calculated table because there are no columns that have the same name and there is a relationship between these tables. Without a relationship, we would get the following error: “No common join columns detected. The join function ‘NATURALINNERJOIN’ requires at-least one common join column.”

If we were joining two related tables that had columns that shared names, we would get an error like the following one: “The Column with the name of ‘ColorOne’ already exists in the ‘Table’ Table.” This limitation is not unique to NATURALINNERJOIN–in general, all column names in materialized tables must be unique in DAX. Virtual tables can have the same column names in some cases. The following calculated table works even if TableOne and TableThree have common column names:

Click here to view code image

```
Aggregated Virtual Table =
ROW (
    "NumRows",
    COUNTROWS ( NATURALINNERJOIN ( TableOne, TableThree ) )
)
```

NumRows
2

The ability to join tables with common names can be useful when passing filters to CALCULATE or CALCULATETABLE.

> ***MORE INFO* USING NATURALINNERJOIN**
>
> For more information on the NATURALINNERJOIN function, see "NATURALINNERJOIN Function (DAX)" at *https://msdn.microsoft.com/en-us/library/dn802543.aspx*.

NATURALLEFTOUTERJOIN

The NATURALLEFTOUTERJOIN function is similar to NATURALINNERJOIN, but it performs the left outer join instead of the inner join. NATURALLEFTOUTERJOIN returns a table with all rows from the first table and extra columns from the second table where values in the join columns of the right table are present in the join columns of the first table:

Click here to view code image

```
NaturalLeftOuterJoinOneTwo =
NATURALLEFTOUTERJOIN (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

Color	Value 1	Value 2
Red	1	
Red	2	
Green	3	3
Blue	1	1
Blue	1	1

Because NATURALLEFTOUTERJOIN performs a left outer join, the order of tables used as parameters is very important. The following table not only has a different order of columns, but the rows are also different:

Click here to view code image

```
NaturalLeftOuterJoinTwoOne =
NATURALLEFTOUTERJOIN (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

Color	Value 2	Value 1
Green	3	3
Blue	1	1
Blue	1	1
Yellow	2	

Same as NATURALINNERJOIN, NATURALLEFTOUTERJOIN can be used with tables that have a relationship and no common column names.

***MORE INFO* USING NATURALLEFTOUTERJOIN**

For more information on the NATURALLEFTOUTERJOIN function, see "NATURALLEFTOUTERJOIN Function (DAX)" at *https://msdn.microsoft.com/en-us/library/dn802527.aspx*.

DATATABLE

The DATATABLE function allows you to create calculated tables with data that you enter manually. Earlier in this book, you used the Enter Data feature of Power BI to enter data manually–DATATABLE provides an alternative.

At a minimum, DATATABLE takes three arguments: column name, data type, and list of values. The data types that you can choose are as follows:

- **BOOLEAN** True/False
- **CURRENCY** Fixed Decimal Number
- **DATETIME** Date/Time
- **DOUBLE** Decimal Number
- **INTEGER** Whole Number
- **STRING** Text

The final argument is a list of values in curly braces that resembles what you would type in M inside the #table construct. Create the Scale table using the DATATABLE function as follows:

Click here to view code image

```
Scale DataTable =
DATATABLE (
    "Scale", INTEGER,
    {
        { 1 },
        { 1000 },
        { 1000000 }
    }
)
```

Scale
1
1000
1000000

You can create a table with more than one column by listing all column names and data types in pairs:

Click here to view code image

```
Enriched Scale =
DATATABLE (
    "Scale", INTEGER,
    "Description", STRING,
    {
        { 1, "Normal" },
        { 1000, "Thousands" },
        { 1000000, "Millions" }
    }
)
```

Scale	Description
1	Normal
1000	Thousands
1000000	Millions

Note that all values in curly braces must be constants–you cannot use expressions inside DATATABLE. For instance, the following calculated table cannot be created:

Click here to view code image

```
Scale Wrong =
DATATABLE (
    "Scale", INTEGER,
    {
        { 1 + 0 },
        { 1000 },
        { 1000000 }
    }
)
```

The error message we get in this case is as follows: “The tuple at index ‘1’ from the table definition of the DATATABLE function does not have a constant expression in the column at index ‘1’.”

DAX also allows you to create anonymous tables without defining column names, though the syntax is slightly different from the DATATABLE syntax. After the equation operator, you also use curly braces, but inside them, you do not use curly braces again. If you define a one-column table, you list your values separated by a comma, and DAX will call the new column Value. You can also define multicolumn tables by listing values of each row in parenthesis; the parenthesis sets should be separated by commas as well. In this case, DAX will give the new columns names like Value1, Value2, and so on. Data types will also be defined automatically. For instance, we can create the following table:

```
Single-column table = { 1, 2 }
```

Value
1
2

The following is an example of an anonymous table with three columns:

Click here to view code image

```
Anonymous table =
{
    ( 1, "a", DATE ( 2018, 5, 1 ) ),
    ( 2, "b", DATE ( 2018, 5, 23 ) )
}
```

Value1	Value2	Value3
1	a	1/05/2018 12:00:00 AM
2	b	23/05/2018 12:00:00 AM

***MORE INFO* USING DATATABLE**

For more information on the DATATABLE function, see “DATATABLE Function” at *https://msdn.microsoft.com/en-us/library/mt674921.aspx*. You can read more about using

DATATABLE to create static tables in an article by Marco Russo, "Create Static Tables in DAX Using the DATATABLE Function" at: *https://www.sqlbi.com/articles/create-static-tables-in-dax-using-the-datatable-function/*.

Using variables in calculated tables

DAX variables can store scalar values, as well as tables. For example, rewrite one of the previous table expressions as follows:

Click here to view code image

```
Tables Var =
VAR Table1 =
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
VAR Table2 =
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
RETURN
    NATURALINNERJOIN ( Table1, Table2 )
```

While variables improve the readability of your code and potentially increase performance, it is important to understand that variables are evaluated only once in the context in which they are defined. This means that using CALCULATE or CALCULATETABLE on a variable has no effect. To illustrate this behavior, create the following calculated tables:

LISTING 2-14 Buying Groups

Click here to view code image

```
Buying Groups = VALUES ( Customer[Buying Group] )
```

Buying Group
N/A
Tailspin Toys
Wingtip Toys

LISTING 2-15 Buying Group filtered to one value

Click here to view code image

```
Wingtip Toys =
CALCULATETABLE (
    VALUES ( Customer[Buying Group] ),
    Customer[Buying Group] = "Wingtip Toys"
)
```

Buying Group
Wingtip Toys

Note that the following expression returns a table that is different from the one that you just reviewed in Listing 2-15:

Click here to view code image

```
Wingtip Toys VAR =
VAR BuyingGroups =
    VALUES ( Customer[Buying Group] )
RETURN
    CALCULATETABLE (
        BuyingGroups,
        Customer[Buying Group] = "Wingtip Toys"
    )
```

Buying Group
N/A
Tailspin Toys
Wingtip Toys

The resulting table is the same as in Listing 2-14 because the BuyingGroups variable is evaluated outside CALCULATETABLE, and then it becomes immutable. Therefore, putting the result of this variable in a different context makes no difference to it.

Variables in calculated tables can be used to create more readable code that is still complex. This allows you to save the time spent on creating calculated columns one by one; instead, you can define them in one calculated table expression. For instance, you can create a full calendar table in one expression using the GENERATE/ROW pattern developed by Marco Russo:

Click here to view code image

```
Calendar =
VAR Days =
    CALENDAR ( "2018-1-1", "2020-12-31" )
RETURN
    GENERATE (
        Days,
        VAR BaseDate = [Date]
        VAR MonthName = FORMAT ( BaseDate, "MMM" )
        VAR MonthNumber = MONTH ( BaseDate )
        VAR BaseYear = YEAR ( BaseDate )
        RETURN
            ROW (
                "Month", MonthName,
                "MonthNo", MonthNumber,
                "Year", BaseYear
            )
    )
```

Date	Month	MonthNo	Year
1/01/2018 12:00:00 AM	Jan	1	2018
2/01/2018 12:00:00 AM	Jan	1	2018
3/01/2018 12:00:00 AM	Jan	1	2018
…	…	…	…
31/12/2020 12:00:00 AM	Dec	12	2020

Note that in this case, you have multiple levels of VAR/RETURN constructs. At the top level, you are defining the Days variable, which stores

the table returned by the CALENDAR function. Use the Days variable in the outer RETURN expression. You have another VAR/RETURN construct, where you define more variables; this time, the variables hold scalar values, because GENERATE enables you to access the row context. The variables defined in the inner VAR/RETURN construct are not accessible in the outer construct.

> ***MORE INFO*** **USING GENERATE AND ROW TOGETHER**
>
> For more detailed discussion on using the GENERATE/ROW pattern as an alternative to ADDCOLUMNS, see Marco Russo's article, "Using GENERATE and ROW instead of ADDCOLUMNS in DAX" at *https://www.sqlbi.com/articles/using-generate-and-row-instead-of-addcolumns-in-dax/*.

Measures

An important limitation of calculated columns is that not all values can be calculated with them. For example, if you need to calculate the profit percentage, the calculated column formula might look as follows:

Click here to view code image

```
Net Profit % = DIVIDE ( Example[Net Profit], Example[Gross
Profit] )
```

While this formula calculates the profit percentage for each row of the table, it would be incorrect to use the values from this column in a visual, because they would show an arithmetic average at best. To illustrate the problem, consider the following two-row table called Example, shown in Table 2-13.

TABLE 2-13 Sample profit values in the Example table

Product	Gross Profit	Net Profit	Net Profit %
A	3,000	300	10%
B	2,000	1,000	50%

If you use the Net Profit % column from this table in a visual, you can only show 30%, the average of 10% and 50%. This is not correct; a correct value would be (300 + 1,000) / (3,000 + 2,000) = 26%. In other words, you need to sum all Net Profit values first, then divide the result by the sum of all Gross Profit values. While displaying this value in a calculated column is possible, this value will be incorrect as soon as you filter by Product. In this case, you need to create a measure.

To create a measure, you can select **Modeling** > **Calculations** > **New Measure**. This will create a measure in the currently selected table. Alternatively, you can right-click on a table in which you want to create a measure and select New Measure. Either option will open the formula bar where you can write your DAX formula. Once you are finished, you can either click on the tick icon to the left of the formula bar or press Enter. If you create a measure in the wrong table, you can move it by selecting the correct table by selecting **Modeling** > **Properties** > **Home Table**.

> ***MORE INFO*** **CREATING MEASURES**
>
> For a video overview on how to create measures, see "Create calculated measures" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/modeling#step-5*.

Measures aggregate columns and tables, and they always work in filter context. For this reason, there is no concept of current row in measures by default. A measure with the following formula cannot be created:

Click here to view code image

```
Net Profit % = DIVIDE ( Example[Net Profit], Example[Gross
Profit] )
```

Note that this is the same formula that works for a calculated column. This formula does not work in a measure because DAX does not know what it should do with the columns that you used. Because of this, any table or column used in a measure must be aggregated. The following functions are the most popular aggregation functions:

- SUM
- AVERAGE
- MIN

- MAX

For instance, you can create a measure that sums all Net Profit column values as follows:

```
Total Net Profit = SUM ( Example[Net Profit] )
```

The functions listed above always take one argument: column reference. If you want to reference two columns in an aggregation function, you need to use the iterator functions, which usually have X suffix. These are the most commonly used ones:

- SUMX
- AVERAGEX
- MINX
- MAXX

These functions always take two arguments: a table to iterate over and an expression to evaluate for each row in row context. The aggregation functions without X suffix are often syntactic sugar for their X-suffixed counterparts. The following two expressions are equivalent:

Click here to view code image

```
Total Net Profit = SUM ( Example[Net Profit] )

Total Net Profit = SUMX ( Example, Example[Net Profit] )
```

As mentioned above, the iterator functions can reference more than one column at a time. For example, in the following expression we iterate over the Sale table, and, for each row, multiply Quantity by Unit Price. Finally, sum all of the resulting values:

Click here to view code image

```
Gross Sales = SUMX ( Sale, Sale[Quantity] * Sale[Unit Price] )
```

Because iterators generate row context, all row functions, such as RELATED, can be used:

Click here to view code image

```
Full Price Sales =
SUMX (
    Sale,
    Sale[Quantity] * RELATED ( 'Stock Item'[Recommended Retail
```

```
Price] )
)
```

In the Example table above, you can create the weighted profit percentage measure with the following code:

Click here to view code image

```
Net Profit % =
DIVIDE (
    SUM ( Example[Net Profit] ),
    SUM ( Example[Gross Profit] )
)
```

While this measure works and displays the correct values, you can make the code more readable and easier to maintain by splitting the calculation into three measures: Total Net Profit, Total Gross Profit, and Net Profit %. Because you can reference other measures when you create a measure, you can first create the Total Net Profit and Total Gross Profit measures, then the Net Profit % one:

Click here to view code image

```
// Create these two measures first
Total Net Profit = SUM ( Example[Net Profit] )
Total Gross Profit = SUM ( Example[Gross Profit] )

// Create this measure last
Net Profit % = DIVIDE ( [Total Net Profit], [Total Gross Profit]
)
```

At this stage, the Net Profit and Gross Profit columns can be hidden. This practice is called fixing implicit measures; when you use a column in the values field well in a visual, an implicit measure is created using the default aggregation. When you write measures with DAX, you are creating explicit measures. Hiding the columns you aggregate in explicit measures is usually a good practice because it prevents user confusion over which field should be used.

> ***MORE INFO*** **DEFAULT SUMMARIZATION**
>
> Default aggregation, also known as default summarization, will be covered in Skill 2.5: “Create and format interactive visualizations.”

Measures vs. calculated columns

In many cases, the same values can be computed by using either a measure or a calculated column. Calculated columns are computed when they are defined and at data refresh time. Because they are materialized in a data model, they consume RAM and disk space. Measures, on the other hand, are calculated at query time, which means every time you interact with a visual, a measure recalculates its value. For this reason, measures consume CPU resources.

As you have seen, some values, such as weighted averages, cannot be computed in a calculated column, which leaves creating measures as the only option. At the same time, there are situations in which you should create a calculated column instead of a measure:

- Slicing or filtering by values: it is currently impossible to put a measure into a slicer. If you want to slice by newly created values, such as Price Category we created before, you must create a calculated column and not a measure.
- Writing CPU-intensive formulas: if your formula is very complex and it takes many seconds to compute its values, this may result in a poor user experience. In this case, it might be a good idea to precompute results in a calculated column or a calculated table, and then aggregate the results with a measure.

> ***NOTE*** **MEASURES AND CALCULATED COLUMNS**
>
> For a more detailed discussion on the difference between calculated columns and measures and which to use when, see “Calculated Columns and Measures in DAX” at *https://www.sqlbi.com/articles/calculated-columns-and-measures-in-dax/*.

Counting values in DAX

There are several functions in DAX with which you can count values:

- COUNT
- COUNTA
- COUNTAX

- COUNTBLANK
- COUNTROWS
- COUNTX
- DISTINCTCOUNT

One of the most frequently used functions is COUNTROWS, which we have already used before. The function takes one parameter: a table expression. The following measure counts the number of rows in the Sale table in the current filter context:

```
Sale Rows = COUNTROWS ( Sale )
```

Because you are not limited to physical tables, you can also count rows in tables calculated dynamically. The following measure returns the number of Calendar Year and Month combinations in which we had sales:

Click here to view code image

```
Years and Months with Sales =
COUNTROWS (
    SUMMARIZE (
        Sale,
        'Date'[Calendar Year],
        'Date'[Month]
    )
)
```

The COUNT function takes a column reference as the only argument and counts the number of non-blank values in a column. For example, the following measure counts the number of non-blank Delivery Date Key values in the Sale table:

Click here to view code image

```
Count DeliveryDate = COUNT ( Sale[Delivery Date Key] )
```

The limitation of COUNT function is that it cannot count Boolean values. To count the number of non-blank values regardless of data type, you can use the COUNTA function.

***MORE INFO* COUNT AND COUNTA**

For more information on the COUNT and COUNTA functions, see "COUNT Function (DAX)" at *https://msdn.microsoft.com/en-*

us/library/ee634791.aspx, and "COUNTA Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634956.aspx*.

The COUNTX and COUNTAX, the behavior of which correspond to COUNT and COUNTA, respectively, allow you to count non-blank expressions when iterating over a table. Both tables take two arguments: a table to iterate over and an expression to be evaluated in row context. For instance, the following measure counts the number of rows in the Sale table where either Invoice Date Key or Delivery Date Key is not blank:

Click here to view code image

```
CountX Invoice Delivery =
COUNTX (
    Sale,
    Sale[Invoice Date Key] + Sale[Delivery Date Key]
)
```

This measure returns the same results as COUNTROWS because in our data model, either Invoice Date Key or Delivery Date Key will always be not blank for any row.

***MORE INFO* COUNTX AND COUNTAX**

For more information on the COUNTX and COUNTAX functions, see "COUNTX Function (DAX)" at: *https://msdn.microsoft.com/en-us/library/ee634549.aspx*, and "COUNTAX Function (DAX)" at: *https://msdn.microsoft.com/en-us/library/ee634219.aspx*.

To count blank values in a column, you can use the COUNTBLANK function, which always received a column reference as its only parameter. You can count the number of blank Delivery Date Key values with the following measure:

```
CountBlank DeliveryDate = COUNTBLANK ( Sale[Delivery Date Key] )
```

***MORE INFO* COUNTBLANK**

For more information on the COUNTBLANK function, see "COUNTBLANK Function (DAX)" at:

https://msdn.microsoft.com/en-us/library/ee634230.aspx.

The sum of COUNT (and its equivalent COUNTA, COUNTX, and COUNTAX expressions) and COUNTBLANK will always give you the same result as COUNTROWS. In Table 2-14, you can see that the sum of Count DeliveryDate and CountBlank DeliveryDate is equal to Sale Rows:

TABLE 2-14 Sale Rows, Count DeliveryDate, and CountBlank DeliveryDate sliced by Calendar Year Label

Calendar Year Label	Sale Rows	Count DeliveryDate	CountBlank DeliveryDate
	284		284
CY2013	60678	60678	
CY2014	65957	65957	
CY2015	71828	71828	
CY2016	29519	29519	
Total	228266	227982	284

If you want to count the number of distinct values in a column, you can either use a combination of COUNTROWS and DISTINCT, or you can use DISTINCTCOUNT, which takes a column reference as its only parameter. For instance, the following measure returns the number of Stock Item Key values that have been sold at least once:

```
Sold StockItems = DISTINCTCOUNT ( Sale[Stock Item Key] )
```

Note that this measure returns different results compared to a measure that counts the distinct Stock Item Key values in the Stock Item table when sliced by Calendar Year Label or Brand. Note also that when a column contains unique values only, you can safely use COUNTROWS instead. The following two measures produce equivalent results:

Click here to view code image

```
Distinct StockItems = DISTINCTCOUNT( 'Stock Item'[Stock Item
Key] )
```

```
Distinct StockItems = COUNTROWS ( 'Stock Item' )
```

TABLE 2-15 Sold StockItems and Distinct StockItems sliced by Calendar Year Label

Calendar Year Label	Sold StockItems	Distinct StockItems
	164	672
CY2013	219	672
CY2014	219	672
CY2015	219	672
CY2016	228	672
Total	228	672

We see the same value for Distinct StockItems in this table because filter context from the Date table does not pass to the Stock Item table. Note that because of the nature of distinct counts, the results might not always be additive.

TABLE 2-16 Sold StockItems and Distinct StockItems sliced by Brand

Brand	Sold StockItems	Distinct StockItems
N/A	210	605
Northwind	18	67
Total	228	672

In Table 2-16, the difference between Sold StockItems and Distinct StockItems is that some stock items do not sell at all, even though we have them in our data model.

Using CALCULATE in measures

The CALCULATE function, covered earlier in the chapter, is most often used in measures. Its syntax is identical to CALCULATETABLE except it received a scalar expression as the first parameter instead of a table expression. Using CALCULATE in a measure without filters is useless

because every measure has an implicit CALCULATE wrapped around it. To illustrate this effect, look at the Scale table and create the following calculated column and measure in it:

Click here to view code image

```
// Calculated column
Sum Column = SUM ( Scale[Scale] )

// Measure
Sum Measure = SUM ( Scale[Scale] )
```

Note that the formula used in two expressions is the same. As expected, Sum Column shows the same value for each row. Now, create the following calculated column in the Scale table:

```
Sum Measure Column = [Sum Measure]
```

In this calculated column, you are referencing the Sum Measure only. Note the square brackets around the measure name; this is the standard way to reference measures in DAX. Now, the Scale table should look like the one in Figure 2.32.

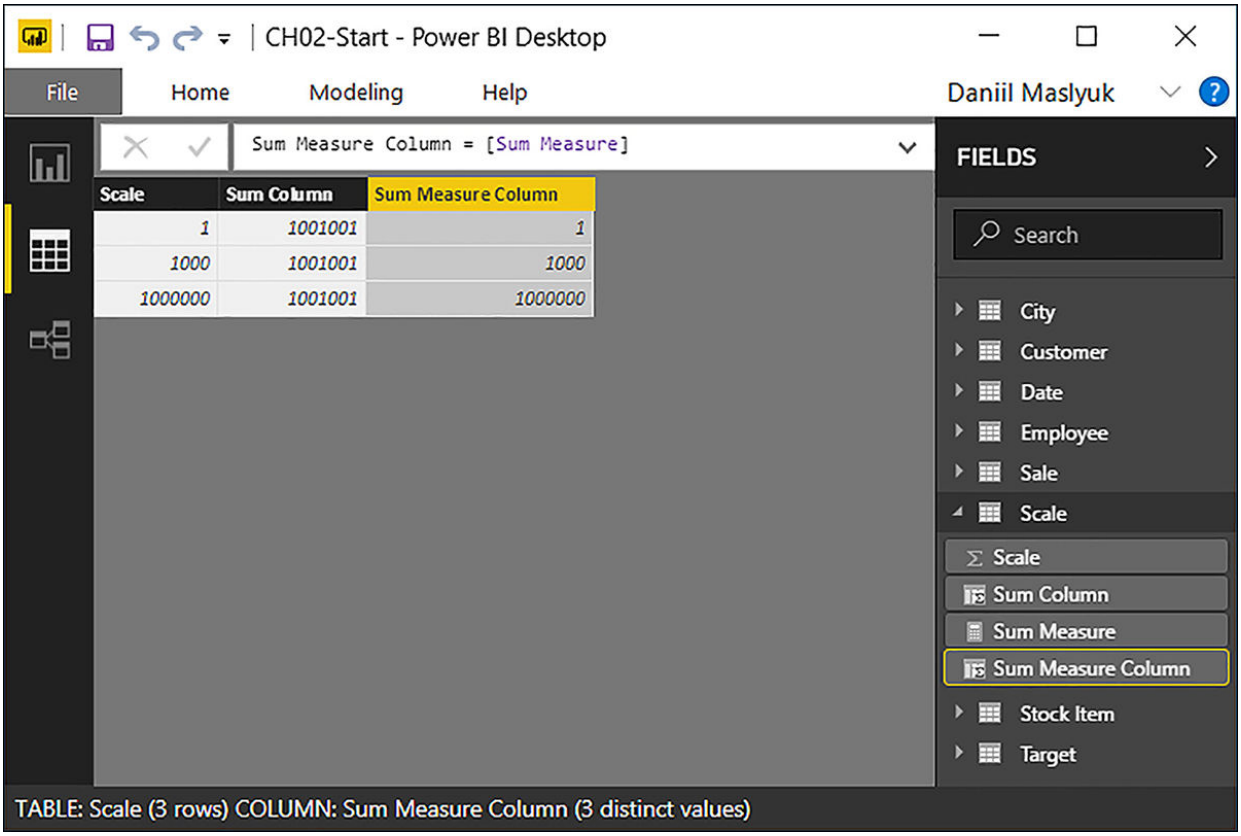


Figure 2.32 The Scale table with Sum Column and Sum Measure Column created

Even though the formulas of Sum Measure and Sum Column are identical, the values in the Sum Measure Column calculated column are different because each measure has an implicit CALCULATE wrapped around it. In other words, you canrewrite the Sum Measure Column expression as follows:

```
Sum Measure Column = CALCULATE ( SUM ( Scale[Scale] ) )
```

For this reason, CALCULATE becomes useful inside measures when you want to change the filter context. As with CALCULATETABLE, you can pass Boolean expressions or tables as filter parameters in CALCULATE. For example, you can create the All-time Profit measure, which displays the all-time profit made, regardless of the selections made in the Date table:

Click here to view code image

```
// Create the Total Profit measure first
Total Profit = SUM ( Sale[Profit] )
```

```
// Create the All-time Profit measure that references Total Profit
All-time Profit = CALCULATE ( [Total Profit], ALL ( 'Date' ) )
```

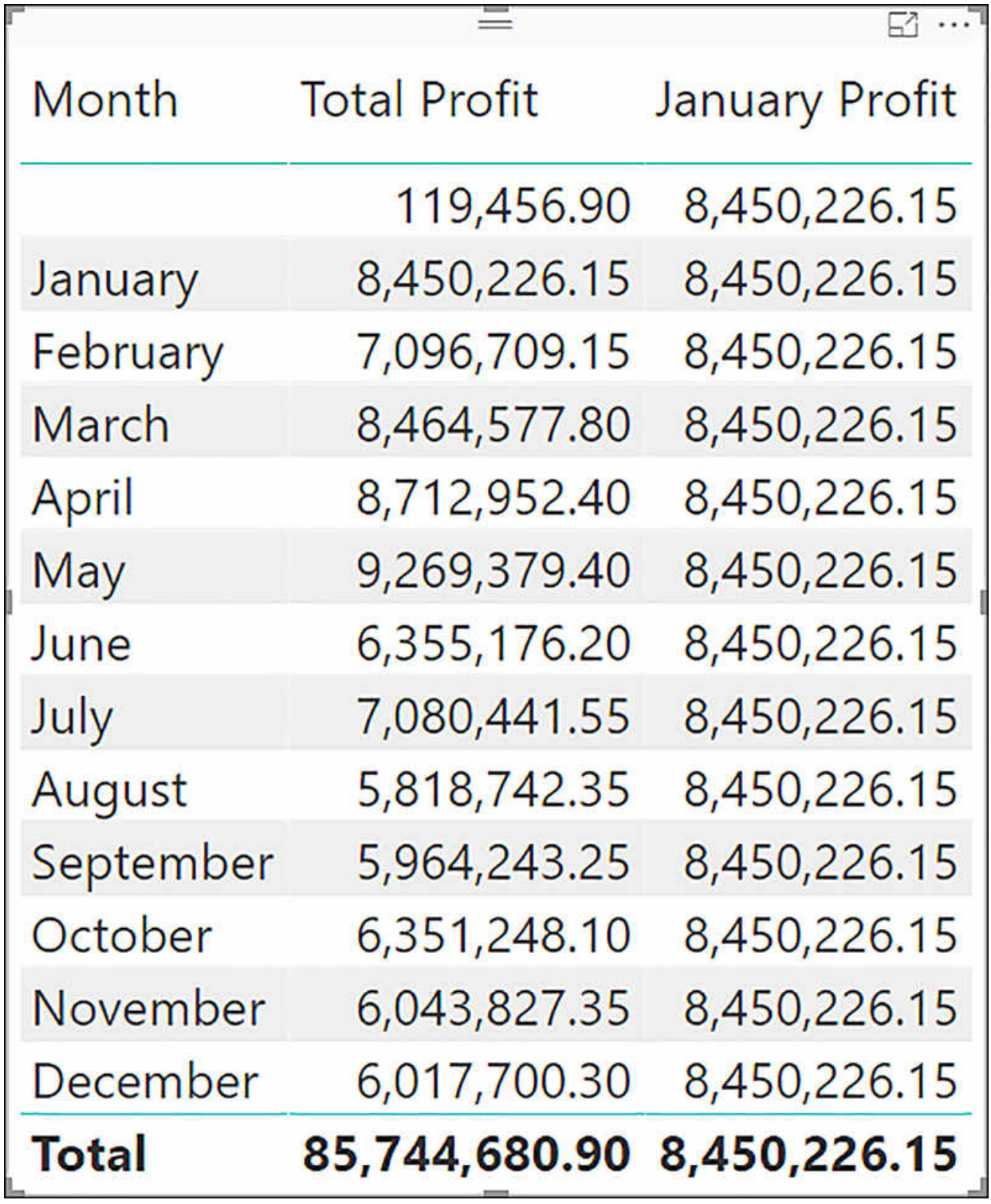
If you use this measure in a table visual alongside Total Profit, you will see results similar to Figure 2.33.

Month	Total Profit	All-time Profit
	119,456.90	85,744,680.90
January	8,450,226.15	85,744,680.90
February	7,096,709.15	85,744,680.90
March	8,464,577.80	85,744,680.90
April	8,712,952.40	85,744,680.90
May	9,269,379.40	85,744,680.90
June	6,355,176.20	85,744,680.90
July	7,080,441.55	85,744,680.90
August	5,818,742.35	85,744,680.90
September	5,964,243.25	85,744,680.90
October	6,351,248.10	85,744,680.90
November	6,043,827.35	85,744,680.90
December	6,017,700.30	85,744,680.90
Total	**85,744,680.90**	**85,744,680.90**

Figure 2.33 All-time Profit alongside Total Profit sliced by Month

As expected, the measure shows the same value for each row, which is the same as the grand total of Total Profit. While ALL removes the filter, we can also set the new filter context at the same time. For example, here we show

profit made in January regardless of the selected Month. Figure 2.34 shows the expected result.



Month	Total Profit	January Profit
	119,456.90	8,450,226.15
January	8,450,226.15	8,450,226.15
February	7,096,709.15	8,450,226.15
March	8,464,577.80	8,450,226.15
April	8,712,952.40	8,450,226.15
May	9,269,379.40	8,450,226.15
June	6,355,176.20	8,450,226.15
July	7,080,441.55	8,450,226.15
August	5,818,742.35	8,450,226.15
September	5,964,243.25	8,450,226.15
October	6,351,248.10	8,450,226.15
November	6,043,827.35	8,450,226.15
December	6,017,700.30	8,450,226.15
Total	**85,744,680.90**	**8,450,226.15**

FIGURE 2.34 January Profit measure alongside Total Profit sliced by Month

When passing Boolean expressions as filters in CALCULATE, they are transformed into table filters with FILTER and ALL combined. Because you only have the Month column visible, and you sort it by the Calendar Month Number column, the following formula will not work correctly. You can see the values it returns in Figure 2.35.

Click here to view code image

```
January Profit Wrong = CALCULATE ( [Total Profit], 'Date'[Month]
= "January" )
```

Month	Total Profit	January Profit Wrong
	119,456.90	
January	8,450,226.15	8,450,226.15
February	7,096,709.15	
March	8,464,577.80	
April	8,712,952.40	
May	9,269,379.40	
June	6,355,176.20	
July	7,080,441.55	
August	5,818,742.35	
September	5,964,243.25	
October	6,351,248.10	
November	6,043,827.35	
December	6,017,700.30	
Total	**85,744,680.90**	**8,450,226.15**

FIGURE 2.35 January Profit Wrong visualized

The formula does not return the expected results because when slicing by Month, Calendar Month Number is also part of filter context because it sorts Month. To make the formula work correctly, either list conditions for both columns, or iterate over a table that includes both columns. Either of the following expressions returns the expected result:

Click here to view code image

```
// Including conditions for both columns
January Profit =
CALCULATE (
    [Total Profit],
    'Date'[Month] = "January",
    'Date'[Calendar Month Number] = 1
)

// Boolean expressions internally converted to table filters
January Profit =
CALCULATE (
    [Total Profit],
    FILTER (
        ALL ( 'Date'[Month] ),
        'Date'[Month] = "January"
    ),
    FILTER (
        ALL ( 'Date'[Calendar Month Number] ),
        'Date'[Calendar Month Number] = 1
    )
)

// Alternative approach: iterating over a table that includes
both columns
January Profit =
CALCULATE (
    [Total Profit],
    FILTER (
        ALL ( 'Date'[Month], 'Date'[Calendar Month Number] ),
        'Date'[Month] = "January"
    )
)
```

There is another function in the ALL family of functions, which we have not reviewed yet: ALLSELECTED. This function received one optional argument: either a table or a column reference. According to the official documentation, this function removes filter context from rows and columns of a table, while retaining all other context filters or explicit filters. In Power BI, rows and columns of a table can be extended to mean axes, legends, and so on. To review the effect of ALLSELECTED, follow these steps:

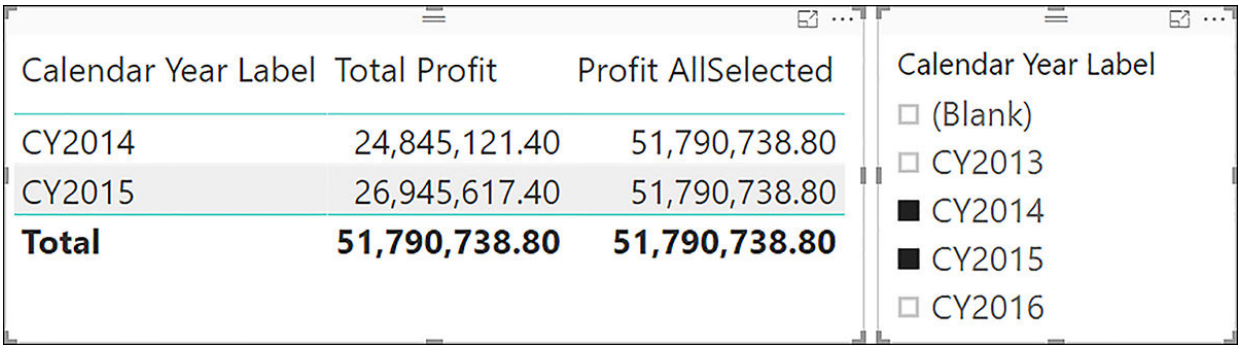
1. Create a new measure with the following formula:

   Click here to view code image

   ```
   Profit AllSelected = CALCULATE ( [Total Profit], ALLSELECTED
   ( 'Date' ) )
   ```

2. Create a table visual with the following fields:
   - Calendar Year Label
   - Total Profit
   - Profit AllSelected
3. Create a slicer with Calendar Year Label as its field.
4. Hold the Ctrl key and select CY2014 and CY2015.

At this stage, you should see figures like those shown in Figure 2.36.



Calendar Year Label	Total Profit	Profit AllSelected
CY2014	24,845,121.40	51,790,738.80
CY2015	26,945,617.40	51,790,738.80
Total	**51,790,738.80**	**51,790,738.80**

FIGURE 2.36 Profit AllSelected used in a table

Note the values displayed by the Profit AllSelected measure; they are the same as the grand total of Total Profit in this table. If you hold the Ctrl key and select CY2016 as well, the values will change to the new grand total.

> ***MORE INFO* SLICERS IN POWER BI**
>
> For a video overview and more information on slicers in Power BI, including formatting options, see "Slicers in Power BI service (Tutorial)" at: *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-slicers*.

Because the only parameter of ALLSELECTED is optional, you can use it with no parameters as follows:

Click here to view code image

```
Profit AllSelected = CALCULATE ( [Total Profit], ALLSELECTED ()
)
```

Used in this way, ALLSELECTED will consider filters from the entire data model, not just one table or column.

***MORE INFO* ALLSELECTED**

To learn more about the ALLSELECTED function, you can refer to the official documentation article, "ALLSELECTED Function (DAX)" at: *https://msdn.microsoft.com/en-us/library/gg492186.aspx*.

For an exact explanation of how ALLSELECTED works, see an article by Alberto Ferrari, "Understanding ALLSELECTED" at: *https://www.sqlbi.com/articles/understanding-allselected/*.

Time Intelligence

Time Intelligence in DAX is an umbrella term that often refers to calculations that span over predefined periods of time. DAX has more than 30 built-in functions to handle Time Intelligence. An example of using Time Intelligence is the comparison of different periods—this year versus last year, for instance.

Most Time Intelligence functions receive a date column as a parameter and return a table that can be used as a filter in CALCULATE, while a small group of functions return scalar values. The functions that return scalar values are all shorthand and can be rewritten using CALCULATE and one of the functions that return a table.

For the Time Intelligence functions to work correctly, you must have a date table, which is also known as a calendar table. The table should be similar to the Date table from the Wide World Importers data model, where you have a row for each date between the earliest and latest dates in your data model with no gaps. If your data source does not contain such a table, you can create one yourself. We reviewed the CALENDAR and CALENDARAUTO functions earlier in the chapter.

Another requirement for the calendar table is to be part of a one-to-many relationship with a column of type date. This way, the Time Intelligence functions will work without needing modifications.

***MORE INFO* TIME INTELLIGENCE IN POWER BI DESKTOP**

Data models do not always contain date columns of type date. Sometimes, dates can be stored as integers such as: 20180501 for 1 May 2018. If your data model has integers instead of dates, you have several options:

- Add ALL ('Date') filter to each expression
- Add date columns to your data model
- Create a dummy fact table

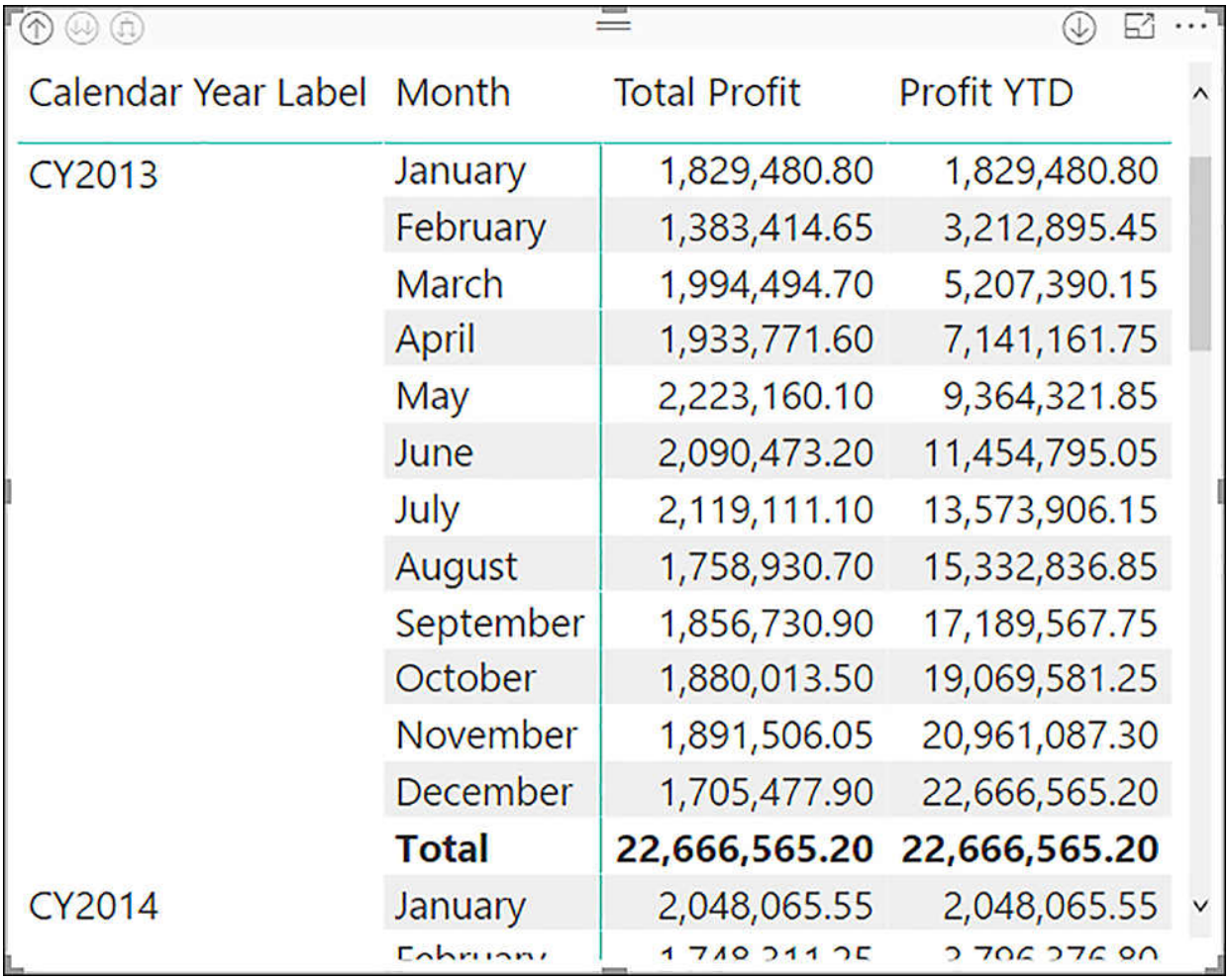
The last approach provides the least arduous way to make sure your Time Intelligence functions will work correctly. For a detailed discussion on how to make sure the Time Intelligence functions work in Power BI, see an article by Marco Russo, "Time Intelligence in Power BI Desktop" at: *https://www.sqlbi.com/articles/time-intelligence-in-power-bi-desktop/*.

Once your calendar table satisfies the requirements, you can use the Time Intelligence functions correctly. For example, to calculate year-to-date profit, you can use the DATESYTD function and write the following formula:

Click here to view code image

```
Profit YTD = CALCULATE ( [Total Profit], DATESYTD ( 'Date'[Date]
) )
```

If you use this measure alongside the Total Profit measure in a matrix visual and add Calendar Year Label and Month as values, you can see a visual like in Figure 2.37.



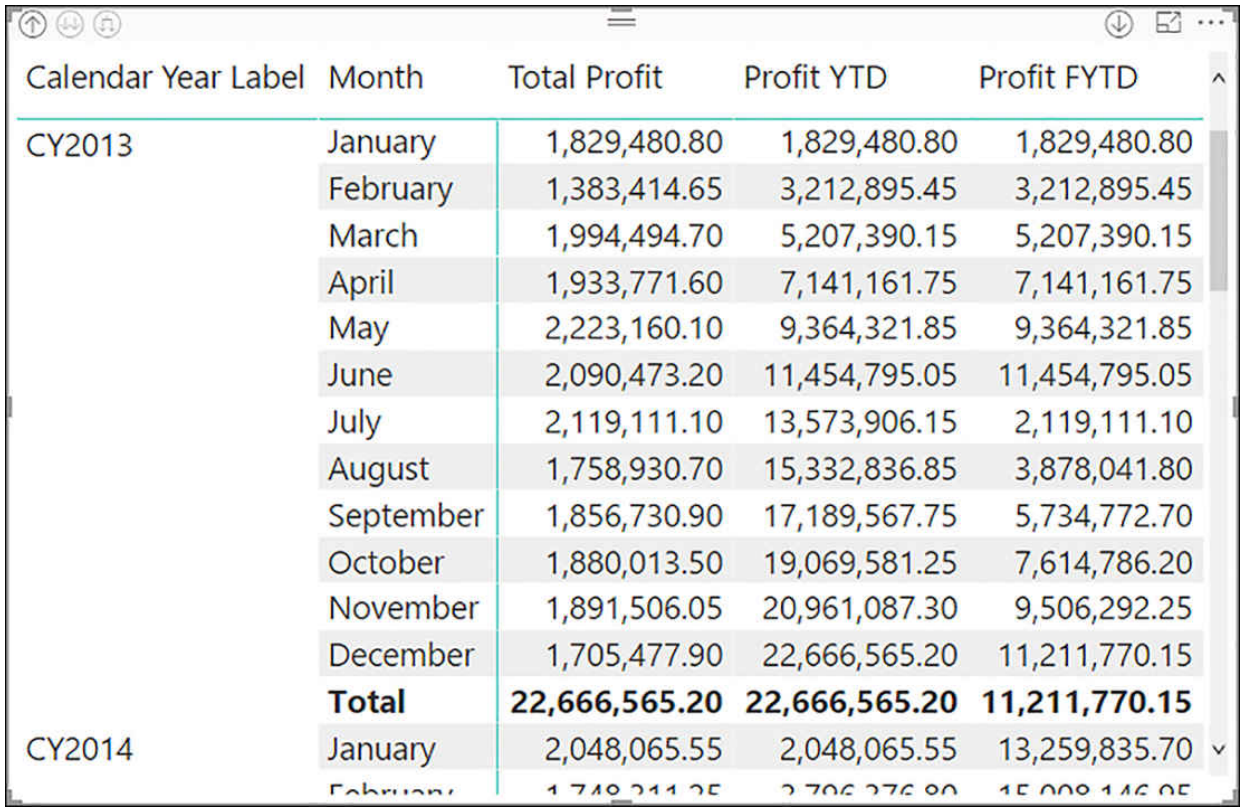
Calendar Year Label	Month	Total Profit	Profit YTD
CY2013	January	1,829,480.80	1,829,480.80
	February	1,383,414.65	3,212,895.45
	March	1,994,494.70	5,207,390.15
	April	1,933,771.60	7,141,161.75
	May	2,223,160.10	9,364,321.85
	June	2,090,473.20	11,454,795.05
	July	2,119,111.10	13,573,906.15
	August	1,758,930.70	15,332,836.85
	September	1,856,730.90	17,189,567.75
	October	1,880,013.50	19,069,581.25
	November	1,891,506.05	20,961,087.30
	December	1,705,477.90	22,666,565.20
	Total	**22,666,565.20**	**22,666,565.20**
CY2014	January	2,048,065.55	2,048,065.55

FIGURE 2.37 Profit YTD shown alongside Total Profit and sliced by Calendar Year Label and Month

Note how the profit amount is being added month by month from January to December 2013, and then it is starts at January 2014 again. The DATESYTD has an optional second parameter, which is the year-end date. With this parameter, you can specify a custom year end date such as “30-6” or “6-30,” depending on your locale. This option is often used for calculations involving fiscal or financial years. When omitted, the default option is 31 December. For instance, the following measure calculates year-to-date profit for the year ending on 30 June. You can see the values it returns in Figure 2.38 below.

Click here to view code image

```
Profit FYTD = CALCULATE ( [Total Profit], DATESYTD (
'Date'[Date], "30-6" ) )
```



Calendar Year Label	Month	Total Profit	Profit YTD	Profit FYTD
CY2013	January	1,829,480.80	1,829,480.80	1,829,480.80
	February	1,383,414.65	3,212,895.45	3,212,895.45
	March	1,994,494.70	5,207,390.15	5,207,390.15
	April	1,933,771.60	7,141,161.75	7,141,161.75
	May	2,223,160.10	9,364,321.85	9,364,321.85
	June	2,090,473.20	11,454,795.05	11,454,795.05
	July	2,119,111.10	13,573,906.15	2,119,111.10
	August	1,758,930.70	15,332,836.85	3,878,041.80
	September	1,856,730.90	17,189,567.75	5,734,772.70
	October	1,880,013.50	19,069,581.25	7,614,786.20
	November	1,891,506.05	20,961,087.30	9,506,292.25
	December	1,705,477.90	22,666,565.20	11,211,770.15
	Total	**22,666,565.20**	**22,666,565.20**	**11,211,770.15**
CY2014	January	2,048,065.55	2,048,065.55	13,259,835.70

FIGURE 2.38 Profit FYTD shown alongside Profit YTD and Total Profit

Note how the calculation of year-to-date profit now starts over in July instead of January.

DATESYTD also has two sister functions: DATESMTD and DATESQTD, which return month-to-date and quarter-to-date date tables, respectively. Both functions always receive one parameter only, with no optional parameters.

Because DATESMTD, DATESQTD, and DATESYTD functions are almost always used as filters for CALCULATE expression, there are three functions that simplify writing formulas with these functions: TOTALMTD, TOTALQTD, and TOTALYTD. For example, you can rewrite the Profit FYTD formula as follows:

Click here to view code image

```
Profit FYTD = TOTALYTD ( [Total Profit], 'Date'[Date], "30-6" )
```

The three functions receive two mandatory arguments: a scalar expression and the calendar table date column. The optional third parameter can be used to pass an additional filter. TOTALYTD can also receive an optional fourth

parameter, the year-end date. When no third parameter is specified, the year-end date can be used as the third parameter.

***MORE INFO* PERIOD TO DATE FUNCTIONS**

For more information on and examples of usage of the period to date functions, see:

- "DATESMTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634359.aspx*
- "DATESQTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634901.aspx*
- "DATESYTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634221.aspx*
- "TOTALMTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634560.aspx*
- "TOTALQTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634579.aspx*
- "TOTALYTD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634400.aspx*

With DAX, it is possible to calculate semi-additive measures such as opening and closing balances. For these purposes, there are monthly, quarterly, and yearly functions for both opening and closing balances:

- OPENINGBALANCEMONTH
- OPENINGBALANCEQUARTER
- OPENINGBALANCEYEAR
- CLOSINGBALANCEMONTH
- CLOSINGBALANCEQUARTER
- CLOSINGBALANCEYEAR

Each of the six functions received two required parameters: a scalar expression and the date column of a calendar table. A filter can be passed as the optional third parameter. Also, the yearly functions can receive an optional fourth parameter specifying the year-end date.

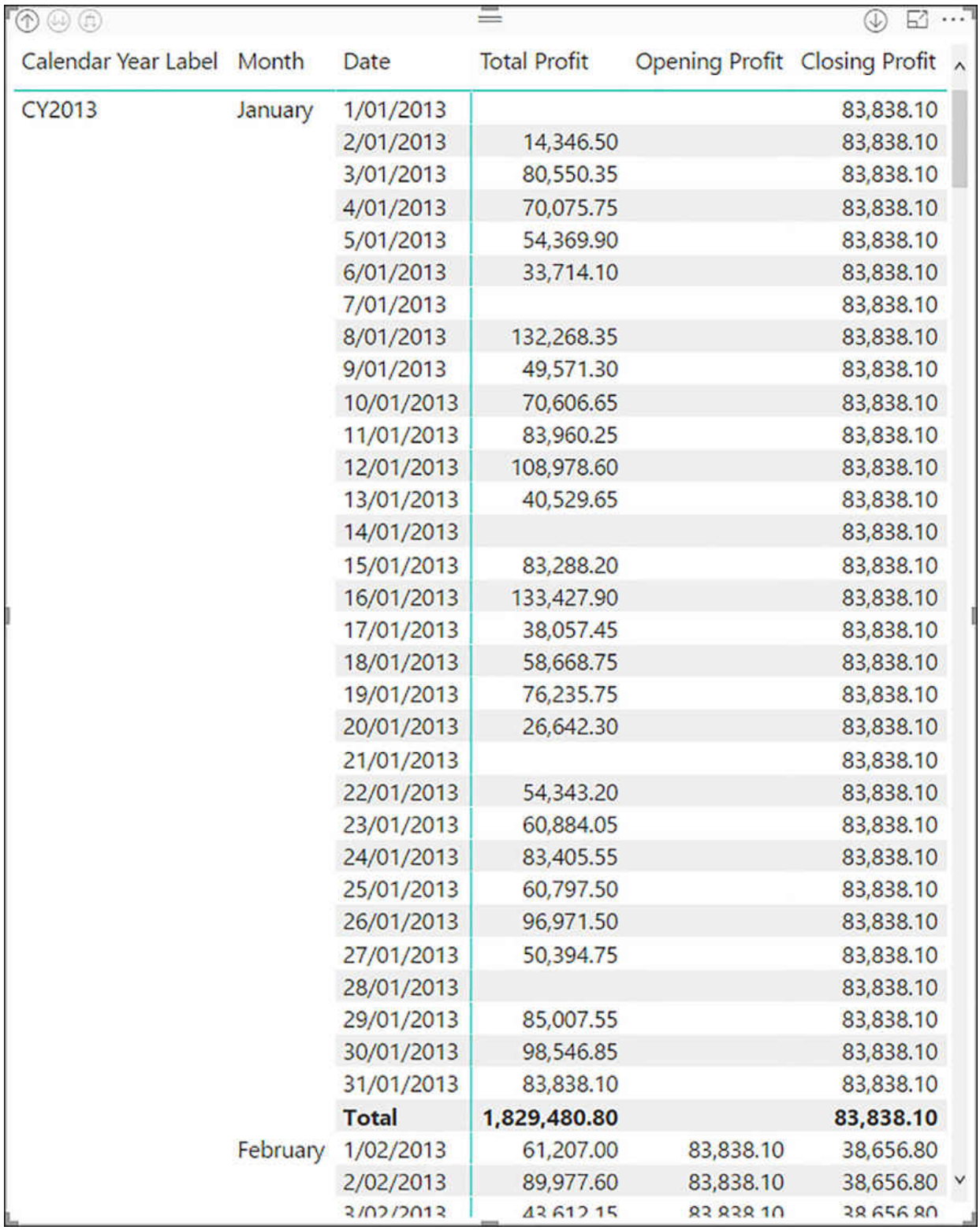
For review purposes, calculate the opening and closing month balance of profit as follows, even if the measures makes no sense financially.

Click here to view code image

```
Opening Profit = OPENINGBALANCEMONTH ( [Total Profit],
'Date'[Date] )

Closing Profit = CLOSINGBALANCEMONTH ( [Total Profit],
'Date'[Date] )
```

The OPENINGBALANCEMONTH function calculates the scalar value used as the first parameter for the last day of the previous month. In general, the opening balance functions return the same values as closing balance functions for the previous month. For example, the opening monthly balance for May 2018 will be the same as the closing monthly balance for April 2018. Both measures are shown in Figure 2.39.



Calendar Year Label	Month	Date	Total Profit	Opening Profit	Closing Profit
CY2013	January	1/01/2013			83,838.10
		2/01/2013	14,346.50		83,838.10
		3/01/2013	80,550.35		83,838.10
		4/01/2013	70,075.75		83,838.10
		5/01/2013	54,369.90		83,838.10
		6/01/2013	33,714.10		83,838.10
		7/01/2013			83,838.10
		8/01/2013	132,268.35		83,838.10
		9/01/2013	49,571.30		83,838.10
		10/01/2013	70,606.65		83,838.10
		11/01/2013	83,960.25		83,838.10
		12/01/2013	108,978.60		83,838.10
		13/01/2013	40,529.65		83,838.10
		14/01/2013			83,838.10
		15/01/2013	83,288.20		83,838.10
		16/01/2013	133,427.90		83,838.10
		17/01/2013	38,057.45		83,838.10
		18/01/2013	58,668.75		83,838.10
		19/01/2013	76,235.75		83,838.10
		20/01/2013	26,642.30		83,838.10
		21/01/2013			83,838.10
		22/01/2013	54,343.20		83,838.10
		23/01/2013	60,884.05		83,838.10
		24/01/2013	83,405.55		83,838.10
		25/01/2013	60,797.50		83,838.10
		26/01/2013	96,971.50		83,838.10
		27/01/2013	50,394.75		83,838.10
		28/01/2013			83,838.10
		29/01/2013	85,007.55		83,838.10
		30/01/2013	98,546.85		83,838.10
		31/01/2013	83,838.10		83,838.10
		Total	**1,829,480.80**		**83,838.10**
	February	1/02/2013	61,207.00	83,838.10	38,656.80
		2/02/2013	89,977.60	83,838.10	38,656.80
		3/02/2013	43,612.15	83,838.10	38,656.80

Figure 2.39 Opening Profit and Closing Profit shown alongside Total Profit

***MORE INFO* OPENING AND CLOSING BALANCE FUNCTIONS**

For more information on the opening and closing balance functions, see:

- "OPENINGBALANCEMONTH Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634224.aspx*
- "OPENINGBALANCEQUARTER Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634931.aspx*
- "OPENINGBALANCEYEAR Function (DAX)" a: *https://msdn.microsoft.com/en-us/library/ee634886.aspx*
- "CLOSINGBALANCEMONTH Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634215.aspx*
- "CLOSINGBALANCEQUARTER Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634876.aspx*
- "CLOSINGBALANCEYEAR Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634562.aspx*

The opening and closing balance functions we have just reviewed return scalar values. There are DAX functions that return table functions for beginnings and ends of periods:

- STARTOFMONTH
- STARTOFQUARTER
- STARTOFYEAR
- ENDOFMONTH
- ENDOFQUARTER
- ENDOFYEAR

Each function in the list above receives one required parameter: the date column from a calendar table. As before, the yearly functions can also receive a year-end date as an optional parameter. For example, you can rewrite the Closing Profit measure using the ENDOFMONTH function as follows:

Click here to view code image

```
Closing Profit = CALCULATE ( [Total Profit], ENDOFMONTH (
'Date'[Date] ) )
```

The Opening Profit measure cannot be rewritten using only CALCULATE and STARTOFMONTH. Use a function that can shift dates because the opening balance of a measure is its closing balance for the previous month. The most often used function is DATEADD, which receives exactly three arguments: the dates column of a calendar table, number of intervals, and the interval. The interval can be one of the following:

- DAY
- MONTH
- QUARTER
- YEAR

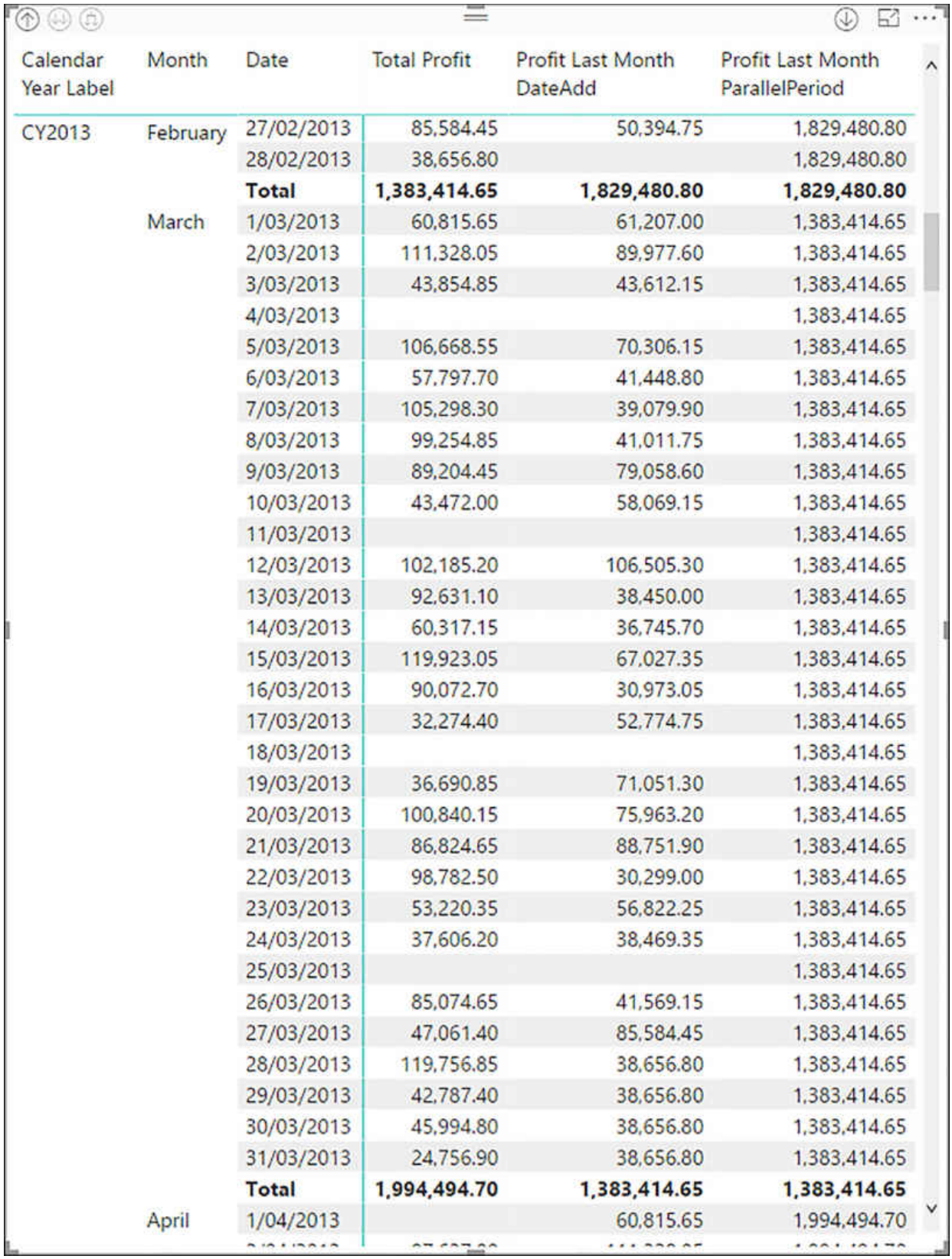
DATEADD is not the only function that can shift dates. There is a similar function, PARALLELPERIOD, which receives the same arguments and DATEADD. However, PARALLELPERIOD cannot receive DAY as the fourth parameter. The difference between the two functions is that DATEADD shifts dates for each date in the current filter context, while PARALLELPERIOD returns a full parallel period as a result. To illustrate the difference between the two functions, create the following two functions:

Click here to view code image

```
Profit Last Month DateAdd =
CALCULATE (
    [Total Profit],
    DATEADD ( 'Date'[Date], -1, MONTH )
)

Profit Last Month ParallelPeriod =
CALCULATE (
    [Total Profit],
    PARALLELPERIOD ( 'Date'[Date], -1, MONTH )
)
```

The two measures are shown side by side in Figure 2.40.



Calendar Year Label	Month	Date	Total Profit	Profit Last Month DateAdd	Profit Last Month ParallelPeriod
CY2013	February	27/02/2013	85,584.45	50,394.75	1,829,480.80
		28/02/2013	38,656.80		1,829,480.80
		Total	**1,383,414.65**	**1,829,480.80**	**1,829,480.80**
	March	1/03/2013	60,815.65	61,207.00	1,383,414.65
		2/03/2013	111,328.05	89,977.60	1,383,414.65
		3/03/2013	43,854.85	43,612.15	1,383,414.65
		4/03/2013			1,383,414.65
		5/03/2013	106,668.55	70,306.15	1,383,414.65
		6/03/2013	57,797.70	41,448.80	1,383,414.65
		7/03/2013	105,298.30	39,079.90	1,383,414.65
		8/03/2013	99,254.85	41,011.75	1,383,414.65
		9/03/2013	89,204.45	79,058.60	1,383,414.65
		10/03/2013	43,472.00	58,069.15	1,383,414.65
		11/03/2013			1,383,414.65
		12/03/2013	102,185.20	106,505.30	1,383,414.65
		13/03/2013	92,631.10	38,450.00	1,383,414.65
		14/03/2013	60,317.15	36,745.70	1,383,414.65
		15/03/2013	119,923.05	67,027.35	1,383,414.65
		16/03/2013	90,072.70	30,973.05	1,383,414.65
		17/03/2013	32,274.40	52,774.75	1,383,414.65
		18/03/2013			1,383,414.65
		19/03/2013	36,690.85	71,051.30	1,383,414.65
		20/03/2013	100,840.15	75,963.20	1,383,414.65
		21/03/2013	86,824.65	88,751.90	1,383,414.65
		22/03/2013	98,782.50	30,299.00	1,383,414.65
		23/03/2013	53,220.35	56,822.25	1,383,414.65
		24/03/2013	37,606.20	38,469.35	1,383,414.65
		25/03/2013			1,383,414.65
		26/03/2013	85,074.65	41,569.15	1,383,414.65
		27/03/2013	47,061.40	85,584.45	1,383,414.65
		28/03/2013	119,756.85	38,656.80	1,383,414.65
		29/03/2013	42,787.40	38,656.80	1,383,414.65
		30/03/2013	45,994.80	38,656.80	1,383,414.65
		31/03/2013	24,756.90	38,656.80	1,383,414.65
		Total	**1,994,494.70**	**1,383,414.65**	**1,383,414.65**
	April	1/04/2013		60,815.65	1,994,494.70

FIGURE 2.40 Profit Last Month DateAdd and Profit Last Month ParallelPeriod used in a matrix visual

Note how the two measures display the same results at the month level, but the values are different at date level: DATEADD displays a different value for each date, while PARALLELPERIOD shows the same value for each date within one month. Also, note that the value of the DATEADD measure is the same from 28 to 31 March 2013 because the last date in February 2013 is 28 February 2013, and dates from 28 to 31 March 2013 are all treated as the last day of the month when compared to the previous month.

In addition to DATEADD and PARALLELPERIOD, there are the following functions with descriptive names that shift by a predefined period:

- SAMEPERIODLASTYEAR: same as DATEADD (Dates, -1, YEAR)
- PREVIOUSDAY
- PREVIOUSMONTH
- PREVIOUSQUARTER
- PREVIOUSYEAR
- NEXTDAY
- NEXTMONTH
- NEXTQUARTER
- NEXTYEAR

This list of functions receives the date column of a calendar table as the only required parameter. PREVIOUSYEAR and NEXTYEAR can also receive the year-end date as the optional second parameter.

You can combine some Time Intelligence functions. For example, using either DATEADD or PARALLELPERIOD, you can rewrite the Opening Profit measure as follows:

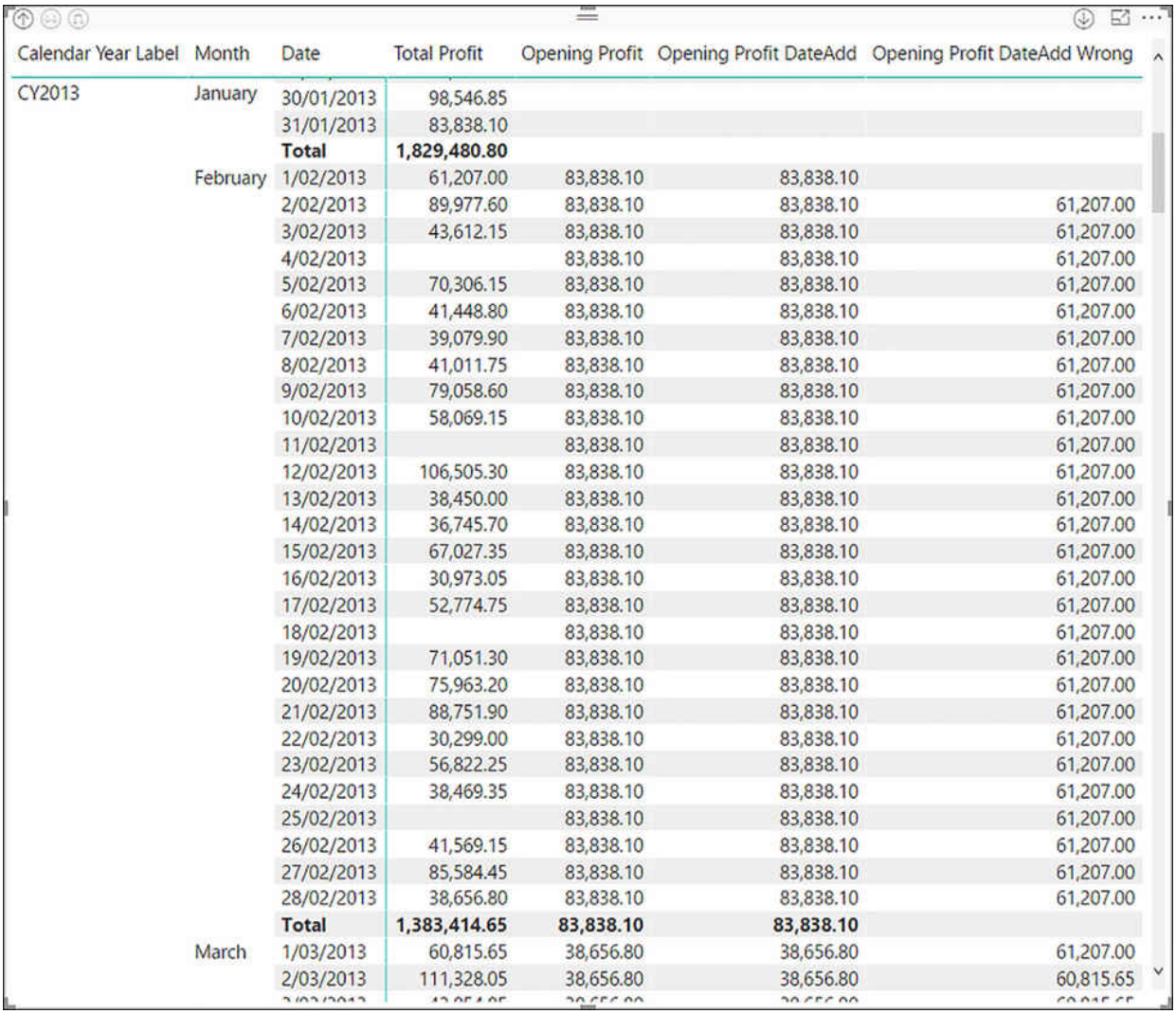
Click here to view code image

```
Opening Profit DateAdd =
CALCULATE (
    [Total Profit],
    DATEADD ( STARTOFMONTH ( 'Date'[Date] ), -1, DAY )
)
```

Note that the order in which you nest functions matters. While the following measure works, it returns incorrect results, as seen in Figure 2.41:

Click here to view code image

```
Opening Profit DateAdd Wrong =
CALCULATE (
    [Total Profit],
    STARTOFMONTH ( DATEADD ( 'Date'[Date], -1, DAY ) )
)
```



Calendar Year Label	Month	Date	Total Profit	Opening Profit	Opening Profit DateAdd	Opening Profit DateAdd Wrong
CY2013	January	30/01/2013	98,546.85			
		31/01/2013	83,838.10			
		Total	**1,829,480.80**			
	February	1/02/2013	61,207.00	83,838.10	83,838.10	
		2/02/2013	89,977.60	83,838.10	83,838.10	61,207.00
		3/02/2013	43,612.15	83,838.10	83,838.10	61,207.00
		4/02/2013		83,838.10	83,838.10	61,207.00
		5/02/2013	70,306.15	83,838.10	83,838.10	61,207.00
		6/02/2013	41,448.80	83,838.10	83,838.10	61,207.00
		7/02/2013	39,079.90	83,838.10	83,838.10	61,207.00
		8/02/2013	41,011.75	83,838.10	83,838.10	61,207.00
		9/02/2013	79,058.60	83,838.10	83,838.10	61,207.00
		10/02/2013	58,069.15	83,838.10	83,838.10	61,207.00
		11/02/2013		83,838.10	83,838.10	61,207.00
		12/02/2013	106,505.30	83,838.10	83,838.10	61,207.00
		13/02/2013	38,450.00	83,838.10	83,838.10	61,207.00
		14/02/2013	36,745.70	83,838.10	83,838.10	61,207.00
		15/02/2013	67,027.35	83,838.10	83,838.10	61,207.00
		16/02/2013	30,973.05	83,838.10	83,838.10	61,207.00
		17/02/2013	52,774.75	83,838.10	83,838.10	61,207.00
		18/02/2013		83,838.10	83,838.10	61,207.00
		19/02/2013	71,051.30	83,838.10	83,838.10	61,207.00
		20/02/2013	75,963.20	83,838.10	83,838.10	61,207.00
		21/02/2013	88,751.90	83,838.10	83,838.10	61,207.00
		22/02/2013	30,299.00	83,838.10	83,838.10	61,207.00
		23/02/2013	56,822.25	83,838.10	83,838.10	61,207.00
		24/02/2013	38,469.35	83,838.10	83,838.10	61,207.00
		25/02/2013		83,838.10	83,838.10	61,207.00
		26/02/2013	41,569.15	83,838.10	83,838.10	61,207.00
		27/02/2013	85,584.45	83,838.10	83,838.10	61,207.00
		28/02/2013	38,656.80	83,838.10	83,838.10	61,207.00
		Total	**1,383,414.65**	**83,838.10**	**83,838.10**	
	March	1/03/2013	60,815.65	38,656.80	38,656.80	61,207.00
		2/03/2013	111,328.05	38,656.80	38,656.80	60,815.65

Figure 2.41 Opening Profit DateAdd Wrong shown alongside the correct Opening Profit measure

The reason why this is not correct is that the dates are shifted back first, then STARTOFMONTH returns the first date of the month of the shifted

date.

> ***MORE INFO* TIME INTELLIGENCE DAX FUNCTIONS**
>
> For more information on the functions discussed above, see "Time Intelligence Functions (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634763.aspx*.

If you need to calculate a value for the first or last date in the current filter context, you can use the FIRSTDATE or LASTDATE function, respectively. Both functions can be used as filters in CALCULATE. The functions do not override the currently selected dates. More specifically, the two functions can be rewritten using FILTER and VALUES as follows:

Click here to view code image

```
// Same as FIRSTDATE
FILTER (
    VALUES ( 'Date'[Date] ),
    'Date'[Date] = MIN ( 'Date'[Date] )
)

// Same as LASTDATE
FILTER (
    VALUES ( 'Date'[Date] ),
    'Date'[Date] = MAX ( 'Date'[Date] )
)
```

> ***MORE INFO* FIRSTDATE AND LASTDATE**
>
> For more information on FIRSTDATE and LASTDATE, see a blog article by Marco Russo, "Difference between LASTDATE and MAX for semi-additive measures in #DAX" at *http://sqlblog.com/blogs/marco_russo/archive/2013/10/22/difference-between-lastdate-and-max-for-semi-additive-measures-in-dax.aspx*.

If you need to filter by a custom date interval, you can use DATESBETWEEN. The function receives three arguments: the date column of a calendar table, a start date, and an end date. For example, calculate Total Profit between 10 June 2013 and 5 April 2014, inclusive, as follows:

Click here to view code image

```
Profit Custom Period =
CALCULATE (
    [Total Profit],
    DATESBETWEEN (
        'Date'[Date],
        DATE ( 2013, 10, 6 ),
        DATE ( 2014, 4, 5 )
    )
)
```

The measure shows the same value regardless of the selected date.

> ***MORE INFO* USING DATESBETWEEN**
>
> For more details on the DATESBETWEEN function, see "DATESBETWEEN Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634557.aspx*.

If your custom time interval is based on days, months, quarters, or years, you can use the DATESINPERIOD function. The function receives exactly three arguments: the date column from a calendar table, a start date, number of intervals, and the interval, which can be DAY, MONTH, QUARTER, or YEAR. For instance, you can calculate the rolling monthly profit with the following measure:

Click here to view code image

```
Rolling Monthly Profit =
CALCULATE (
    [Total Profit],
    DATESINPERIOD (
        'Date'[Date],
        MAX ( 'Date'[Date] ),
        -1,
        MONTH
    )
)
```

> ***MORE INFO* DATESINPERIOD**
>
> For more details and examples on the DATESINPERIOD function, see "DATESINPERIOD Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634539.aspx*.

***MORE INFO* TIME INTELLIGENCE IN DAX**

For an overview of Time Intelligence in DAX, see a blog article by Matt Allington: “DAX Time Intelligence Explained” at *https://exceleratorbi.com.au/dax-time-intelligence-beginners/*,

For a video overview, see a presentation Alberto Ferrari: “Time Intelligence in DAX” at *https://www.sqlbi.com/tv/time-intelligence-in-dax-sqlbits-x/*.

Using inactive relationships

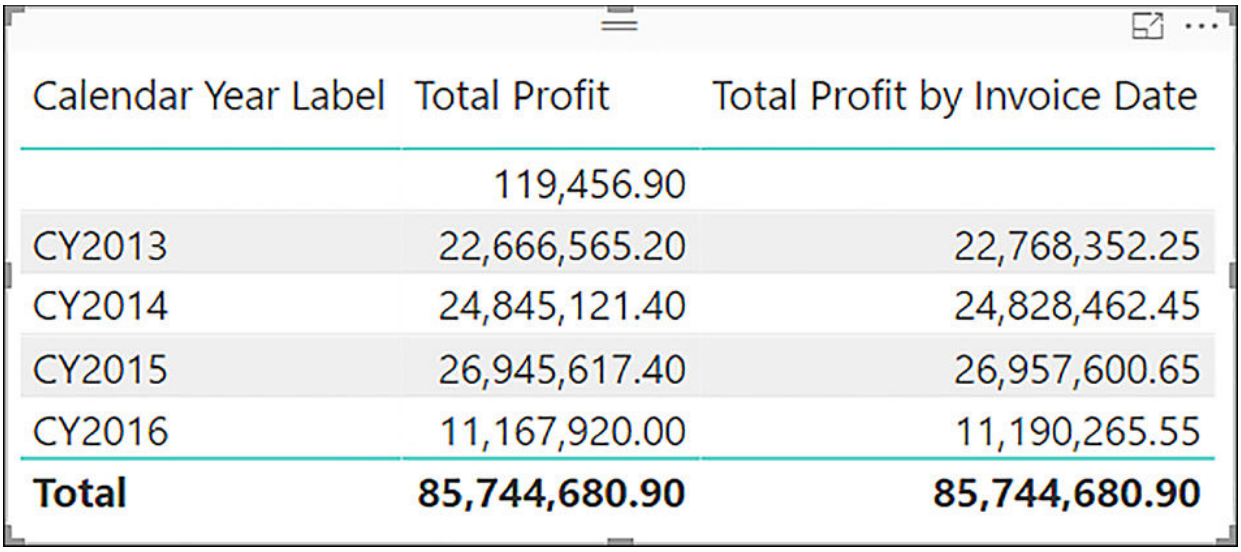
As reviewed earlier, there can be no more than one active physical relationship between two tables. Between the Sale and Date tables there are two active relationships:

- Active one from Sale (Delivery Date Key) to Date (Date)
- Inactive one from Sale (Invoice Date Key) to Date (Date)

By default, all values we aggregate in the Sale table are going to be filtered by Delivery Date Key. To use the inactive relationship, activate it programmatically with DAX using the USERELATIONSHIP function. This function receives two parameters, which are the columns used in a relationship. To calculate Total Profit by Invoice Date Key, you can write the following measure formula, as shown in Figure 2.42.

Click here to view code image

```
Total Profit by Invoice Date =
CALCULATE (
    [Total Profit],
    USERELATIONSHIP ( 'Date'[Date], Sale[Invoice Date Key] )
)
```



Calendar Year Label	Total Profit	Total Profit by Invoice Date
	119,456.90	
CY2013	22,666,565.20	22,768,352.25
CY2014	24,845,121.40	24,828,462.45
CY2015	26,945,617.40	26,957,600.65
CY2016	11,167,920.00	11,190,265.55
Total	**85,744,680.90**	**85,744,680.90**

FIGURE 2.42 Total Profit by Invoice Date used in a table visual

Note that the Total Profit by Invoice Date measure is blank when Calendar Year Label is blank. This is because no Invoice Date Key value is blank, which means there is no blank row automatically added to the Date table when filtering by Invoice Date Key instead of Delivery Date Key.

The order of the columns used as parameters does not matter, though there must exist a relationship between two columns. Otherwise, you will get the error seen in Figure 2.43.

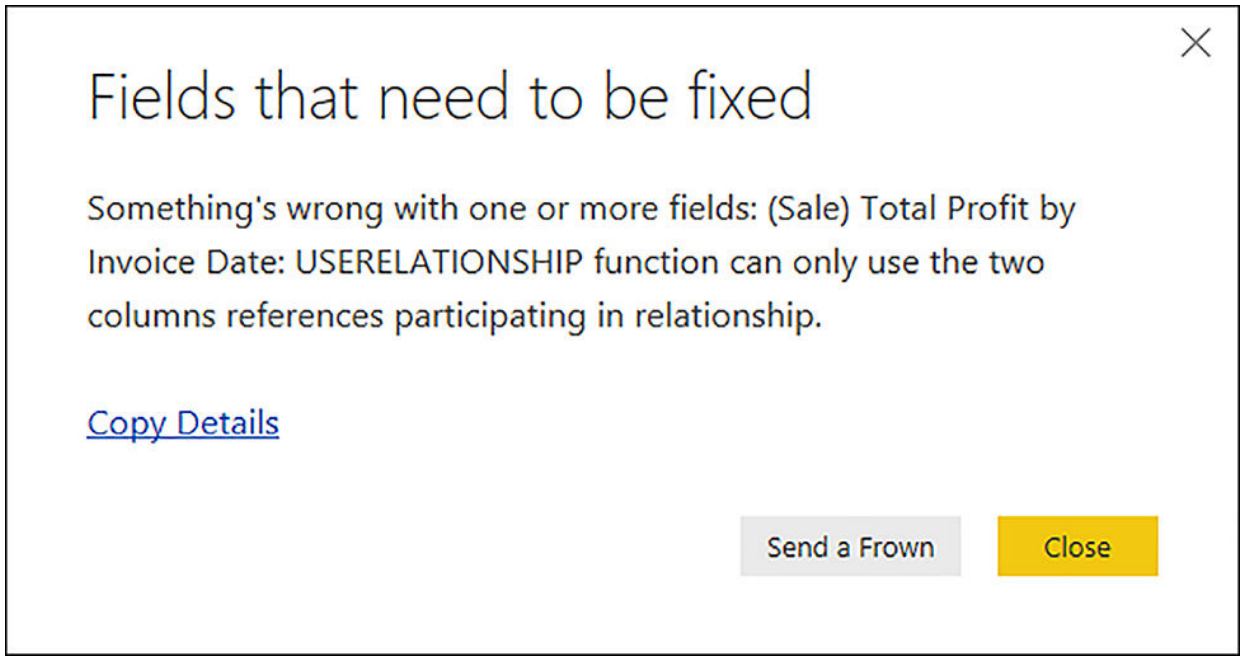


FIGURE 2.43 Error message when using USERELATIONSHIP without a relationship

> ***MORE INFO*** **USING USERELATIONSHIP**
>
> For more information on the USERELATIONSHIP function, see "USERELATIONSHIP Function (DAX)" at *https://msdn.microsoft.com/en-us/library/hh230952.aspx*.

Using SELECTEDVALUE

The SELECTEDVALUE function receives a column reference as the first parameter, which is required, and SELECTEDVALUE receives a default value as the optional second parameter. The function returns a column value if there is only one in the current filter context. Otherwise, it returns the default value. The function acts as a shortcut for the following syntax:

Click here to view code image

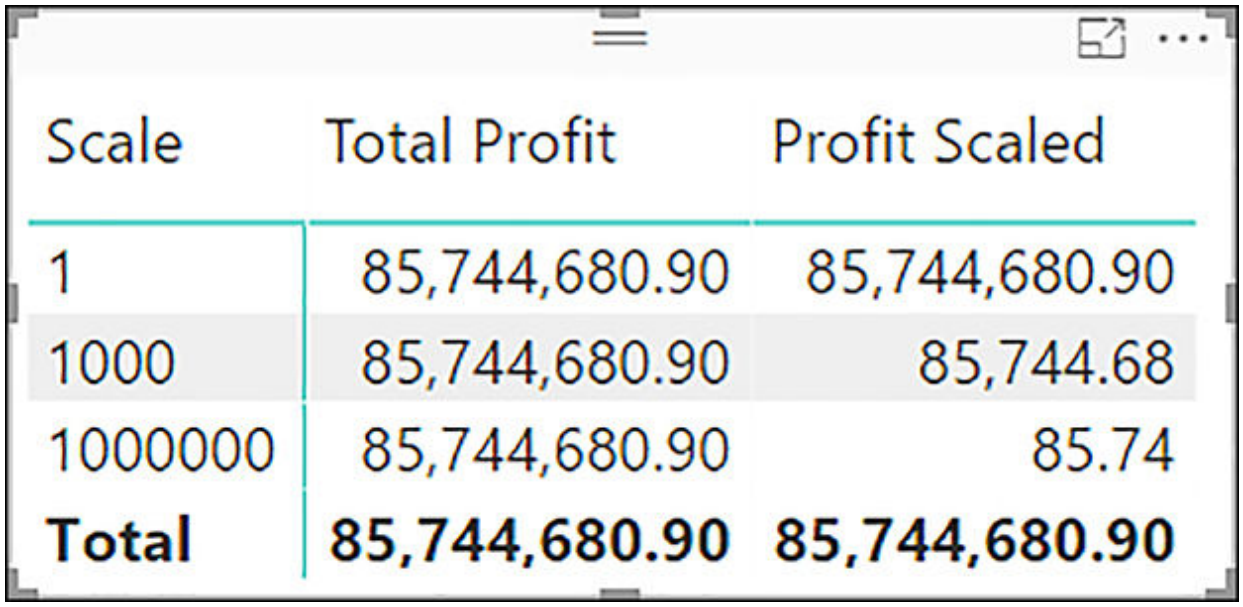
```
MyParameter Value =
IF (
    HASONEVALUE ( MyParameter[MyParameter] ),
    VALUES ( MyParameter[MyParameter] ),
    5
)
```

The HASONEVALUE function checks whether the input column has only one value in the current filter context. If it does, then the VALUES function, which returns a table, is converted into a scalar value because it has only one row and one column. If the input column has more than one value, the default value—in this case, 5—is returned. This function can be useful when you want to change a metric based on the selection. For instance, you can scale the Total Profit value based on the selected value in the Scale column of the Scale table with the following measure:

Click here to view code image

```
Profit Scaled = [Total Profit] / SELECTEDVALUE ( Scale[Scale], 1
)
```

If you put the Scale column values on rows in a matrix visual and Total Profit and Profit Scaled as values, you should see a visual similar to Figure 2.44.



Scale	Total Profit	Profit Scaled
1	85,744,680.90	85,744,680.90
1000	85,744,680.90	85,744.68
1000000	85,744,680.90	85.74
Total	**85,744,680.90**	**85,744,680.90**

Figure 2.44 Total Profit and Profit Scaled put against the Scale column

Note how in Figure 2.44, Total Profit shows the same value for each Scale value because there is no relationship between Scale and any other table. However, Profit Scaled shows different values because it reads the currently selected value of Scale from the current filter context by using SELECTEDVALUE. Also, note two things:

- First, Total Profit and Profit Scaled have the same value when the Scale is 1.
- Second, Profit Scaled at the total level shows the same value as when the Scale is 1 because at the total level, there is no filter on Scale, therefore, the default value of 1 is used.

> ***MORE INFO* SELECTEDVALUE**
>
> For more information on SELECTEDVALUE and more examples of how SELECTEDVALUE can be used, see an article by Marco Russo, "Using the SELECTEDVALUE function in DAX" at *https://www.sqlbi.com/articles/using-the-selectedvalue-function-in-dax/*.

FIRSTNONBLANK and LASTNONBLANK

The FIRSTNONBLANK and LASTNONBLANK functions work in the same way except the former returns the first non-blank value, while the latter returns the last non-blank value. For this reason, we are reviewing only FIRSTNONBLANK.

FIRSTNONBLANK returns a table with one row and one column, which can be used as a filter in CALCULATE. The function always takes two parameters: a column reference and an expression to check for blanks. You can also use single-column table expressions in FIRSTNONBLANK. To illustrate how the function works, calculate first non-blank Total Profit value in the Date column with the following measure:

Click here to view code image

```
First Profit =
CALCULATE (
    [Total Profit],
    FIRSTNONBLANK (
        'Date'[Date],
        [Total Profit]
    )
)
```

While the first date we have in the Date column is 1 January 2013, there is no Total Profit value for that date. The first date with Total Profit is 2 January 2013, which is what the measure will display.

FIRSTNOBLANK is an iterator, which means it evaluates its expression in row context. In the First Profit measure above, Total Profit was evaluated for each row of the Date column. It is important to understand that if you used `SUM (Sale[Profit])` instead of `[Total Profit]`, you would get incorrect results. Namely, because there would be no context transition triggered by implicit CALCULATE wrapped around Total Profit, every row would get the same value, resulting in a blank value—Total Profit for 1 January 2013.

> ***MORE INFO* FIRSTNONBLANK AND LASTNONBLANK**
>
> For more information on the two functions, see "FIRSTNONBLANK Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634210.aspx*, and "LASTNONBLANK Function (DAX)" at *https://msdn.microsoft.com/en-us/library/ee634247.aspx*.
>
> For a detailed discussion on how the functions work, see an article by Matt Allington, "LASTNONBLANK Explained" at *https://exceleratorbi.com.au/lastnonblank-explained/*.

Passing filters from disconnected tables

In some cases, you may need to pass filters from disconnected tables. For example, it might be too expensive to create a column with concatenated keys to create a physical relationship. In this case, it might make sense to use virtual relationships.

One way you can pass filters from one table to another is by using INTERSECT. As discussed previously, this table returns a table with rows that exist in both tables.

To see the effect of using a virtual relationship, follow the next steps:

1. Make the following two relationships inactive:
   - Between Date and Calendar Year tables
   - Between Customer and Bill To Customer tables
2. Create the following three measures in the Target table:

   Click here to view code image

```
Total Target Quantity = SUM ( Target[Target Quantity] )
Total Target Amount = SUM ( Target[Target Amount Excluding
Tax] )
Target Quantity Intersect =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( Target[Calendar Year] ),
        VALUES ( 'Date'[Calendar Year] )
    )
)
```

3. Hide the Target Quantity and Target Amount Excluding Tax columns.

At this stage, if you double-click on the arrow next to the Fields pane title (above the search bar), the Target table should acquire a new icon, which is similar to the measure icon. This means that all the visible fields in this table are measures.

If you create a matrix visual with Calendar Year Label on rows and Total Target Quantity and Target Quantity Intersect as values, you will see a visual similar to Figure 2.45.

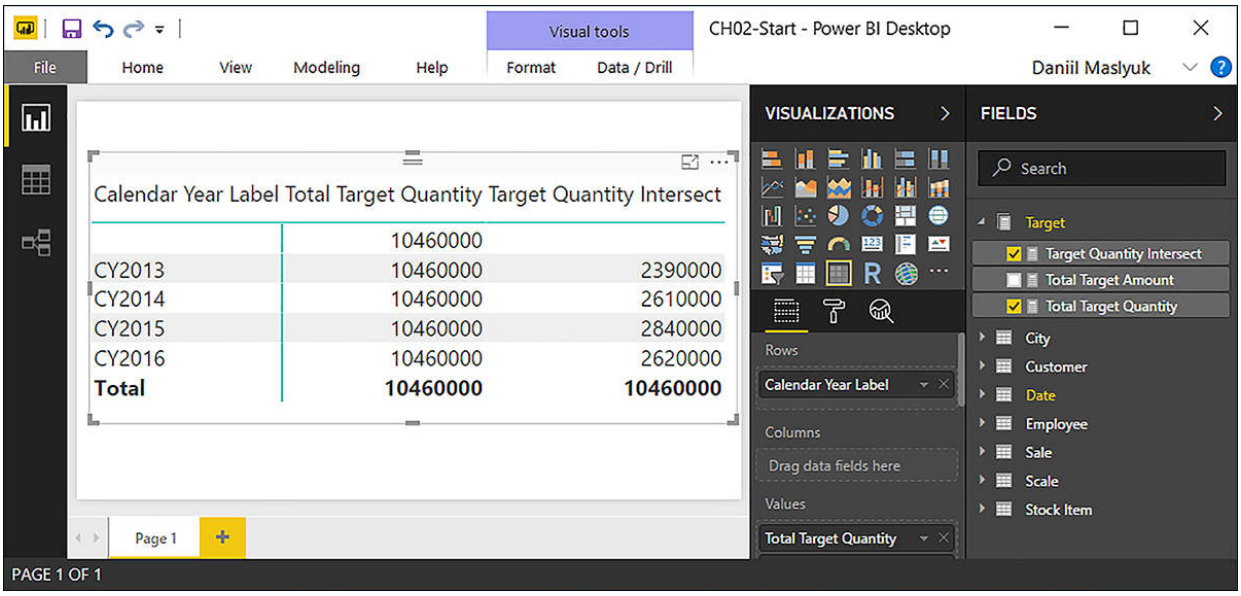


FIGURE 2.45 Calendar Year Label, Total Target Quantity, and Target Quantity Intersect in matrix visual

Note how Total Target Quantity has the same value regardless of the Calendar Year Label value. This is because we deactivated the relationship

between the Date and Calendar Year table, meaning that filters from the Date table do not reach the Target table. However, the Target Quantity Intersect measure shows different values for each Calendar Year Label because it is being filtered by a virtual relationship created with INTERSECT.

It is important to note that the order in which you pass the parameters to INTERSECT matters. The following measure returns the same values as Total Target Quantity, which is the same value for each year:

Click here to view code image

```
Target Quantity Intersect Wrong =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( 'Date'[Calendar Year] ),
        VALUES ( Target[Calendar Year] )
    )
)
```

When you create virtual relationships with INTERSECT, the table you want to pass filters from should come first, while the table you want to filter should come second. In our example so far, we have passed filters from the Date table only, but not from the Customer. While we could do this by adding another INTERSECT filter to CALCULATE, another way is to use ALL and SUMMARIZE. The following measures return the same results:

Click here to view code image

```
Target Quantity Intersect Values =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( Target[Calendar Year] ),
        VALUES ( 'Date'[Calendar Year] )
    ),
    INTERSECT (
        VALUES ( Target[Bill To Customer] ),
        VALUES ( Customer[Bill To Customer] )
    )
)

Target Quantity Intersect Summarize =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        ALL ( Target[Calendar Year], Target[Bill To Customer] ),
```

```
        SUMMARIZE ( Sale, 'Date'[Calendar Year], Customer[Bill
To Customer] )
    )
)
```

It's also possible to create virtual relationships using the TREATAS function. The function receives at least two arguments: a table to pass filters from, and one or more columns to pass filters to. For instance, rewrite the preceding measures in the following way:

Click here to view code image

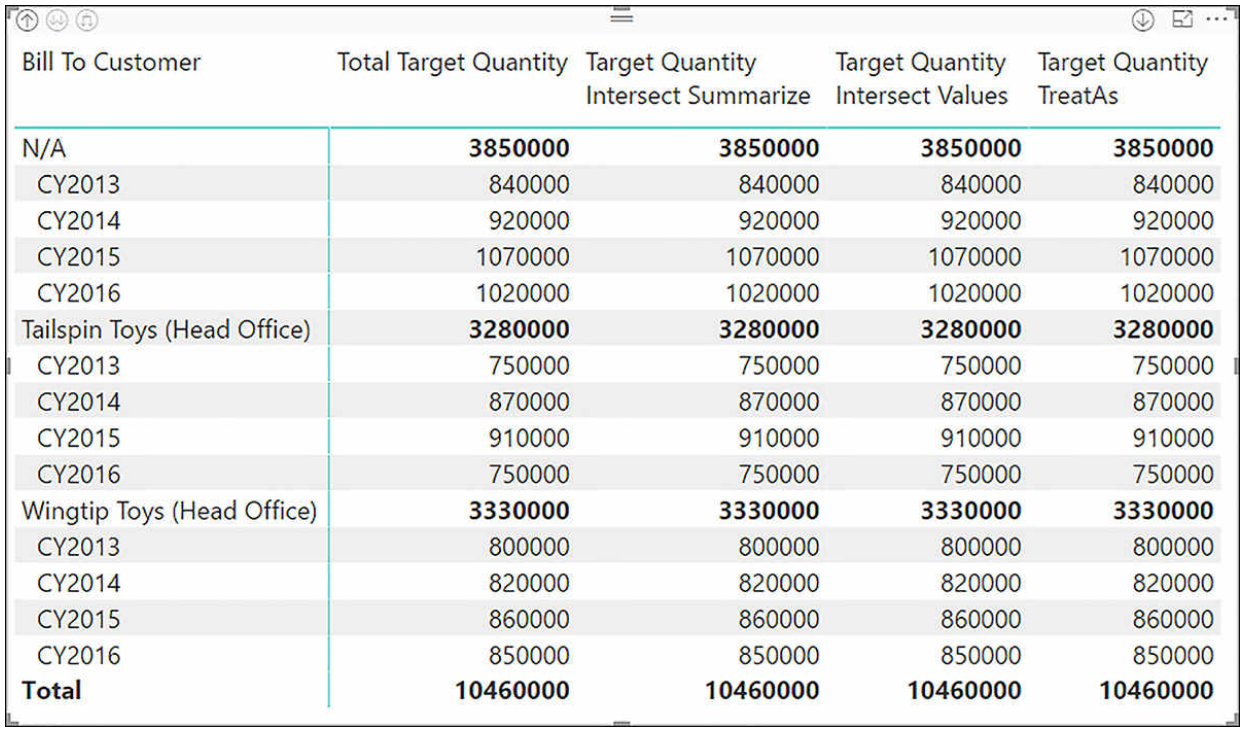
```
Target Quantity TreatAs =
CALCULATE (
    [Total Target Quantity],
    TREATAS (
        SUMMARIZE ( Sale, 'Date'[Calendar Year], Customer[Bill
To Customer] ),
        Target[Calendar Year],
        Target[Bill To Customer]
    )
)
```

If you now add Bill To Customer from the Customer table to the matrix and the three measures created above, you should see a matrix visual similar to Figure 2.46.

Bill To Customer	Total Target Quantity	Target Quantity Intersect Summarize	Target Quantity Intersect Values	Target Quantity TreatAs
N/A	**10460000**	**3850000**	**3850000**	**3850000**
	10460000			
CY2013	10460000	840000	840000	840000
CY2014	10460000	920000	920000	920000
CY2015	10460000	1070000	1070000	1070000
CY2016	10460000	1020000	1020000	1020000
Tailspin Toys (Head Office)	**10460000**	**3280000**	**3280000**	**3280000**
	10460000			
CY2013	10460000	750000	750000	750000
CY2014	10460000	870000	870000	870000
CY2015	10460000	910000	910000	910000
CY2016	10460000	750000	750000	750000
Wingtip Toys (Head Office)	**10460000**	**3330000**	**3330000**	**3330000**
	10460000			
CY2013	10460000	800000	800000	800000
CY2014	10460000	820000	820000	820000
CY2015	10460000	860000	860000	860000
CY2016	10460000	850000	850000	850000
Total	**10460000**	**10460000**	**10460000**	**10460000**

Figure 2.46 Measures with virtual relationships

Note how Total Target Quantity still displays the same value for each row. At the beginning of this exercise, we purposefully deactivated relationships between Date and Calendar Year and between Customer and Bill To Customer. These relationships were deactivated to highlight the effect that virtual relationships can have. While virtual relationships can be very powerful, if you have an option to create physical relationships, as in our case, it is best to pass filters using physical relationships instead of virtual ones. Your code will be much shorter, and it will also perform much better. If at this point, you activate the relationships again, the same matrix visual will looks like the one shown in Figure 2.47.



Bill To Customer	Total Target Quantity	Target Quantity Intersect Summarize	Target Quantity Intersect Values	Target Quantity TreatAs
N/A	**3850000**	**3850000**	**3850000**	**3850000**
CY2013	840000	840000	840000	840000
CY2014	920000	920000	920000	920000
CY2015	1070000	1070000	1070000	1070000
CY2016	1020000	1020000	1020000	1020000
Tailspin Toys (Head Office)	**3280000**	**3280000**	**3280000**	**3280000**
CY2013	750000	750000	750000	750000
CY2014	870000	870000	870000	870000
CY2015	910000	910000	910000	910000
CY2016	750000	750000	750000	750000
Wingtip Toys (Head Office)	**3330000**	**3330000**	**3330000**	**3330000**
CY2013	800000	800000	800000	800000
CY2014	820000	820000	820000	820000
CY2015	860000	860000	860000	860000
CY2016	850000	850000	850000	850000
Total	**10460000**	**10460000**	**10460000**	**10460000**

FIGURE 2.47 The same matrix visual after activating relationships

When using activated relationships, the Total Target Quantity measure displays the same values as the other three measures because the filters from the Date and Customer table now reach the Target table.

> ***MORE INFO* USING TREATAS AND ITS ALTERNATIVES**
>
> For more information on the TREATAS function, see "TREATAS Function" at *https://msdn.microsoft.com/en-us/library/mt825214.aspx*.
>
> For a detailed discussion of using INTERSECT and TREATAS to propagate filters in DAX, including performance considerations, see an article by Marco Russo, "Propagating filters using TREATAS in DAX" at *https://www.sqlbi.com/articles/propagate-filters-using-treatas-in-dax/*.

Quick measures

Quick measures is a Power BI Desktop feature that lets you create explicit measures without writing DAX. To create a Quick Measure, you can either

select **Home** > **Calculations** > **New Quick Measure**, or right-click on a table where you want to create the measure and select **New quick measure**. The window that opens with the Calculation drop-down list expanded is shown in in Figure 2.48.

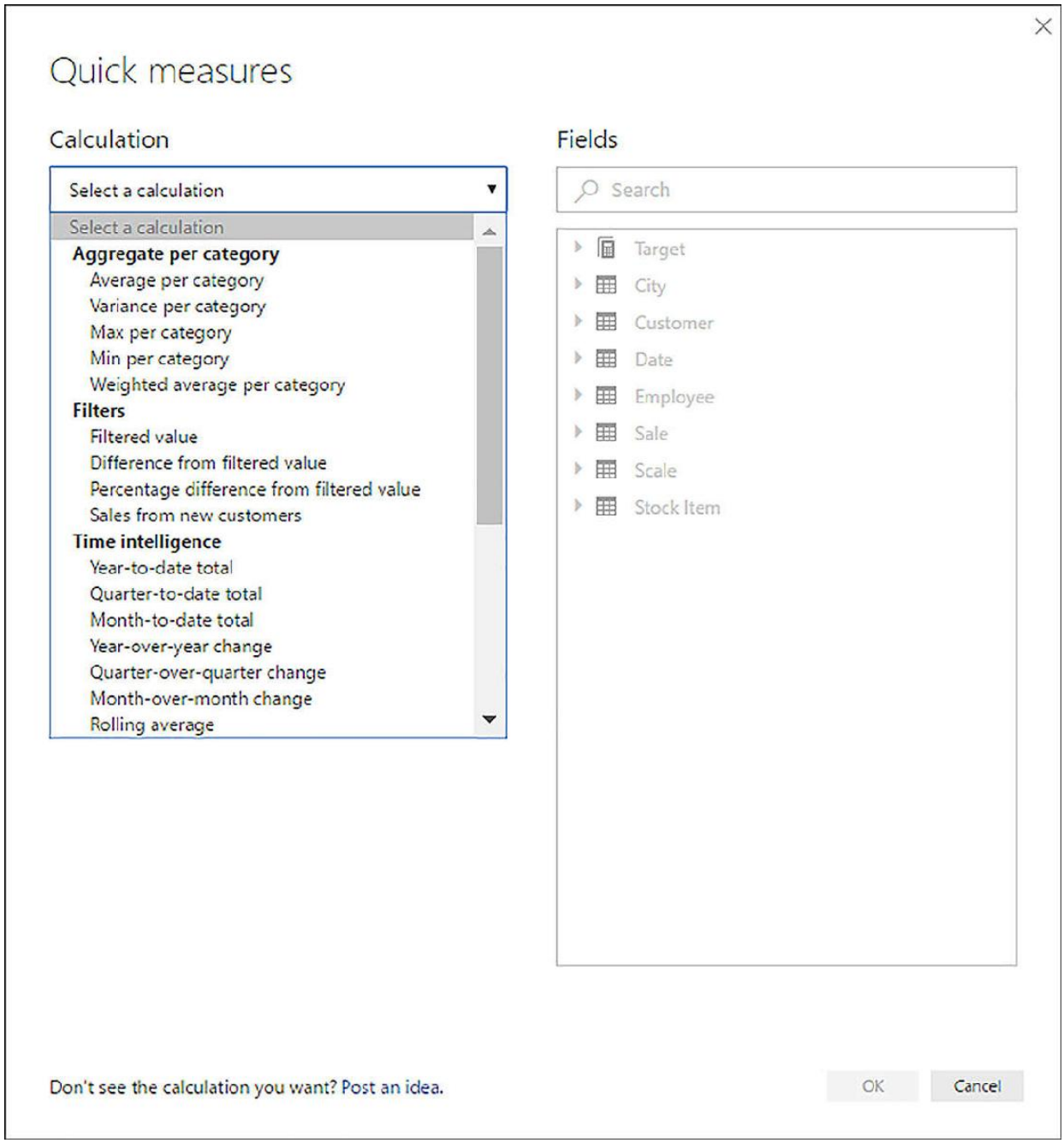


FIGURE 2.48 Quick Measures window

Currently, there are 27 Quick Measures grouped into six categories.

- Aggregate per category
  - Average per category
  - Variance per category
  - Max per category
  - Min per category
  - Weighted average per category
- Filters
  - Filtered value
  - Difference from filtered value
  - Percentage difference from filtered value
  - Sales from new customers
- Time intelligence
  - Year-to-date total
  - Quarter-to-date total
  - Month-to-date total
  - Year-over-year change
  - Quarter-over-quarter change
  - Month-over-month change
  - Rolling average
- Totals
  - Running total
  - Total for category (filters applied)
  - Total for category (filters not applied)
- Mathematical operations
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Percentage difference

- Correlation coefficient
- Text
  - Start rating
  - Concatenated list of values

After you select a calculation, you will need to drag fields in the required field wells. If you opened the Quick Measures window by right-clicking on a measure and selecting New Quick Measure, the measure you right-clicked on would be automatically used in one of the field wells. For example, if you right-click on the Total Profit measure and select New Quick Measure, then select Year-over-year change in the Calculation drop-down list, you can see the Quick Measures window transform as shown in Figure 2.49.

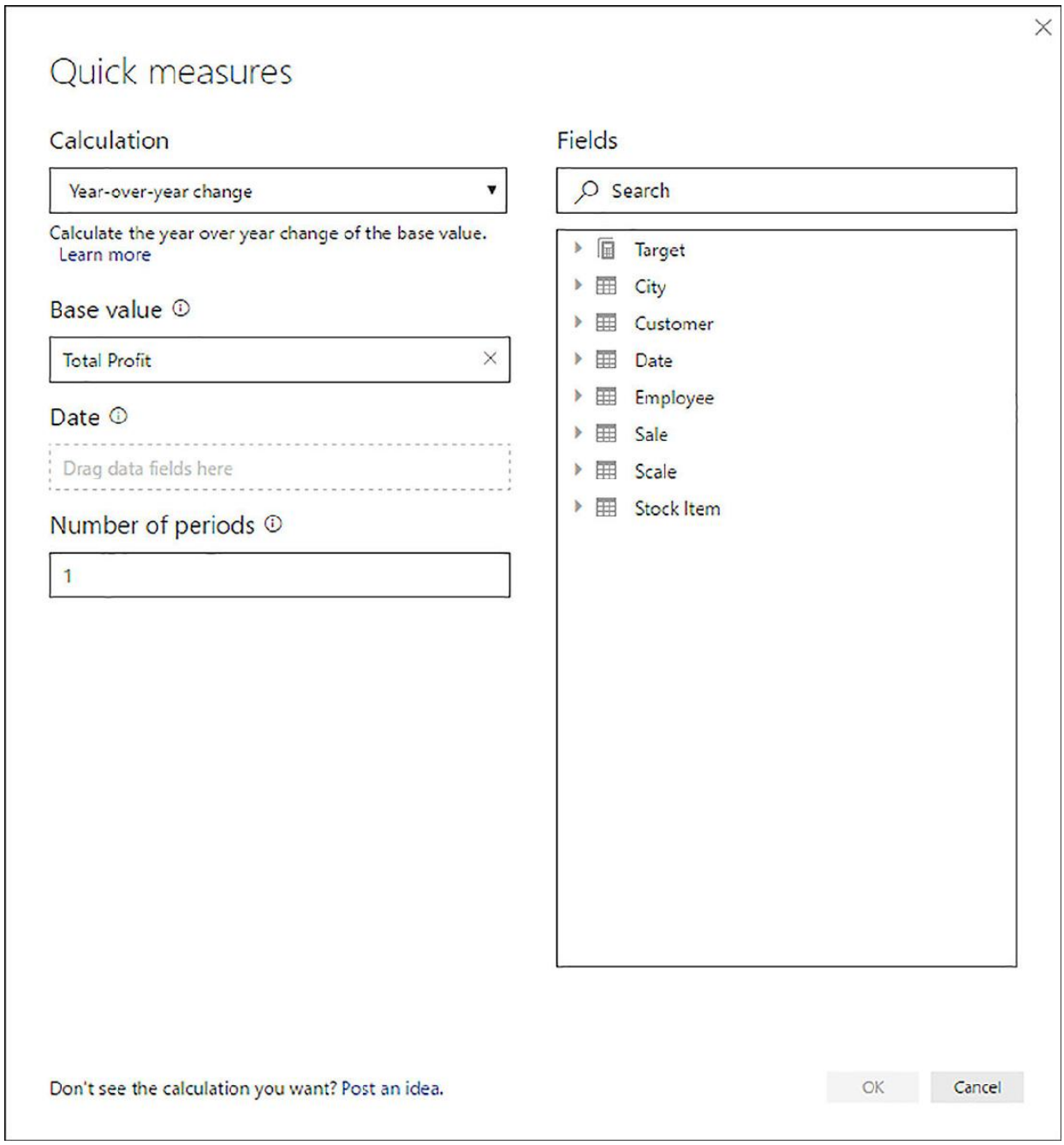


FIGURE 2.49 Quick Measures window with Year-over-Year Change selected

While the Number of periods field displays 1 by default, it means 1 period back, which is different from the way DATEADD works. To finalize this quick measure, drag the Date column from the Date table to the Date field well, then click OK. After doing so, a new measure, Total Profit YoY%, is created If you select it in the Fields pane, you will see its DAX code:

Click here to view code image

```
Total Profit YoY% =
IF(
        ISFILTERED('Date'[Date]),
        ERROR("Time intelligence quick measures can only be
grouped or filtered by the Power BI-provided date hierarchy or
primary date column."),
        VAR __PREV_YEAR = CALCULATE([Total Profit],
DATEADD('Date'[Date].[Date], -1, YEAR))
        RETURN
                DIVIDE([Total Profit] - __PREV_YEAR,
__PREV_YEAR)
)
```

The current limitation of time-related Quick Measures is that they rely on the Power BI-provided date hierarchies. As you can see, inside DATEADD there is a reference to the 'Date'[Date].[Date] column. This column is part of a hidden automatic date hierarchy created by Power BI. These hierarchies are covered in "Skill 2.4: Create hierarchies."

Another noteworthy aspect of the formula is the use of the ERROR function. This function can purposefully return an error with user-defined text. For example, if you filter by the Date column instead of using the automatic date hierarchy, the visual will display the "Can't display the visual" message along with the "See details" hyperlink. Clicking on the hyperlink will open the "Couldn't load the data for this visual" window shown in Figure 2.50.

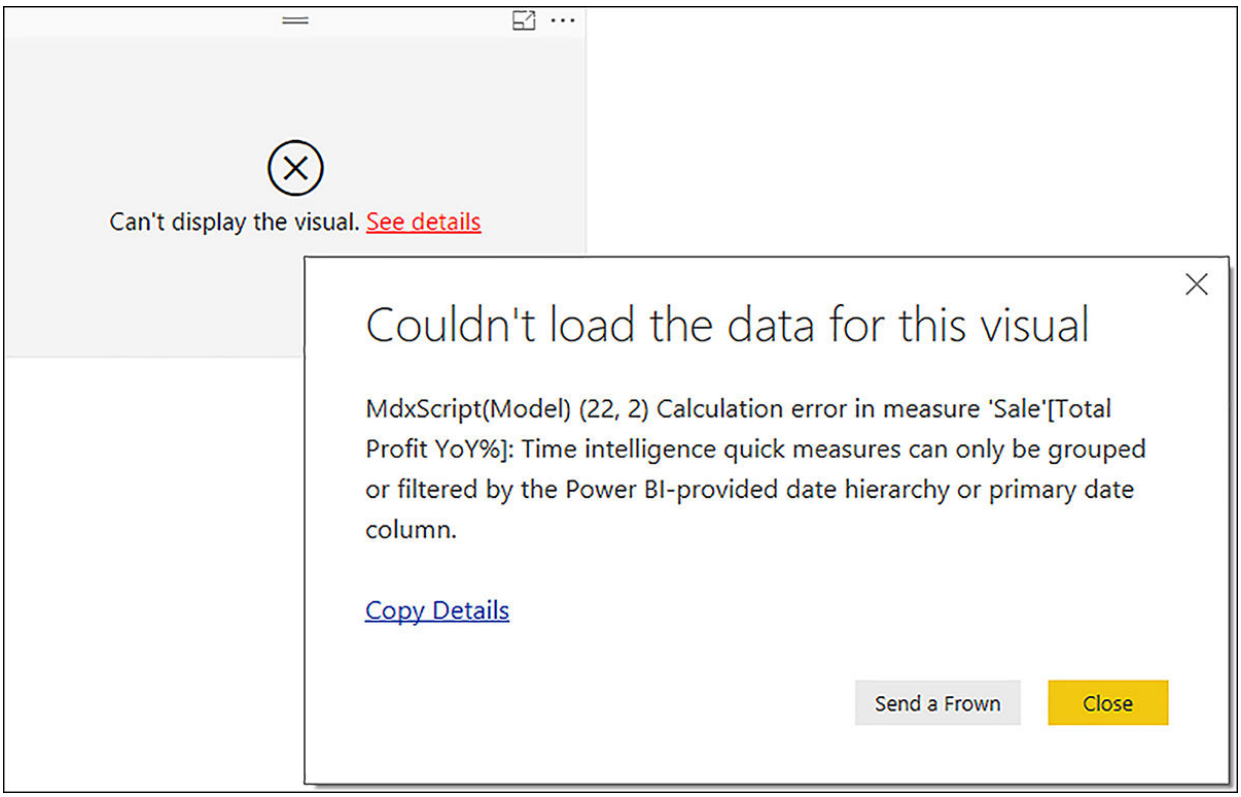


FIGURE 2.50 A window with user-defined error message

Quick measures are available where data modeling is available. Also, quick measures may be available with some SQL Server Analysis Services (SSAS) live connections, as well as Power BI Service live connection. With SSAS, the supported versions of the engine are Tabular and Azure Analysis Services. Quick measures are not available in SSAS Multidimensional mode. Furthermore, not all quick measures are supported by each version of the Tabular engine; this is because some quick measures use functions that are not available in the early SSAS versions.

Because Quick Measures is a feature that is in constant development, your experience might not exactly match the above figures, but the principles with which you create a Quick Measure remain the same.

The reader is encouraged to try other quick measures to get familiarized with how each of them works.

MORE INFO QUICK MEASURES

For more details on the Quick Measures feature of Power BI, including limitations, considerations, and other examples, see “Use

Quick measures to easily perform common and powerful calculations” at *https://docs.microsoft.com/en-us/power-bi/desktop-quick-measures*.

Use What-if parameters

If you want to create a parameter that you can change with a slicer and see how changing it affects some of your calculations, you can use the What-if feature of Power BI Desktop. This feature allows you to create a calculated table with a pre-defined range, a slicer to select a value from the table, and a measure that captures the selected parameter value.

The parameter created using the What-if feature is a DAX parameter, and it is different from the parameters created in Power Query Editor. Editing the latter prompts data refresh while selecting a DAX parameter value in a slicer allows you to alter its selected value without refreshing your dataset.

To create a parameter using this feature, select **Modeling** > **What-If** > **New Parameter**. This feature can only be accessed in the Report view. The What-if parameter dialog appears, as shown in Figure 2.51.

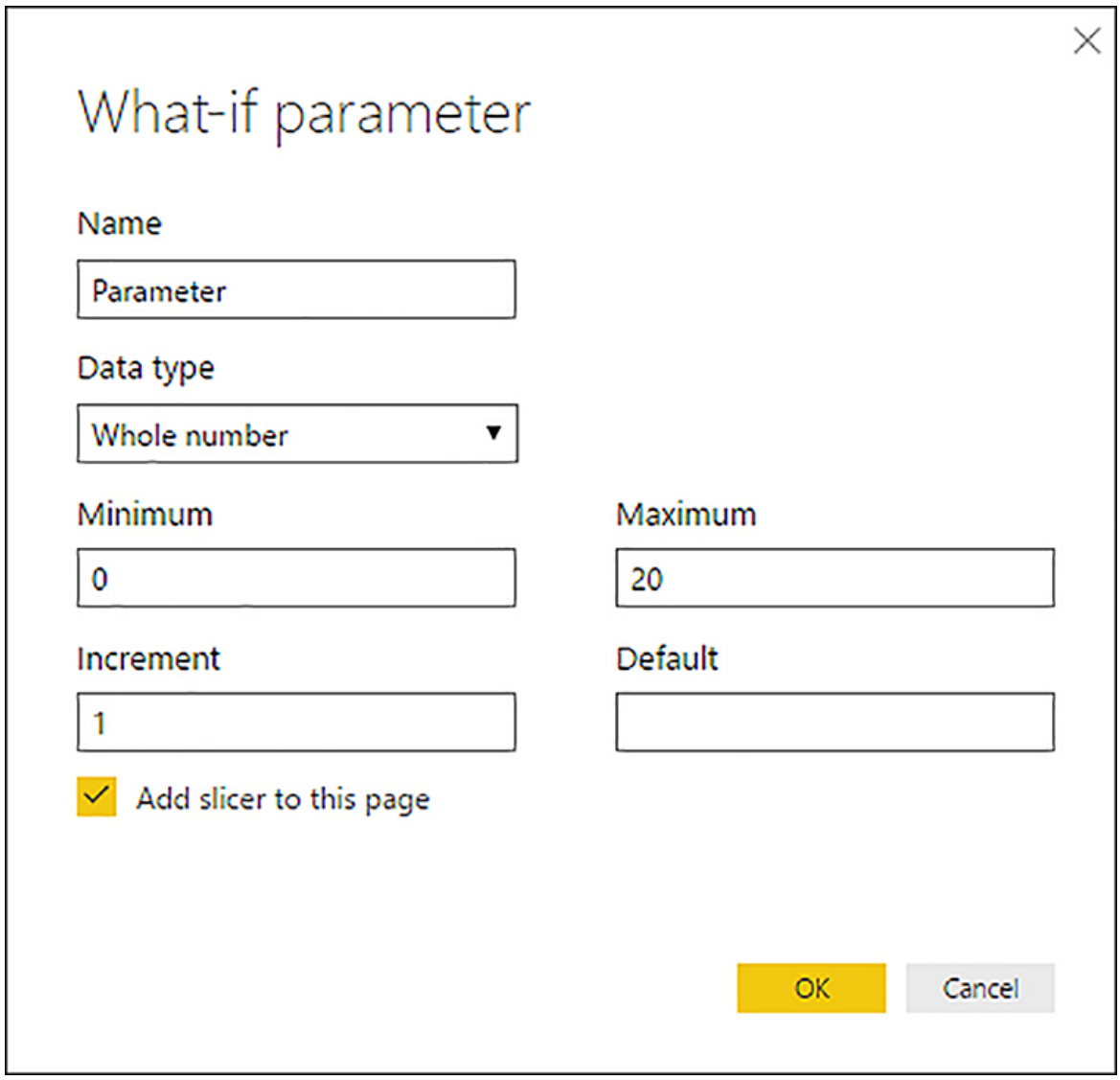


FIGURE 2.51 What-if parameter window

The parameter name will set the name of both the parameter table and its only column. The data types available in the drop-down list are Whole number, Decimal number, and Fixed decimal number. The Minimum, Maximum, and Increment values will be used in the GENERATESERIES function, which creates the parameter table, as the first, second, and third arguments, respectively. The Default value will be used in the selected value measure, which uses the SELECTEDVALUES function and is covered next. By default, a slicer with the parameter will be added to the active page.

If you call your parameter MyParameter and use a default value of 5, the selected value measure formula will be as follows:

Click here to view code image

```
MyParameter Value = SELECTEDVALUE(MyParameter[MyParameter], 5)
```

When you put the measure into a card visual alongside the slicer created by the What-if feature, your results should be similar to Figure 2.52.

FIGURE 2.52 MyParameter used in a slicer and MyParameter Value used in a card visual

If you then make a selection in the slicer, the card visual will display the selected value. Furthermore, you can use the MyParameter Value measure in another measure and immediately see the effect of your changing the parameter value with the slicer. For example, you can create the following measure:

Click here to view code image

```
Discounted Profit = [Total Profit] * ( 1 - [MyParameter Value] / 100
)
```

As you move the parameter slider, you will see the Discounted Profit value change.

> ***MORE INFO* WHAT-IF PARAMETERS**
>
> For more details on the What-if parameters feature of Power BI, see "Create and use a What if parameter to visualize variables in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-what-if*.

Skill 2.3: Measure performance by using KPIs, gauges, and cards

Organizations usually have targets towards which they work. These targets, when compared to the actual figures, are often called Key Performance Indicators, or KPIs. KPIs help organizations with tracking how close they are to achieving their targets, as well as which areas fare better compared to others. In Power BI, there are several ways to visualize targets and actual figures. For example, you can

display a value by itself in a Card visual, or you can use the KPI or Gauge visuals to compare actual and target figures.

If you want to follow the examples in this section, you can open the CH02-2.3Start.pbix file from the companion files folder.

This section covers how to:

- Calculate the actual
- Calculate the target
- Calculate actual to target
- Configure values for gauges
- Use the format settings to manually set values

Calculate the actual

In general, every KPI has at least two parts: the base, or actual figure, and the target figure. In the data model that we have created so far, we already have the necessary data to calculate the actual. In the Sale table, there is the Total Excluding Tax column, from which you can create an explicit measure:

```
Total Actual Amount = SUM ( Sale[Total Excluding Tax] )
```

You can now use this measure in a Card visual. To do this, click on the Card visual icon in the Visualizations pane and then click on the check box next to the Total Actual Amount measure. Alternatively, drag the measure into the Fields field well below the visualizations icons. Another way to create a visual is to drag a measure onto the canvas or click the check box next to it, which will create a clustered column chart. You can then convert this visual into a Card visual by clicking on its icon in the Visualizations pane. Whatever method you choose, your visual should look as shown in Figure 2.53.

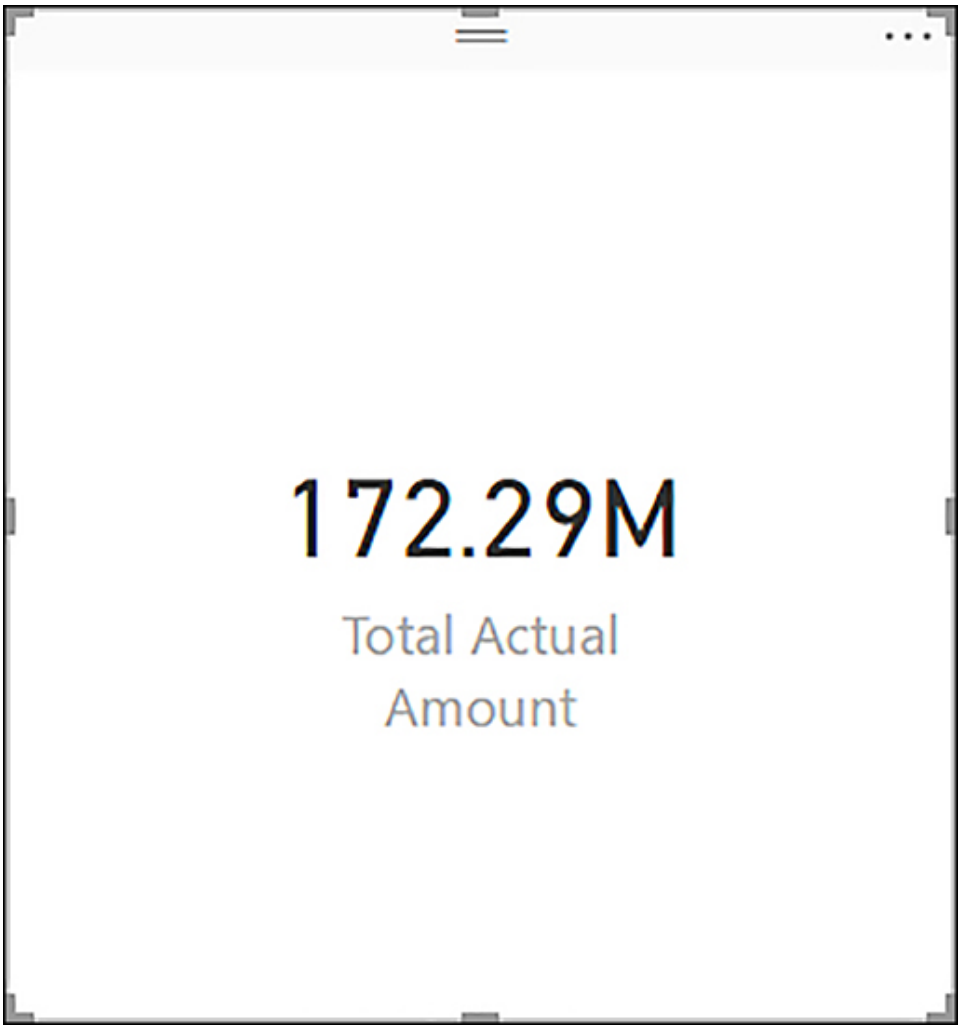


FIGURE 2.53 Card visual

With the card visual, you can use only one field. If you want to display several actuals in one visual, one way to do so would be a Multi-row Card. For instance, if you create a Multi-row card with Total Actual Amount, Total Profit, Total Target Amount, and Total Target Quantity as fields, your visual will look like in Figure 2.54.

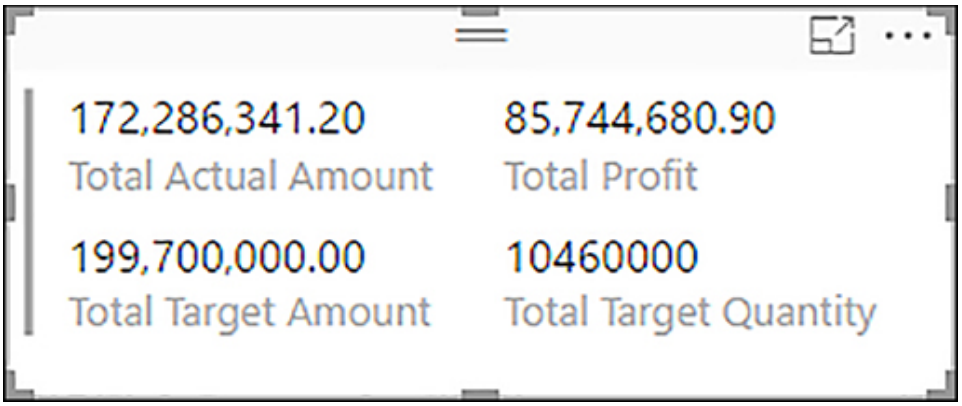
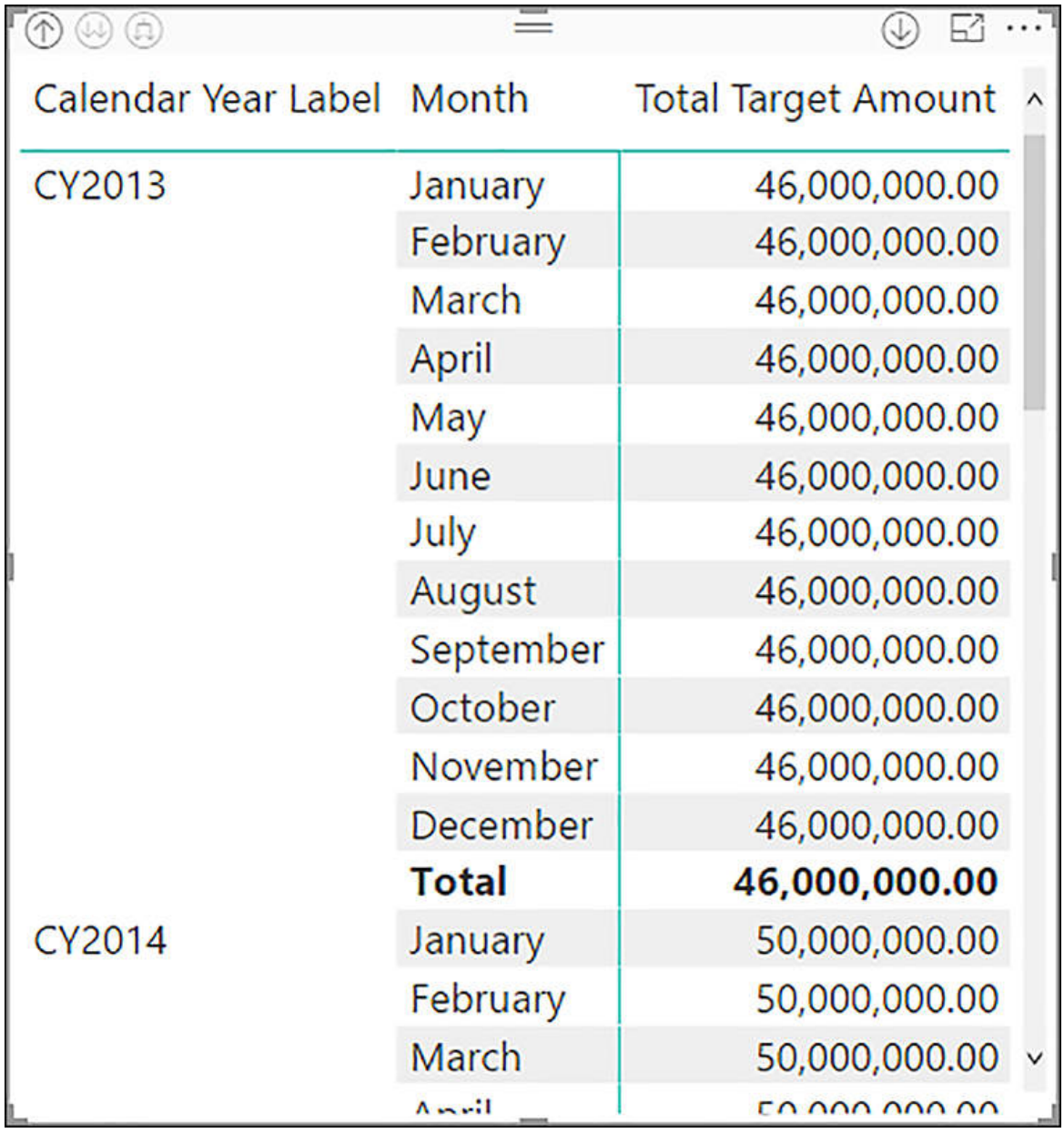


Figure 2.54 Multi-row card

If instead, you want to use the KPI or Gauge visual, you need to calculate the target amount.

Calculate the target

Target figures are often provided in separate tables from actual figures, and they are usually at different granularity level. For example, businesses rarely set targets at day level; instead, they might set targets at monthly or yearly level. In our data model, we have target figures set at Calendar Year and Bill To Customer level. The problem with the measures we have now is that they will still show values when analyzed at other levels of granularity. For example, if we display the Total Target Amount measure by Calendar Year Label and Month like in Figure 2.55, we will see the same values for each month within a given Calendar Year Label.



Calendar Year Label	Month	Total Target Amount
CY2013	January	46,000,000.00
	February	46,000,000.00
	March	46,000,000.00
	April	46,000,000.00
	May	46,000,000.00
	June	46,000,000.00
	July	46,000,000.00
	August	46,000,000.00
	September	46,000,000.00
	October	46,000,000.00
	November	46,000,000.00
	December	46,000,000.00
	Total	**46,000,000.00**
CY2014	January	50,000,000.00
	February	50,000,000.00
	March	50,000,000.00

FIGURE 2.55 Total Target Amount by Calendar Year Label and Month

These figures, while technically correct, are misleading to anyone looking at the report. This situation can be addressed in a few ways. First, the monthly figures can be divided by 12 so that they make sense at month level. However, if you browse the Date table at the day level, for instance, then you will again have to deal with this issue.

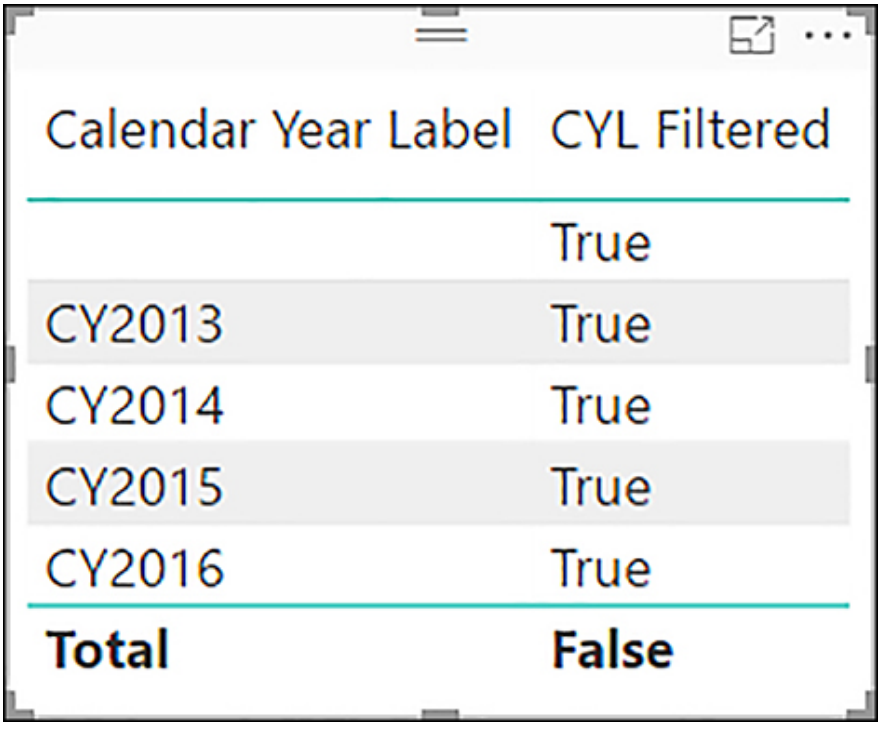
MORE INFO **ALLOCATING TARGETS AT DIFFERENT GRANULARITIES**

If you want to allocate your target or budget figures at higher granularity levels—for example, at month or day level when you only have yearly figures—you can use the budget pattern developed by Alberto Ferrari and Marco Russo. For more details and examples, see "Budget Patterns" at *http://www.daxpatterns.com/budget-patterns/*.

Second, these values can be hidden using DAX so that you only see the figures at Calendar Year level. You also need to take into account the columns from the Customer table that are at higher level of granularity than the Target figures, as well as all other tables.

By using the ISFILTERED function, you can check whether a column has been filtered explicitly. To see the effect of the function, create the following measure and slice it by Calendar Year Label. The resulting table are shown in Figure 2.56.

```
CYL Filtered = ISFILTERED ( 'Date'[Calendar Year Label] )
```



Calendar Year Label	CYL Filtered
	True
CY2013	True
CY2014	True
CY2015	True
CY2016	True
Total	**False**

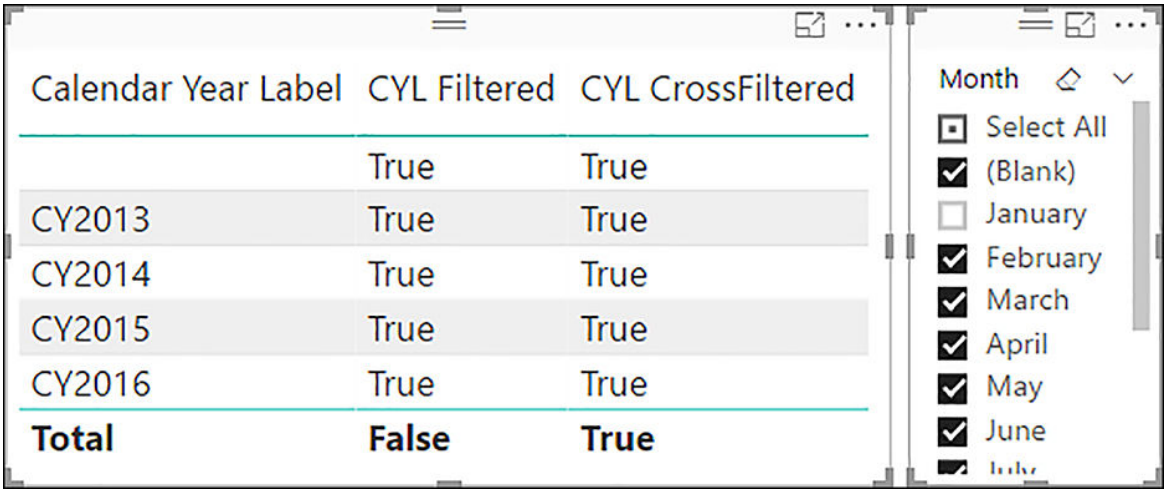
FIGURE 2.56 The CYL Filtered measure sliced by Calendar Year Label

Note how the measure shows False at the total level and True everywhere else. This is because at the total level, Calendar Year Label is not filtered. If you have a slicer that selects a few Calendar Year Label values, the total level will also display True. However, if filtered by Month, for instance, the total would still show False. In this case, you can use the ISCROSSFILTERED function that

checks whether the table in which the column exists has any other filters applied. You can now create the following measure, add it to the table from Figure 2.56 and slice it by Month.

```
CYL CrossFiltered = ISCROSSFILTERED ( 'Date'[Calendar Year Label] )
```

The results are shown in Figure 2.57.



Calendar Year Label	CYL Filtered	CYL CrossFiltered
	True	True
CY2013	True	True
CY2014	True	True
CY2015	True	True
CY2016	True	True
Total	**False**	**True**

FIGURE 2.57 CYL CrossFiltered in a matrix visual

Note how when we slice by Month, the CYL Filtered measure still shows False at the total level, yet the CYL CrossFiltered measure shows True. This is because there is a filter on the Date table which cross-filters Calendar Year Label, even if we do not see a difference in this case.

> ***MORE INFO* ISFILTERED AND ISCROSSFILTERED**
>
> For more information on these two functions, refer to "ISFILTERED Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492163.aspx*, and "ISCROSSFILTERED Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492197.aspx*.

You can now create a measure that checks if the currently browsed granularity is supported by the Target table:

Click here to view code image

```
Target Is Valid =
NOT (
    -- Check the Date table granularity
    ISFILTERED ( 'Date'[Calendar Month Label] )
        || ISFILTERED ( 'Date'[Date] )
```

```
        || ISFILTERED ( 'Date'[Day] )
        || ISFILTERED ( 'Date'[Fiscal Month Label] )
        || ISFILTERED ( 'Date'[Fiscal Year Label] )
        || ISFILTERED ( 'Date'[Month] )
        || ISFILTERED ( 'Date'[Short Month] )
    -- Check the Customer table granularity
        || ISFILTERED ( Customer[Customer] )
        || ISFILTERED ( Customer[Postal Code] )
        || ISFILTERED ( Customer[Primary Contact] )
    -- Check other tables
        || ISCROSSFILTERED ( City )
        || ISCROSSFILTERED ( Employee )
        || ISCROSSFILTERED ( 'Stock Item' )
)
```

> ***NOTE* COMMENTS IN DAX**
>
> You can write both single- and multi-line comments in DAX. For single-line comments, you need to prefix your code with a double forward slash (//) or a double dash (--), like in the Target table formula above. For multi-line comments, you need to write them between /* and */.

Note how ISCROSSFILTERED can also accept a table as a parameter, simplifying checking whether a filter has been applied on any of its columns. At this stage, create the following measure and put it into the table from Figure 2.55.

Click here to view code image

```
Target Amount = IF ( [Target Is Valid], [Total Target Amount] )
```

A sample of results can be seen in Figure 2.58.

Calendar Year Label	Month	Total Target Amount	Target Amount
CY2016	October	49,800,000.00	
	November	49,800,000.00	
	December	49,800,000.00	
	Total	**49,800,000.00**	**49,800,000.00**
Total		**199,700,000.00**	**199,700,000.00**

Figure 2.58 The Target Amount measure used in a table

At this stage, to compare the actual and target figures, use the KPI visual, which has three field wells:

- Indicator (actual or base value)
- Trend axis
- Target goals (one or two)

Only the first two fields are required. If you use Total Actual Amount as the indicator and Calendar Year Label as Trend axis, you will see a visual like the one shown in Figure 2.59.

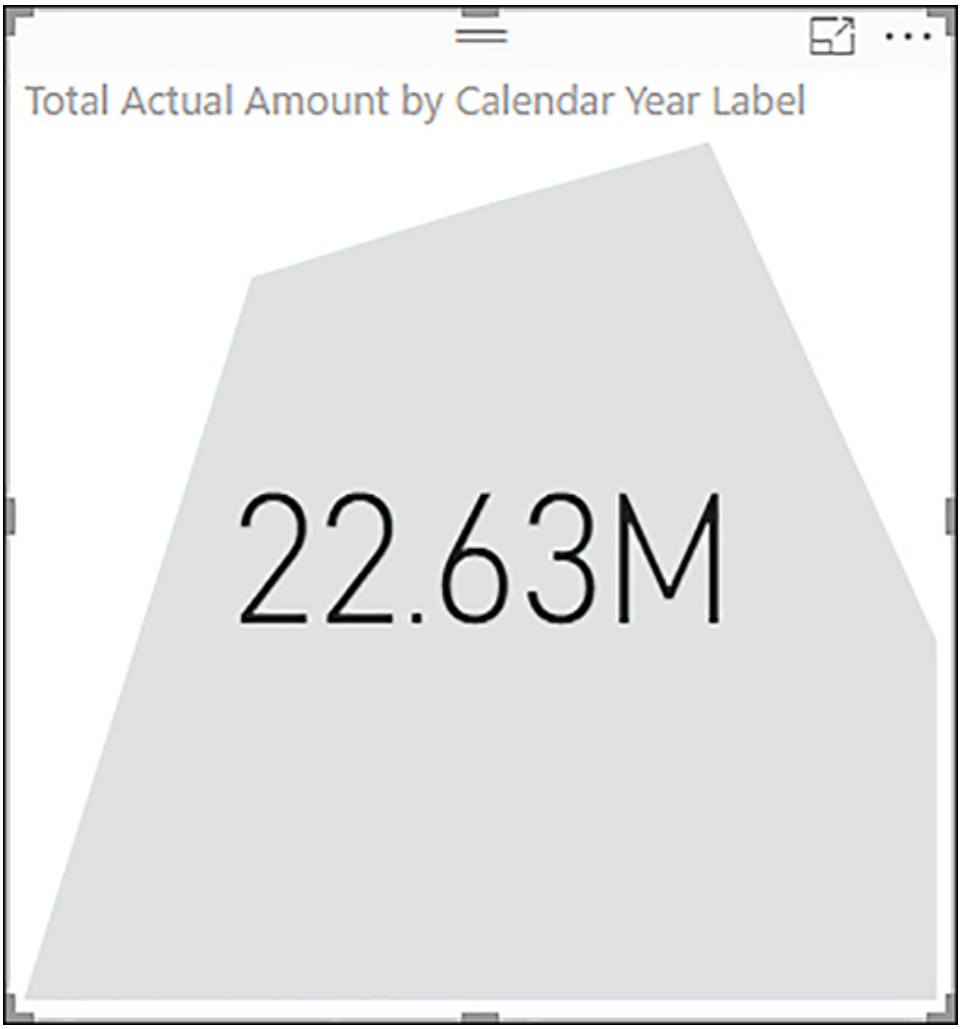


FIGURE 2.59 KPI visual with indicator and trend axis

The visual displays the indicator value that corresponds to the last point on the trend axis. In our case, it shows Total Actual Amount for CY2016. Behind the indicator value, its trend line with shaded underlying area is displayed.

At this point, add one or two target values to the Target Goals field well. If you add Target Amount, the KPI visual will look like the one in Figure 2.60.

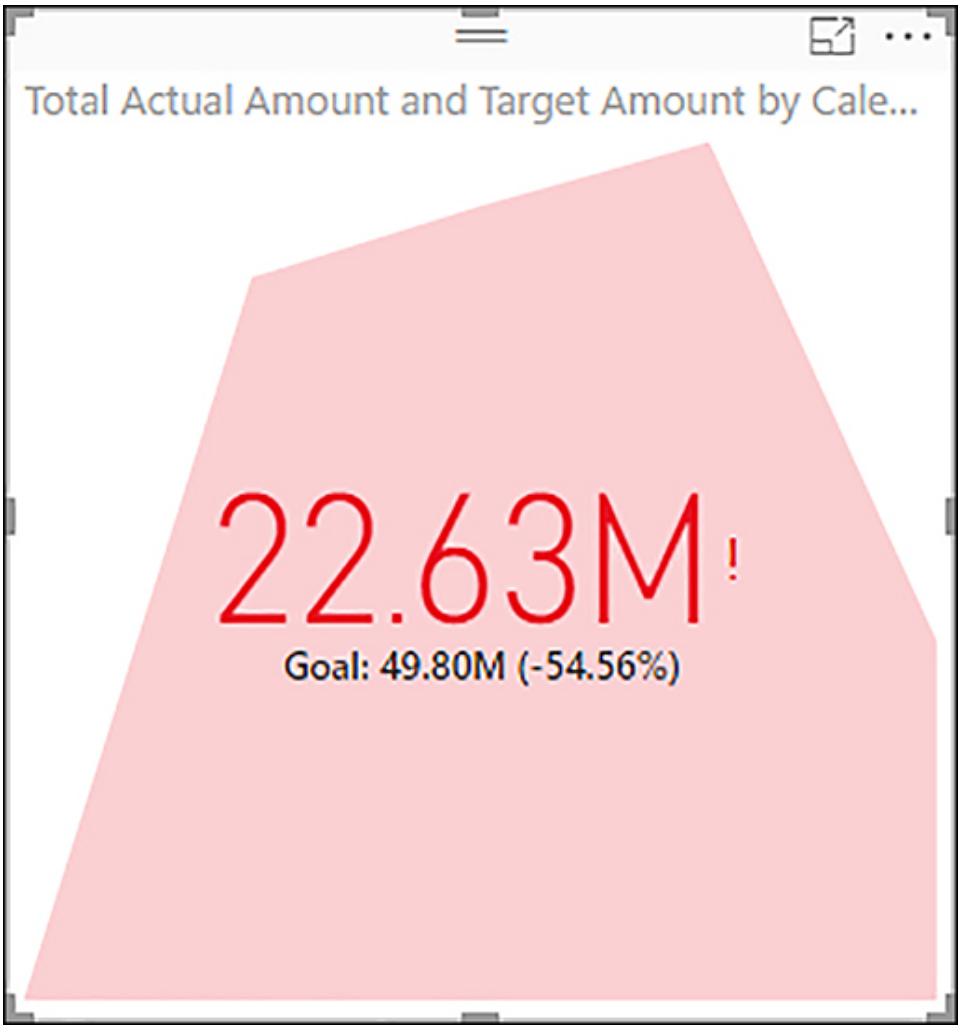


Figure 2.60 KPI visual with target goal

Note how below the indicator, you see the target goal value for the last trend axis point, as well as the percentage difference between the indicator and target goal.

When you use a target goal field in a KPI visual, you can take advantage of conditional formatting. The indicator itself and background will be colored, and an icon will be displayed to the right of the indicator. By default, if the indicator is below the target value, the former will be colored red along with the background; the icon displayed will be the exclamation sign. If the indicator is above the target goal, the color will be green, and a tick mark will be displayed to the right of the indicator.

You can also use two target goals in a KPI visual. In this case, if the indicator is below both targets, the color will be red; if it is above both targets, the color will

be green. If the indicator happens to be between the target goals, the color will be yellow, and a filled point will be displayed next to the indicator.

If for a certain KPI, the low indicator is good, you can change the conditional formatting behavior in the Direction drop-down list in the Color Coding section of the Format pane. The default behavior is called “High is good.” If you select “Low is good” as the direction, the background will be green if the indicator is below the target goal, and a tick mark will be displayed instead of the exclamation sign.

The Format pane has a brush icon and is next to the Fields pane; you can see it highlighted in Figure 2.61.

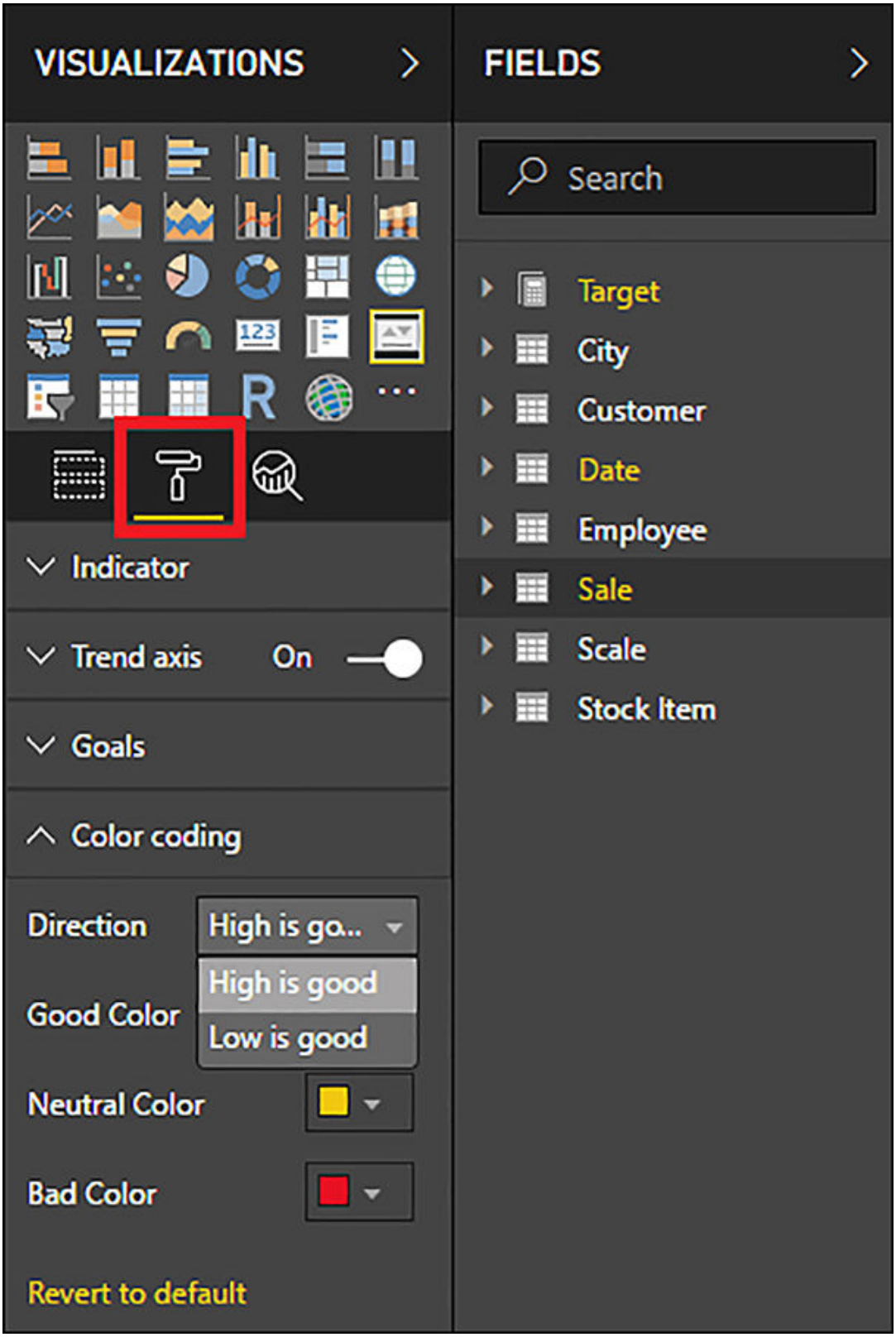


Figure 2.61 Format pane

In the Format pane, you can also turn on and off the goal value, distance, and trend axis. Also, you can adjust the default colors.

> ***MORE INFO*** **KPI VISUALS**
>
> For more details and a video tutorial on the KPI visuals, see "KPI visuals (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-kpi*.

Calculate actual to target

There are several ways in which you can calculate the actual to target, or variance, figure. To calculate the absolute amount, you can create a measure that references both the actual and target measures and subtracts one from another:

Click here to view code image

```
Actual to Target = [Total Actual Amount] - [Target Amount]
```

To calculate the percentage difference, you can either reference the Actual to Target measure, or you can calculate it in one measure only. The following two expressions provide equivalent results:

Click here to view code image

```
Actual to Target % = DIVIDE ( [Actual to Target], [Target Amount] )
Actual to Target % = DIVIDE ( [Total Actual Amount] - [Target
Amount], [Target Amount] )
```

The use of DIVIDE is important in this case because it handles division by zero, and the Target amount may be blank in case a user is browsing the model at the granularity level that Target Amount does not support. When Target Amount is (Blank), DIVIDE returns (Blank) as well. If we opted for the division operator instead, you would see "Infinity" in cases when Target Amount was (Blank).

Configure values for gauges

The Gauge visual has five field wells:

- Value
- Minimum value
- Maximum value
- Target value
- Tooltips

While the Gauge visual will be displayed with any of the field wells filled, it should at least have a field in the Value field well. When you place Total Actual Amount in the Value field well, you will see a visual like in Figure 2.62.

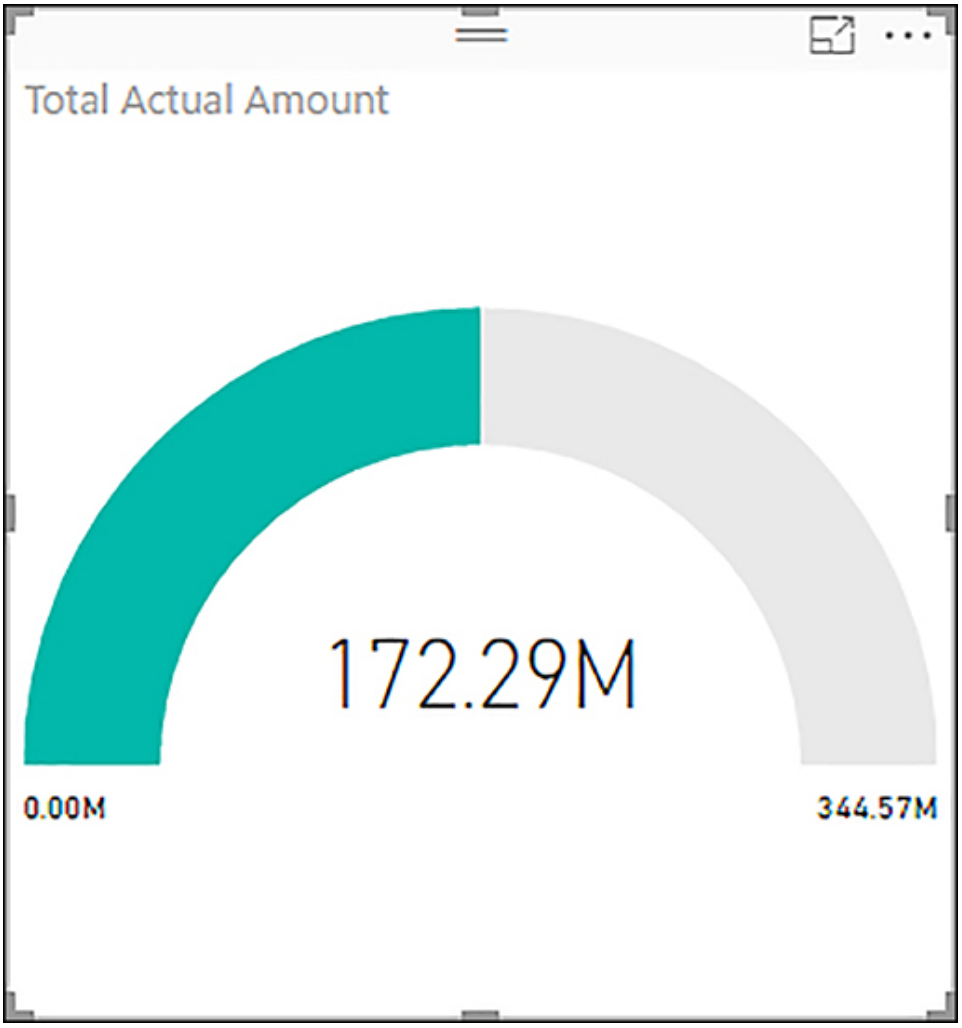


FIGURE 2.62 Gauge visual

The main value is displayed in the center. Since there is no trend axis, the value displayed is the total for all dates in the current filter context. By default, the visual sets the minimum at 0 and the maximum at double the amount of Value in this case, double the value of Total Actual Amount. As a result, the gauge is exactly half-filled. It is possible to adjust these figures by putting measures into the Minimum Value and Maximum Value field wells, respectively. For example, we can create the following measure and put it into the Maximum Value field well:

```
Gauge Maximum = 300 * 10 ^ 6
```

While this measure is hard-coded for example purposes, in real life, this can be a proper dynamic measure. Also, we can use our Target Amount measure in the Target Value field well. Once we add both measures to the visual, it will look like in Figure 2.63.

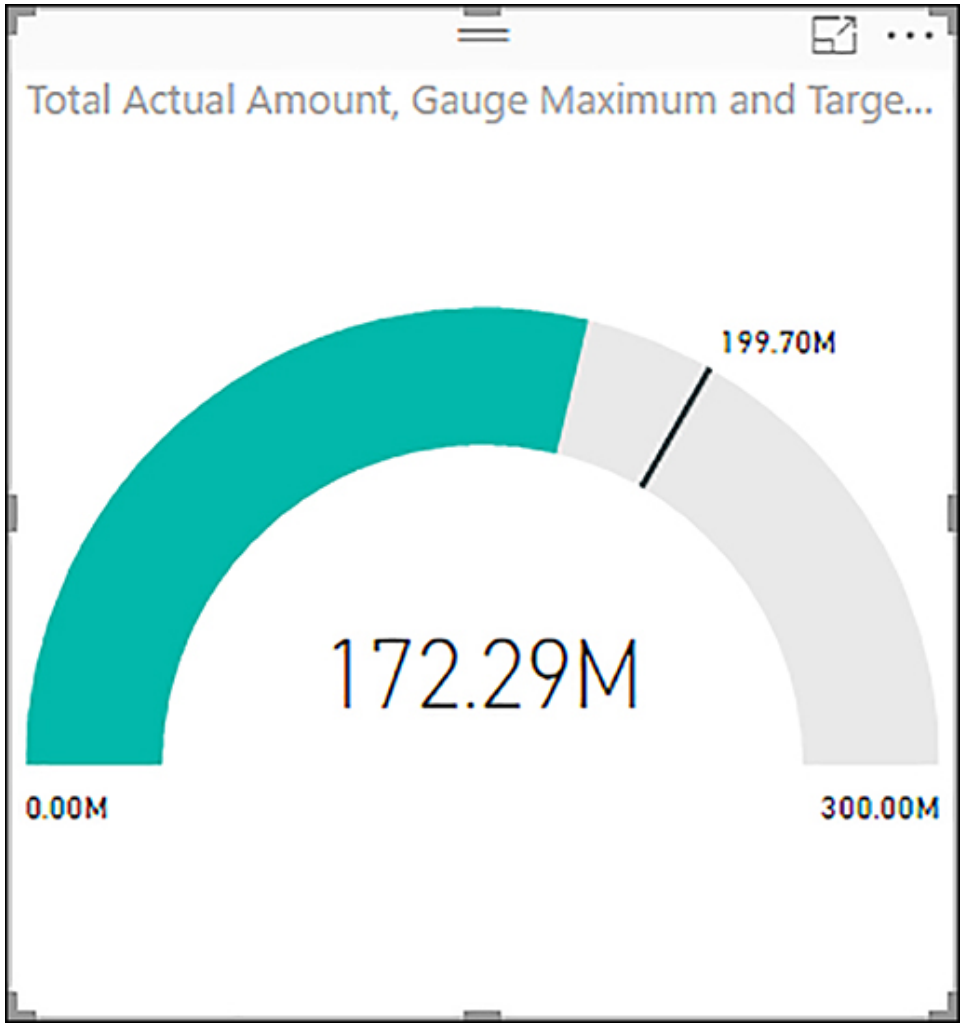


FIGURE 2.63 Gauge with maximum and target values set

The target amount is now shown as a black needle on the gauge.

The final field well that can be used is Tooltips. This field well can accept multiple measures and is not unique to the Gauge visual; many other visuals have it. When you place a measure into the Tooltips field well, you will be able to see its value when you hover over a data point in the visual. If you put Total Profit in the Tooltips field well and hover over the gauge, the visual will look like the one in Figure 2.64.

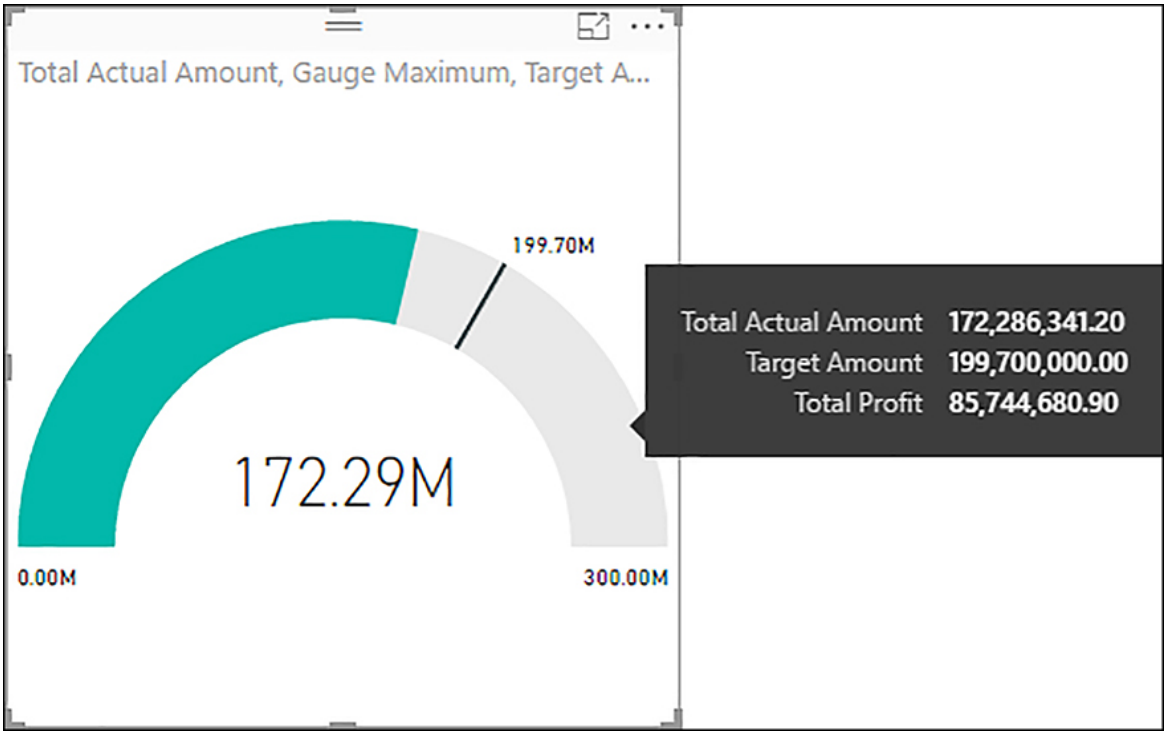


FIGURE 2.64 Tooltips

Note that tooltips do not necessarily contain all the measures that are used in a visual. In our case, the Gauge Maximum measure is absent.

> ***MORE INFO* TOOLTIPS**
>
> Tooltips can be a powerful addition to many visuals. Besides numeric values, they can also display datetime and text measures. For more examples and details on the Tooltip feature, see "Customizing Tooltips in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-custom-tooltips*.

Use the format settings to manually set values

Besides using measures in field wells of a Gauge, you can set the minimum, maximum, and target values manually in the Format pane. If you set a value in a field well, you will not be able to override it in the Format pane. For instance, if we keep the Gauge Maximum measure but remove the Target Amount measure and go to the Gauge Axis section in the Format pane, we will be able to set both the minimum and target values. The settings can be seen in Figure 2.65.

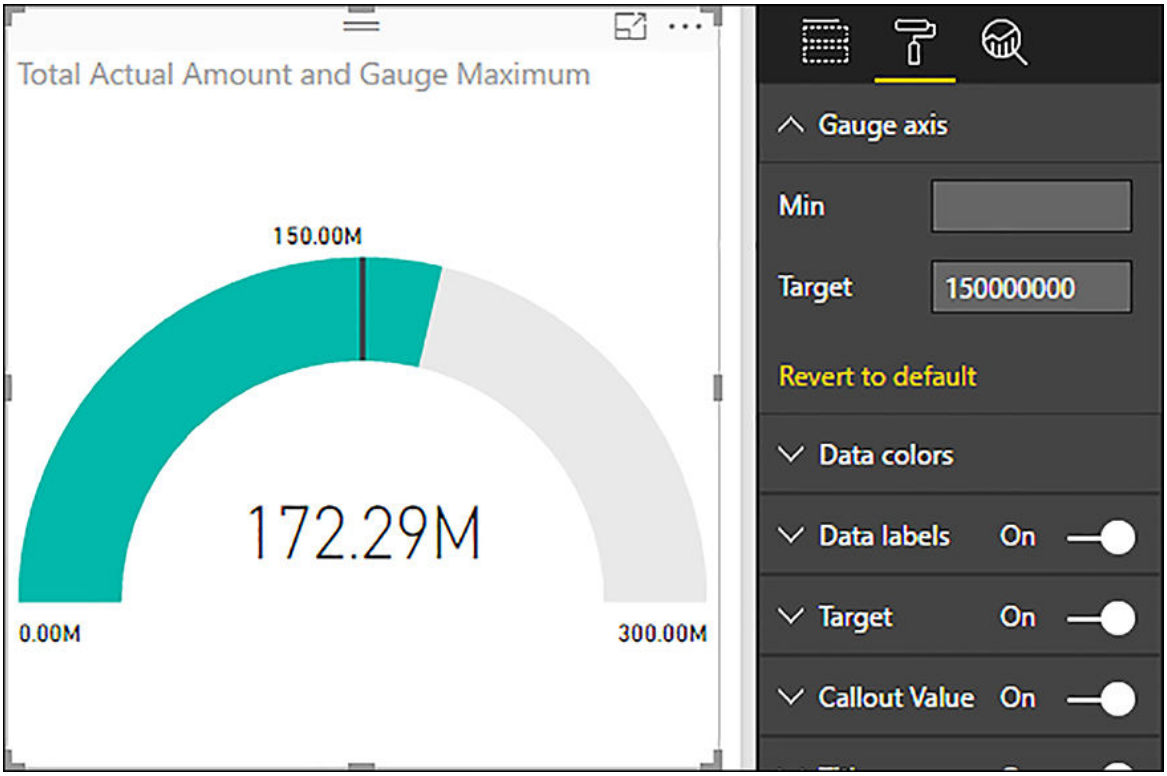


FIGURE 2.65 Gauge axis format settings

You can use the Format pane to do the following in the Gauge:

- Change the colors of the gauge and the target line
- Hide or format the minimum and maximum values, target, and the main value

MORE INFO **GAUGES AND SINGLE-NUMBER CARDS**

For more examples and information on using card visuals and gauges, see "Gauges and single-number cards" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/visualizations#step-9*.

For a video tutorial on using the Gauge visual, see "Radial gauge charts in Power BI (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-radial-gauge-charts*.

Skill 2.4: Create hierarchies

Hierarchies can make browsing your data model easier for end users. In Power BI Desktop, there are hierarchies of two kinds: the automatic date hierarchies and user-defined, or custom, hierarchies. While you can only either enable or disable

the automatic date hierarchies, you have greater flexibility when building your own hierarchies; you can rename both the hierarchy and each element, and you can reorder the levels of a hierarchy.

This section covers how to:

- Create date hierarchies
- Create hierarchies based on business needs
- Add columns to tables to support desired hierarchy

Create date hierarchies

Date hierarchies are used very often because many reports deal with dates. You can take advantage of the built-in Power BI date hierarchies, or you can create your own. Using the built-in date hierarchies means less work, though it carries some limitations while creating your own hierarchies gives you the greatest flexibility.

By default, Power BI creates a date hierarchy for any date or datetime type of column. When you put a date or datetime column in a visual that supports hierarchies, you will see that Power BI adds a date hierarchy instead of a single date column. You can review this behavior by creating a table visual and adding the Date column from the Date table to it. The table will look like in Figure 2.66.

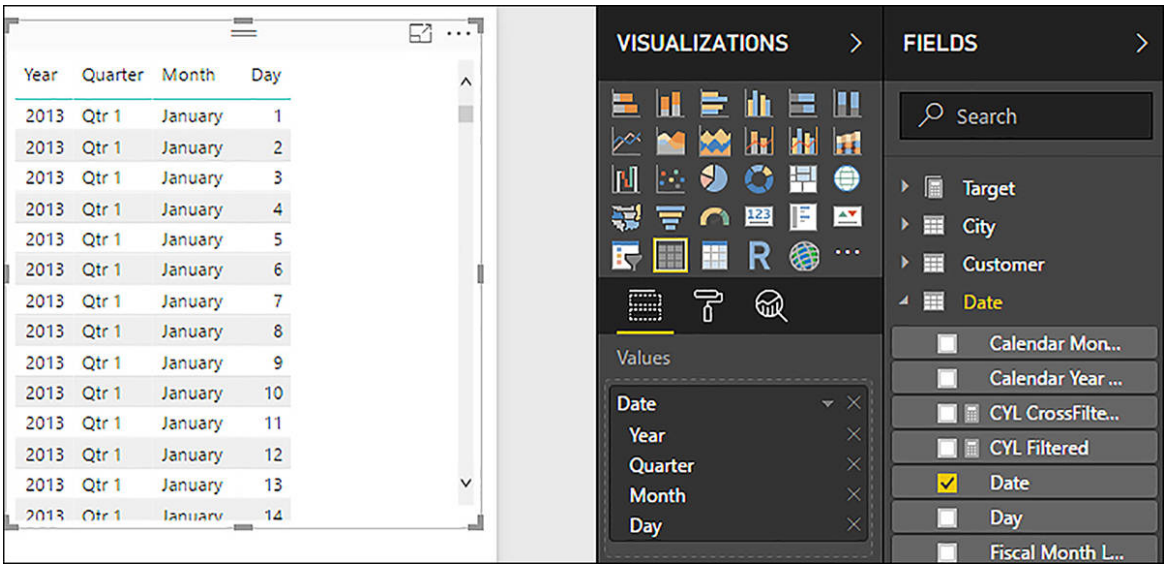


FIGURE 2.66 Date hierarchy

At this stage, note that in the Values field well of the table visual you have the Date hierarchy instead of the Date column, and it has four levels:

- Year
- Quarter
- Month
- Day

You cannot edit the hierarchy because it is automatically created by Power BI, but you can remove some of the levels that are displayed by clicking on the crosses next to the hierarchy level names. If, after removing some levels, you want to show all levels again, you can click on the down arrow next to the hierarchy name, or right-click on it, and select Show All Levels.

To switch between the date hierarchy and the actual column from which it is built, you can right-click on the hierarchy and select the column name instead of the hierarchy. In our example, it would be Date instead of Date Hierarchy. This will replace all the hierarchy columns both in the table visual and the field well with a single column.

The advantage of using automatic date hierarchies is that they are convenient because they exist by default, and you do not need to create them. Furthermore, even if a datetime column contains time portions in addition to dates, the hierarchy will contain dates only, which is essential for any Time Intelligence functions.

You can reference all columns in the automatic date hierarchies with special syntax called variation, or dot notation. For instance, if you want to create a calculated column in the Date table that concatenates the Year and Quarter columns from the automatic date hierarchy, you can write the following formula:

```
Year Quarter = 'Date'[Date].[Quarter] & " " & 'Date'[Date].[Year]
```

IntelliSense can suggest you these columns in the same way as other columns as soon as you type `‘Date’[Date].` The dot notation, in this case, works similar to the RELATED function, though it is currently unique to the automatic date hierarchies.

It is important to note that by default, Power BI creates such a hierarchy for every date or datetime column, and it starts on 1 January of the minimum year and ends at 31 December of the maximum year. This hierarchy is a hidden table. Therefore, even if you have a single-row table with a date column, Power BI will build an automatic date hierarchy with 365 or 366 rows. While in this particular

case this may not be significant, it can cause the data model size to bloat, especially in cases where dates can go up to 31 December 9999.

If automatic date hierarchies are undesirable, they can be turned off for each file individually. To do that, select **File** > **Options and settings** > **Options, then Current File** > **Data Load** > **Time intelligence** > **Auto Date/Time**. Doing so turns off automatic date hierarchies for the whole data model; keeping some date hierarchies, but not others, is not possible.

> ***MORE INFO*** **WORKING WITH AUTOMATIC DATE HIERARCHIES**
>
> For more examples and details on how you can use the automatic date hierarchies, see "Visual hierarchies and drill-down" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/visualizations#step-18*.

Create hierarchies based on business needs

While the automatic date hierarchies can be convenient, you cannot add new columns to them. Furthermore, they are of no use if you use a custom calendar such as 4-4-5, which is popular in retail, for example. For these reasons, you might prefer to use your own calendar table and enrich it with your own date hierarchies. The principles with which you create a custom date hierarchy are the same for any custom hierarchy, so in the next example, we are going to build a custom date hierarchy.

A hierarchy can be created only using existing columns. The columns also need to be in the same table; if you want to use a column from another table in a hierarchy, you will need to add it with the RELATED function.

Once you have all desired columns in the same table, you can right-click on a column and select New Hierarchy. Preferably, this should be the column that will be at the highest level, because that way you will save time on reordering hierarchy levels later. For example, you can right-click on the Calendar Year Label column and choose New Hierarchy, which will create a hierarchy called Calendar Year Label Hierarchy with a hierarchy icon next to it.

Another way to create a hierarchy is to drag and drop one column on top of another. The column you are dragging should be the second level of the hierarchy.

To add an element to an existing hierarchy, you need to right-click on a column and select Add to Hierarchy, then select the hierarchy to which you want to add it.

Alternatively, you can drag and drop columns to the hierarchy of your choice. We should add the following columns to our custom hierarchy:

- Calendar Month Label
- Date

You can rename a hierarchy or any of its elements by either double-clicking on the name or right-clicking on it and selecting Rename. In our case, we should rename the hierarchy to “Calendar Y-M-D” and its elements as follows:

Old name	New name
Calendar Year Label	Year
Calendar Month Label	Month
Date	Date

Note that when you rename a hierarchy element, the original column name remains as is. At the same time, you do not need to sort a hierarchy element by another column, because it inherits this property from the original column. The original column can be hidden, if necessary.

The resulting hierarchy once added to a table visual, can be seen in Figure 2.67.

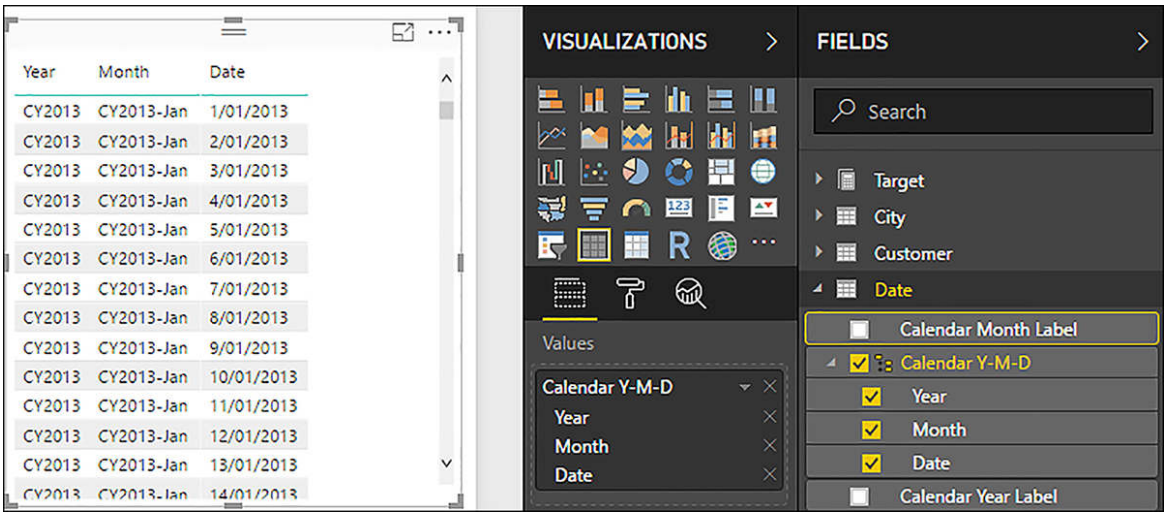


FIGURE 2.67 Calendar Y-M-D

Note that hierarchies can be fully replaced by using individual columns together. Creating hierarchies only saves time when you frequently use the same columns together–they do not provide any additional benefit.

For instance, you can re-create the table from Figure 2.67 by adding Calendar Year Label, Calendar Month Label, and Date to the Value field well. You can even rename the columns within the visual by either right-clicking on their names in the Values field well and selecting Rename or double-clicking on the names. The result can be seen in Figure 2.68.

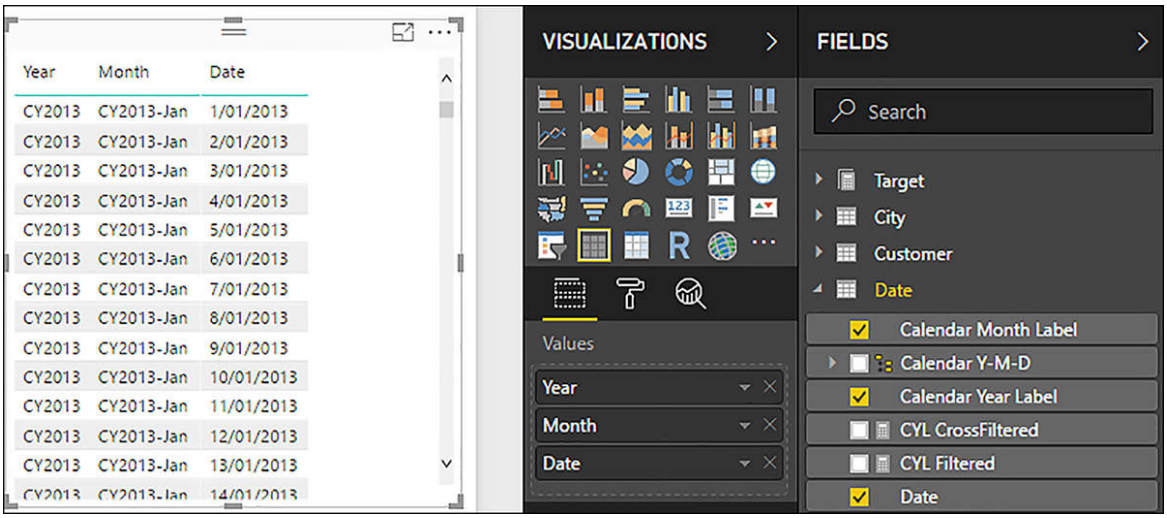


FIGURE 2.68 Hierarchy created with columns only

If you have not disabled automatic date/time hierarchies, you must switch from Date Hierarchy to Date to replicate the Calendar Y-M-D hierarchy, otherwise you will have a Date hierarchy in addition to the Year and Month columns. Note how you can use both columns and hierarchies together in the Values field well.

EXAM TIP

Be aware that using a hierarchy and individual columns together are fully equivalent options for exam purposes.

To reorder elements of a hierarchy, you need to drag and drop elements within the hierarchy in the Fields pane. A yellow line will show you where the element you are dragging will end up. Alternatively, you can click on a hierarchy element and select Move Up or Move Down.

***MORE INFO* DRILL DOWN USING HIERARCHIES**

Hierarchies can be especially useful when you use them to drill down in visuals such as column charts or matrixes. This topic is outside of the

scope of this book, but you can read more about different ways to drill down in a visual at "Drill down in a visualization in Power BI" at: *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-drill-down*.

For a video tutorial on the topic, see "Visual hierarchies and drill-down" at *https://docs.microsoft.com/en-us/power-bi/guided-learning/visualizations#step-18*.

Add columns to tables to support desired hierarchy

As mentioned above, if your hierarchy needs to contain columns from different tables, you need to bring all columns into a single table first with the RELATED function. Usually this is the table with the lowest granularity. For instance, if you had three tables connected with one-to-many relationships—Product, Subcategory, and Category—you create your hierarchy in the Product table.

There are cases when you need to create new columns from existing columns to enable hierarchy creation. A good example of this is parent-child (PC) hierarchies. For example, an organization might have the following hierarchical structure stored in a database:

TABLE 2-17 Sample organizational structure

Employee	Manager
Gustavo	
Luka	Gustavo
Hazem	Gustavo
Kim	Luka
Ahmad	Luka
Michael	Hazem
David	Ahmad
Terry	Ahmad

At its current state, it is not possible to create a meaningful hierarchy in this table because there are several levels in it but only two columns. DAX has several functions with which you can flatten parent-child hierarchies.

The first function that we are going to review is PATH, which receives two parameters: the ID column and the parent column. In our case, you can use the Employee column as the ID column and the Manager column as the Parent column. If we name a table "PC," the formula we should use is as follows:

```
Path = PATH ( PC[Employee], PC[Manager] )
```

> ***IMPORTANT BLANKS AND EMPTY STRINGS***
>
> When there is a top-level employee that has no parent, it must have a blank value in place of its parent. If instead, there is an empty string, you will see an error message like this: "The value '' in 'PC'[Manager] must also exist in 'PC'[Employee]. Please add the missing data and try again." For more information on the PATH function, see "PATH Function (DAX)" at *https://msdn.microsoft.com/en-us/library/gg492167.aspx*.

The function returns a pipe-delimited list of all parents, starting with the top parent, and ending with the current ID. Adding all parent-child columns to the PC table provides results like those shown in Table 2-18.

TABLE 2-18 Path column added

Employee	Manager	Path
Gustavo		Gustavo
Luka	Gustavo	Gustavo|Luka
Hazem	Gustavo	Gustavo|Hazem
Kim	Luka	Gustavo|Luka|Kim
Ahmad	Luka	Gustavo|Luka|Ahmad
Michael	Hazem	Gustavo|Hazem|Michael

As you can see, Kim, Ahmad, and Michael all have two managers above them. You can get the number of levels in a hierarchy using the PATHLENGTH function, which receives the path column as its only parameter:

```
Levels = PATHLENGTH ( PC[Path] )
```

The PC table with the Levels column added can be seen in Table 2-19.

TABLE 2-19 Levels column added

Employee	Manager	Path	Levels
Gustavo		Gustavo	1
Luka	Gustavo	Gustavo|Luka	2
Hazem	Gustavo	Gustavo|Hazem	2
Kim	Luka	Gustavo|Luka|Kim	3
Ahmad	Luka	Gustavo|Luka|Ahmad	3
Michael	Hazem	Gustavo|Hazem|Michael	3

Note that the maximum value in the Levels column is 3, confirming the previous observation. At this point, you can add three columns; one for each level of the hierarchy as follows:

Click here to view code image

```
Level 1 = PATHITEM ( PC[Path], 1 )

Level 2 = PATHITEM ( PC[Path], 2 )

Level 3 = PATHITEM ( PC[Path], 3 )
```

Once you add the above columns, the PC table will look like Table 2-20.

TABLE 2-20 Hierarchy levels added as columns

Employee	Manager	Path	Level 1	Level 2	Level 3
Gustavo		Gustavo	Gustavo		
Luka	Gustavo	Gustavo|Luka	Gustavo	Luka	
Hazem	Gustavo	Gustavo|Hazem	Gustavo	Hazem	
Kim	Luka	Gustavo|Luka|Kim	Gustavo	Luka	Kim
Ahmad	Luka	Gustavo|Luka|Ahmad	Gustavo	Luka	Ahmad
Michael	Hazem	Gustavo|Hazem|Michael	Gustavo	Hazem	Michael

***MORE INFO* PARENT-CHILD HIERARCHIES IN DAX**

At this point, you can already create a hierarchy using the Level 1, Level 2, and Level 3 columns. If you had a column to aggregate, it would correctly show the amounts, but it would also show amounts even if a

level is blank, like in the case of Level 2 and Level 3 for Gustavo. If this is undesirable, you can use the technique developed by Marco Russo: see "Parent-Child Hierarchies" at:*https://www.daxpatterns.com/parent-child-hierarchies/*.

There is also the PATHITEMREVERSE function in DAX, which works like the PATHITEM function, but counts from the end. If you use the following calculated column formulas instead of the preceding ones, our table looks like Table 2-21.

Click here to view code image

```
Level 1 = PATHITEMREVERSE ( PC[Path], 1 )

Level 2 = PATHITEMREVERSE ( PC[Path], 2 )

Level 3 = PATHITEMREVERSE ( PC[Path], 3 )
```

TABLE 2-21 PC table with PATHITEMREVERSE level columns

Employee	Manager	Path	Level 1	Level 2	Level 3
Gustavo		Gustavo	Gustavo		
Luka	Gustavo	Gustavo|Luka	Luka	Gustavo	
Hazem	Gustavo	Gustavo|Hazem	Hazem	Gustavo	
Kim	Luka	Gustavo|Luka|Kim	Kim	Luka	Gustavo
Ahmad	Luka	Gustavo|Luka|Ahmad	Ahmad	Luka	Gustavo
Michael	Hazem	Gustavo|Hazem|Michael	Michael	Hazem	Gustavo

Both PATHITEM and PATHITEMREVERSE can receive an optional third parameter: specifying 0 or omitting the parameter makes the functions return text strings while specifying 1 makes the functions return integers.

Finally, there is the PATHCONTAINS function that checks whether an item is part of a particular path. The function receives two parameters: the path and an item to check. The PATHCONTAINS function can be especially useful when you enable row-level security, which is covered in Chapter 3, "Configure dashboards, reports, and apps in the Power BI Service." For review purposes, use a hardcoded value of "Luka" to check whether it appears in the path:

Employee	Manager	Path	Luka in Path
Gustavo		Gustavo	False
Luka	Gustavo	Gustavo|Luka	True
Hazem	Gustavo	Gustavo|Hazem	False
Kim	Luka	Gustavo|Luka|Kim	True
Ahmad	Luka	Gustavo|Luka|Ahmad	True
Michael	Hazem	Gustavo|Hazem|Michael	False

MORE INFO **PARENT-CHILD HIERARCHY FUNCTIONS IN DAX**

For more examples and information on the parent-child functions in DAX, see "Understanding Functions for Parent-Child Hierarchies in DAX" at *https://msdn.microsoft.com/en-us/library/gg492192.aspx*.

Skill 2.5: Create and format interactive visualizations

Once you have connected to the data sources and created a data model, the next step is to create visualizations. One of the things that sets Power BI apart from other visualization tools is that you can create interactive visualizations—that is, visuals, that can interact with each other by cross-filtering the underlying data.

This section covers how to:

- Select a visualization type
- Configure page layout and formatting
- Configure interactions between visuals
- Configure duplicate pages
- Handle categories that have no data
- Configure default summarization and data category of columns
- Position, align, and sort visuals
- Enable and integrate R visuals
- Format measures

Select a visualization type

So far in this chapter, we have already worked with some visuals, including Gauge, Card, Multi-row Card, KPI, Slicer, Table, and Matrix. In this section, we are going to review some other standard Power BI visuals.

> ***NOTE*** **CHOOSING THE RIGHT VISUAL**
>
> SQLBI has created a concise reference that can help you with selecting the right visual. It lists all the standard visuals and many custom ones. For more details, see "Power BI Visuals Reference" at *https://www.sqlbi.com/ref/power-bi-visuals-reference/*.

Bar charts

Power BI has six variations of bar charts:

- Stacked bar chart
- Stacked column chart
- Clustered bar chart
- Clustered column chart
- 100% Stacked bar chart
- 100% Stacked column chart

All six charts share the same five field wells:

- **Axis** You can use one or more categorical columns in this field well.
- **Legend** One categorical column can be used.
- **Value** You can use one or more numerical fields; if you use a legend or color saturation, you can only put one field into this field well.
- **Color saturation** One numerical field can be used.
- **Tooltips** You can use one or more fields here

You can see all six variations of bar charts in Figure 2.69 with titles manually set to reflect the chart type.

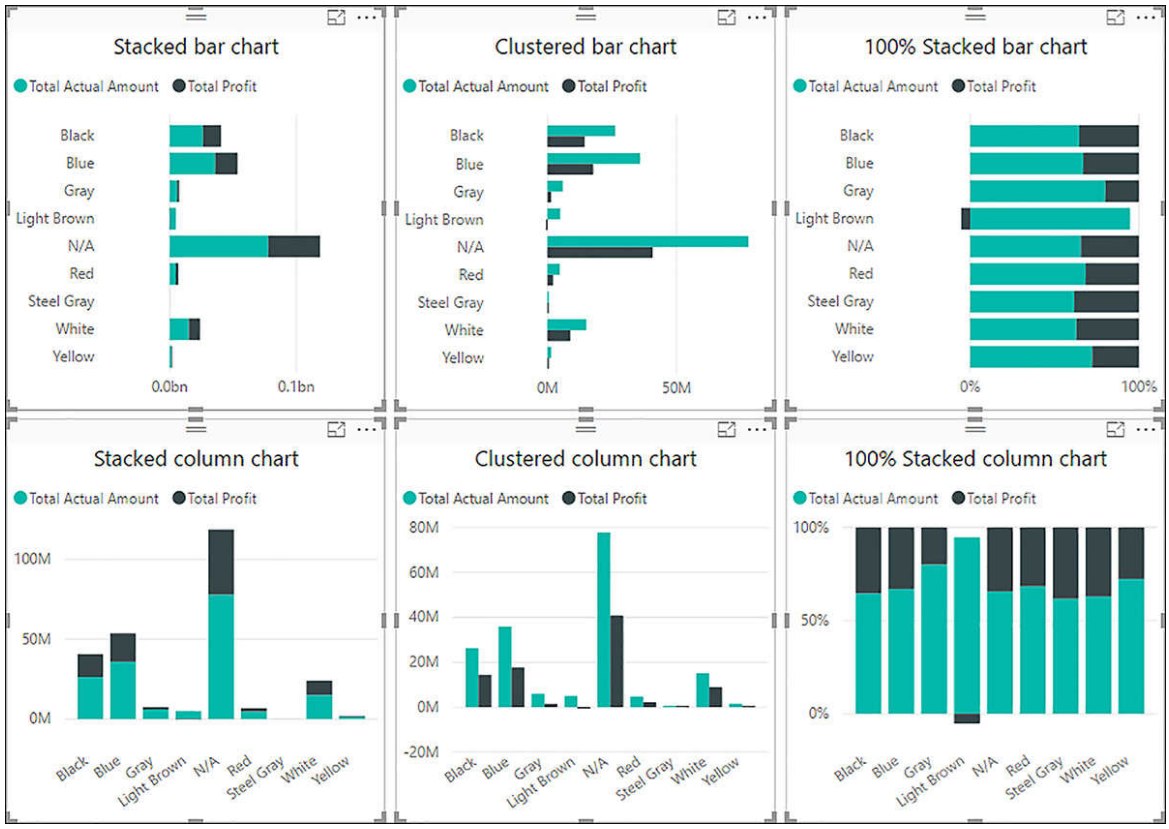


FIGURE 2.69 Six bar chart types

It is best to use bar charts when you are comparing values across categories. If you want to compare values across time, it is best to use line charts, which are covered next.

***MORE INFO* FORMATTING VISUALS**

You can customize many items in the Format pane, including the title, legend, background, axes and change colors in the Format pane. To learn more, see ”Customize visualization titles, legends, and backgrounds (Tutorial)” at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-customize-title-background-and-legend*, and “Getting started with color formatting and axis properties” at *https://docs.microsoft.com/en-us/power-bi/service-getting-started-with-color-formatting-and-axis-properties*.

Line and area charts

Power BI has a line chart and two area charts, which are similar to the line chart, but have a shaded area under the lines:

- Line chart
- Area chart
- Stacked area chart

All three charts have the following field wells:

- **Axis** For one or more categorical columns.
- **Legend** For one categorical column.
- **Values** One or more numerical fields can be used; if you use a legend, you can only use one field in this well.
- **Tooltips** One or more fields can be used.

The three charts can be seen in Figure 2.70.

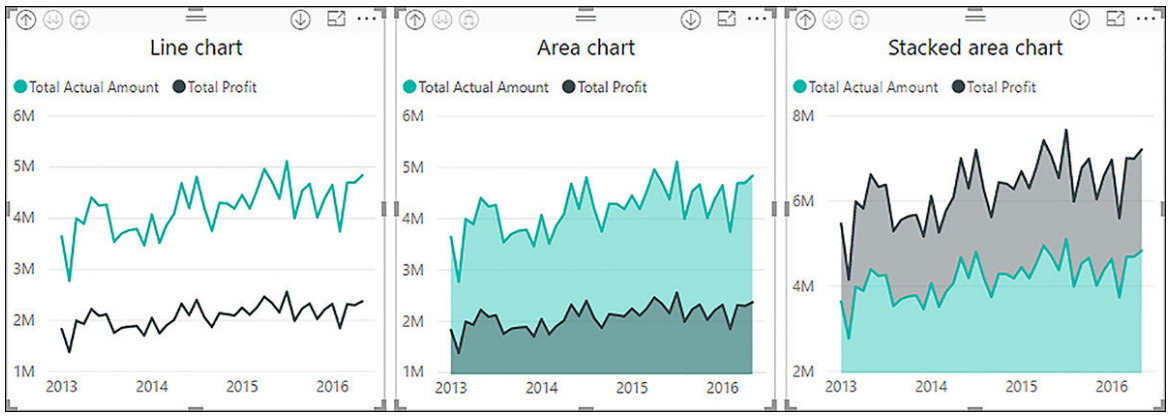


FIGURE 2.70 Line, area, and stacked area charts

Line charts are best used when you are comparing values across time.

> ***MORE INFO*** **AREA CHART**
>
> For a tutorial on a basic area chart, as well as considerations on using it, see "Basic Area chart (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-basic-area-chart*.

Combo charts

There are two combo charts in Power BI:

- Line and stacked column chart
- Line and clustered column chart

Both charts have five field wells:

- **Shared axis** One or more categorical columns.

- **Column series** A legend well in which you can use a categorical column.
- **Column values** You may use one or more numerical fields in this well; if you have a column series, you can only use one field.
- **Line values** One or more numerical fields may be used in this well.
- **Tooltips** One or more fields may be used in this well.

You can see the two charts used in Figure 2.71.

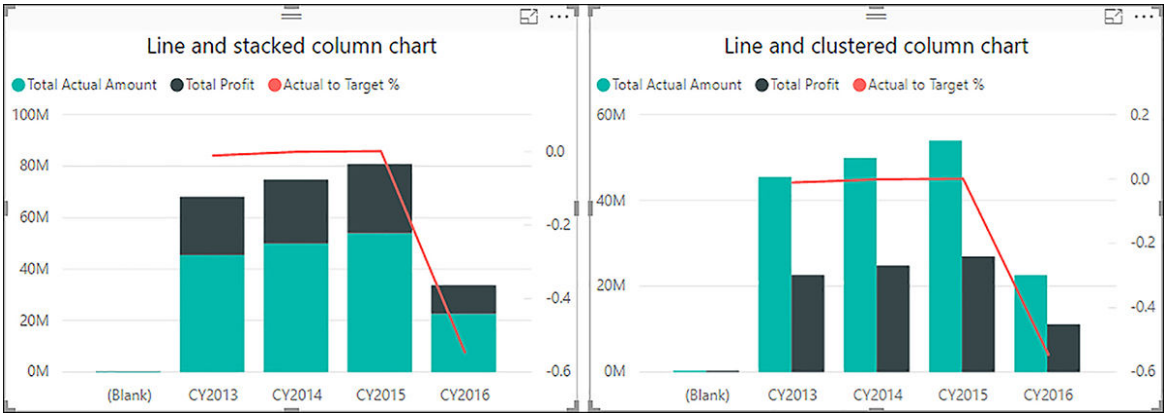


FIGURE 2.71 Combo charts

Combo charts can be the appropriate choice when you plot two fields that have very different value range for instance, dollar amounts and percentages. By default, the column values appear on the right, and the line values appear on the left. In the Format settings, you can switch the axes, as well as hide one or the other.

> ***MORE INFO*** **COMBO CHARTS**
>
> For more information on combo charts, including areas of application and formatting options, see "Combo Chart in Power (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-combo-chart*.

Ribbon chart

The Ribbon chart is similar to a column chart, but it has ribbons between the bars to highlight changes in the relative ranking of categorical items. The item with the highest value will be displayed on top.

The chart has four field wells:

- **Axis** One or more categorical columns.

- **Legend** You may use one categorical column.
- **Value** You can use one or more numerical fields in this well; if you use a legend, you can use one field only.
- **Tooltips** You may use one or more fields.

You can see a ribbon chart used in Figure 2.72.

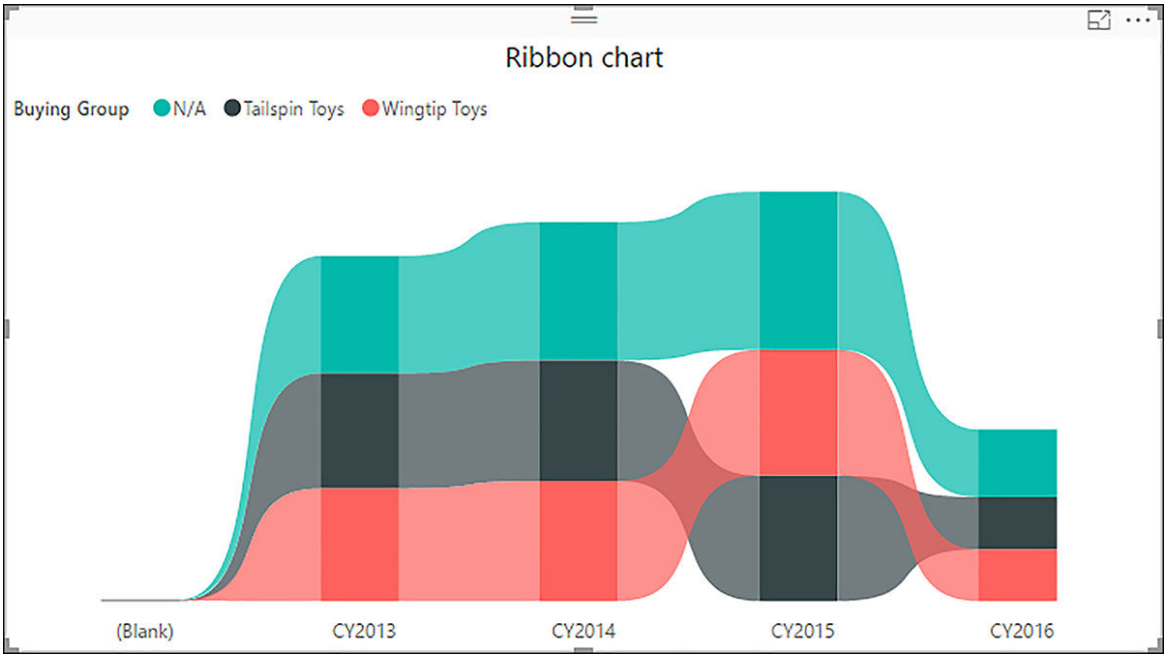


FIGURE 2.72 Ribbon chart

Note how Figure 2.72 shows that almost every year, the ranking of Buying Group items by Total Actual Amount is as follows:

1. N/A
2. Tailspin Toys
3. Wingtip Toys

The exception is CY2015, in which Wingtip Toys overtook Tailspin Toys as number two.

***MORE INFO* RIBBON CHART**

For more information on using the Ribbon chart, including formatting options, see "Use ribbon charts in Power BI" at *https://docs.microsoft.com/en-us/power-bi/desktop-ribbon-charts*.

Waterfall chart

The Waterfall chart shows color-coded values in a running total fashion. By default, positive values are green, while negative values are red. This visual has four field wells:

- **Category** One or more categorical columns.
- **Breakdown** You may use one categorical column.
- **Y Axis** One numerical field.
- **Tooltips** You may use one or more fields.

You can see two waterfall charts: one with and one without a breakdown for three calendar years in Figure 2.73.

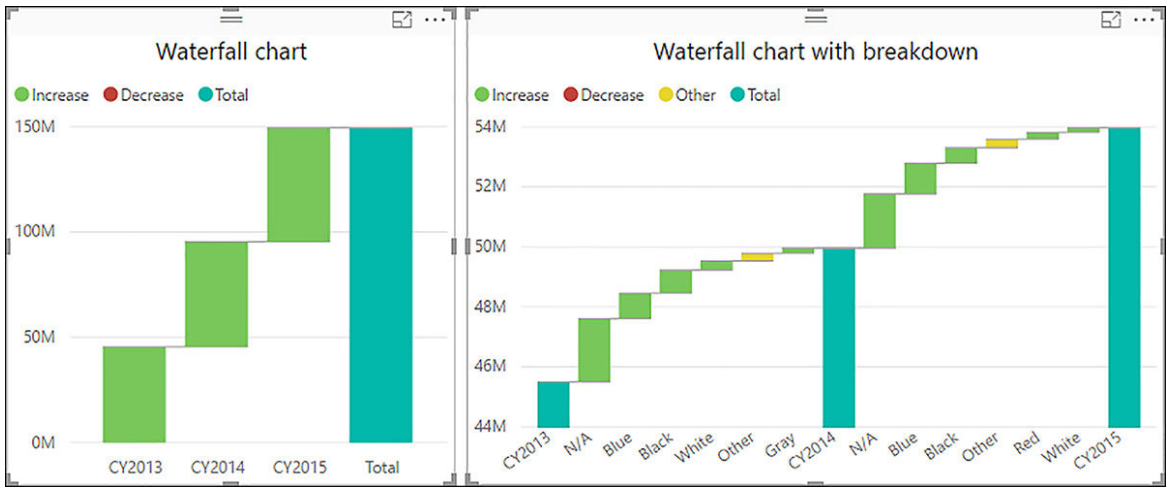


FIGURE 2.73 Waterfall charts

Note how when you use breakdown, only the top five items are shown by default, with the rest being grouped into Other. The Other group is yellow because it may contain both Increase and Decrease items. You can change the colors, as well as number of breakdowns in the Formatting pane.

IMPORTANT TOTAL IN WATERFALL CHART

Be aware that the total in waterfall chart is an arithmetic sum of the items, not the value that you would see without filtering the category, as is the case in the Table visual, for example. This behavior is especially relevant with semi-additive or non-additive measures, such as balances or averages.

A waterfall chart is a good choice when you want to show the major changes or illustrate what the total amount is made of while showing the total at the same time for comparison.

> ***MORE INFO* WATERFALL CHART**
>
> For more information on how you can use the Waterfall visual, including a tutorial, see "Waterfall charts in Power BI (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-waterfall-charts*.

Scatter chart

The Scatter chart can visualize two or more metrics for categorical items. Each item will be plotted according to the X and Y coordinates, which will be taken from the metrics. When you use a third metric for size, the chart can be called a bubble chart. The visual has eight field wells:

- **Details** You can use one or more categorical items in this field well; if you use more than one column, you can drill down.
- **Legend** You may use one categorical column.
- **X Axis** One numerical field.
- **Y Axis** One numerical field.
- **Size** One numerical field.
- **Color saturation** One numerical field may be used; when you use legend, you cannot use color saturation.
- **Play Axis** One categorical column ideally, but not necessary, the time column.
- **Tooltips** You may use one or more fields.

An example of a scatter chart is shown in Figure 2.74.

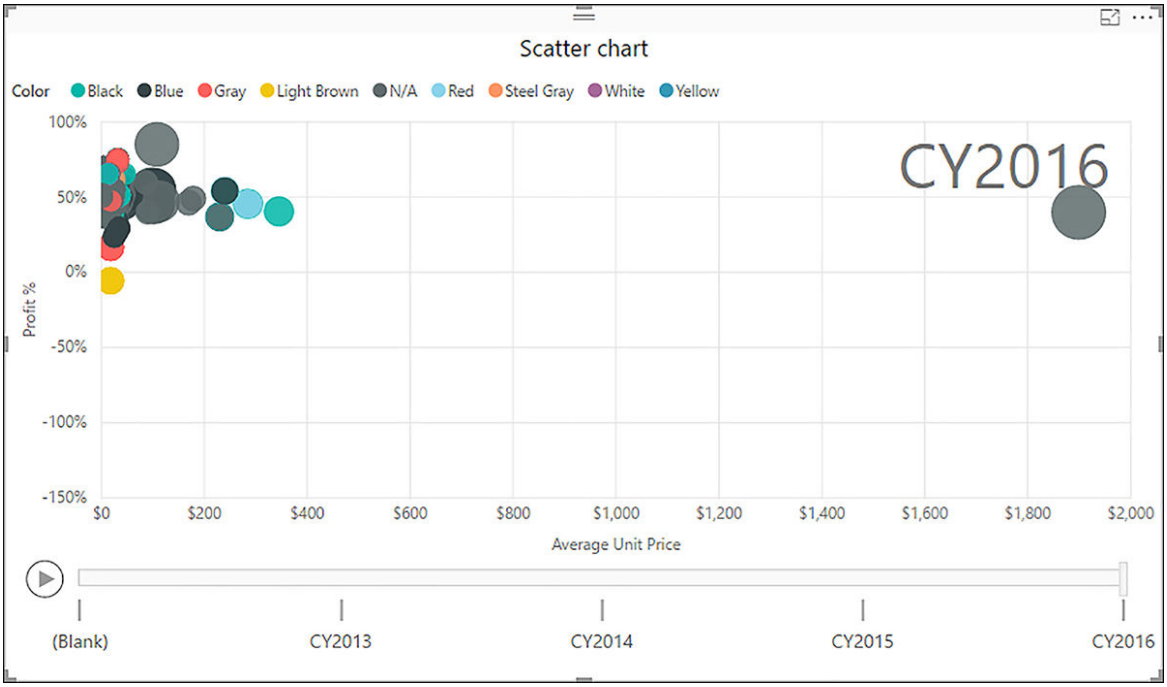


Figure 2.74 Scatter chart

Note how easy it is to see one item that is an outlier: its price is almost five times the second-most expensive price. If you hover over the bubble, you see that it is Air cushion machine (Blue). Scatter charts can be a powerful way to show a relationship between two metrics or highlight outliers.

> ***MORE INFO* SCATTER CHART**
>
> For a video tutorial and more details on how scatter charts can be used, including formatting options, see "Scatter charts and bubble charts in Power BI (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-scatter*.

Pie and doghnut charts

The Pie and Doghnut charts are the same except the latter has empty space in the middle. Both charts have four field wells:

- **Legend** For one or more categorical columns.
- **Details** You may use a categorical column.
- **Values** You may use one or more numerical fields here; if you use Details, you are limited to one field only.
- **Tooltips** You may use one or more fields here.

You can see both charts shown in Figure 2.75:

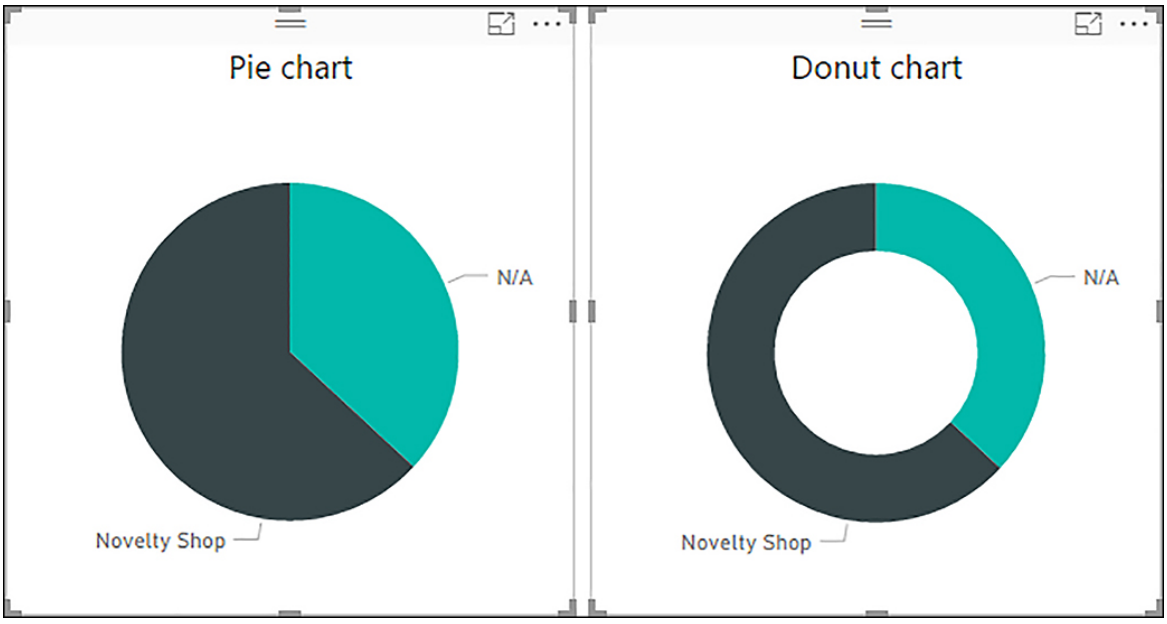


Figure 2.75 Pie and Donut charts

You can use these charts to show the relationships of parts to the whole. Both charts are not considered best practice in data visualization with one exception: you may use a pie or a doghnut chart when there are only two categories. Otherwise, if you are sure that users will not be comparing parts to each other, only to the whole, you may use these charts. But be aware that having too many items will make reading and interpreting values difficult.

> ***MORE INFO* DOGHNUT CHART**
>
> For more details on the Doghnut chart, including considerations, see "Doughnut charts in Power BI (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-doughnut-charts*.

Treemap

Treemap charts can be thought of as rectangular pie charts because they also show the relationship of the parts to the whole. You can nest rectangles to further divide the whole. There are five field wells:

- **Group** For one or more categorical columns.
- **Details** You may use a categorical column.

- **Values** You may use one or more numerical fields here; if you use Details, you are limited to one field only.
- **Color saturation** You may use a numerical field here unless you are using Details.
- **Tooltips** You may use one or more fields here.

For example, create a Treemap visual with Sales Territory as the group, State Province for the details, and Total Actual Amount as the values. The resulting chart is shown in Figure 2.76.

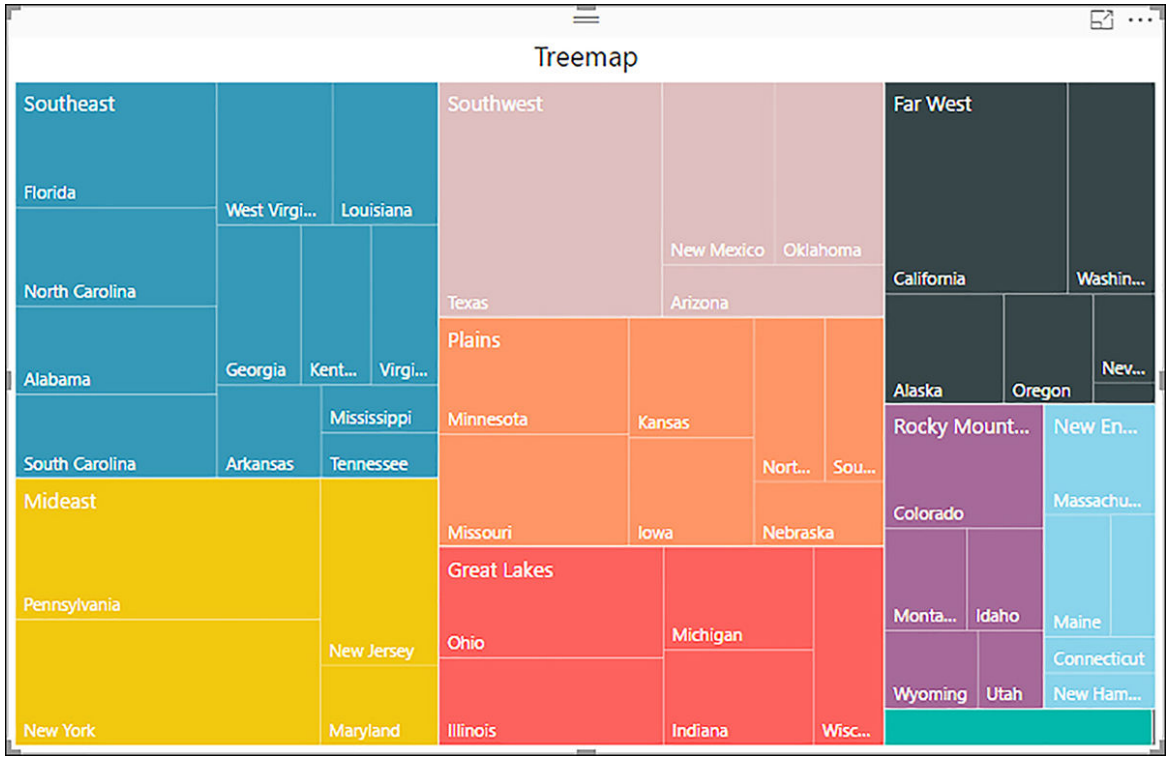


Figure 2.76 Treemap

In this case, the size of each rectangle corresponds to Total Actual amount. The rectangles are arranged top to bottom and left to right based on numerical values in descending order.

A treemap visual can be a good choice when you want to show proportions between each part and the whole, as well as highlight the most-important contributors and outliers.

MORE INFO **TREEMAP**

For a video tutorial and more information on the Treemap visual, including areas of application, see “Treemaps in Power BI (Tutorial)” at

https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-treemaps.

Maps in Power BI

Power BI has several options when it comes to visualizing geospatial data. In this section, we are going to review two map visuals: Map and Filled Map. You can use the following field wells:

- **Location** One or more categorical columns can be used.
- **Legend** You may use one categorical field here.
- **Latitude** You may use one field here.
- **Longitude** You may use one field here.
- **Size** For a numerical field (Map only).
- **Color saturation** You may use a numerical field here unless you use a Legend.
- **Tooltips** You may use one or more fields here.

When you use a Map visual, you plot bubbles on a map, with the size of each bubble corresponding to the value in the Size field well. If you are using a Legend, your bubbles will be turned into pie charts. If your Legend field well is empty, you can also use a numerical field for color saturation.

If you use geographical hierarchies in the Location field well, you will be able to drill down into specific areas on your map.

NOTE **IMPROVING ACCURACY OF MAPS**

Sometimes points on a map might be plotted inaccurately. To improve the accuracy of maps, you should categorize your data. Data categories are covered later in this chapter.

Figure 2.77 shows an example of a Map chart with State Province used as the location, Buying Group used as the legend, and Total Actual Amount used as the size.

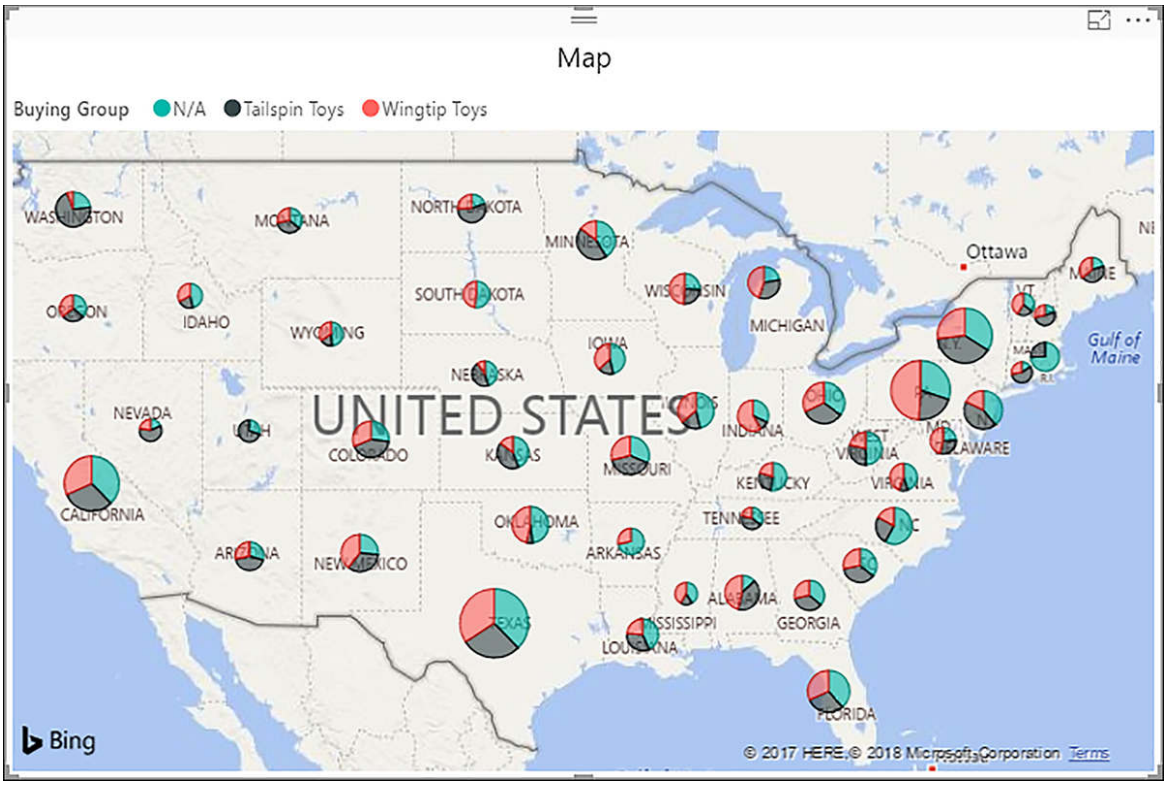


FIGURE 2.77 Map example

The map in Figure 2.77 was filtered to show only the contiguous U.S. states. One way to reproduce the result is to add the CONUS column to the Visual Level Filters field well, which is in the Filters pane underneath the visual field wells, and select 1.

> ***MORE INFO* USING MAPS**
>
> For a video overview and more information on how you can use the Map visual, including tips and tricks, see "Tips and Tricks for Power BI Map visualizations" at *https://docs.microsoft.com/en-us/power-bi/power-bi-map-tips-and-tricks*.

When you use the Filled Map visual, also known as a choropleth map, you are not able to use the Size field well. Instead, you can use Color Saturation to highlight the difference between areas.

> ***IMPORTANT LEGEND IN FILLED MAP***
>
> If you decide to use a legend, you need to be aware that color saturation will stop working. Furthermore, you need to make sure that each

location has exactly one corresponding value in legend—otherwise all items will appear with the same color, and the color saturation value in tooltips will reflect only the first legend item, which may confuse users. Either way, if you use a legend, placing a value in the Color Saturation field well and using it as a tooltip is the same.

An example of the Filled Map is shown in Figure 2.78.

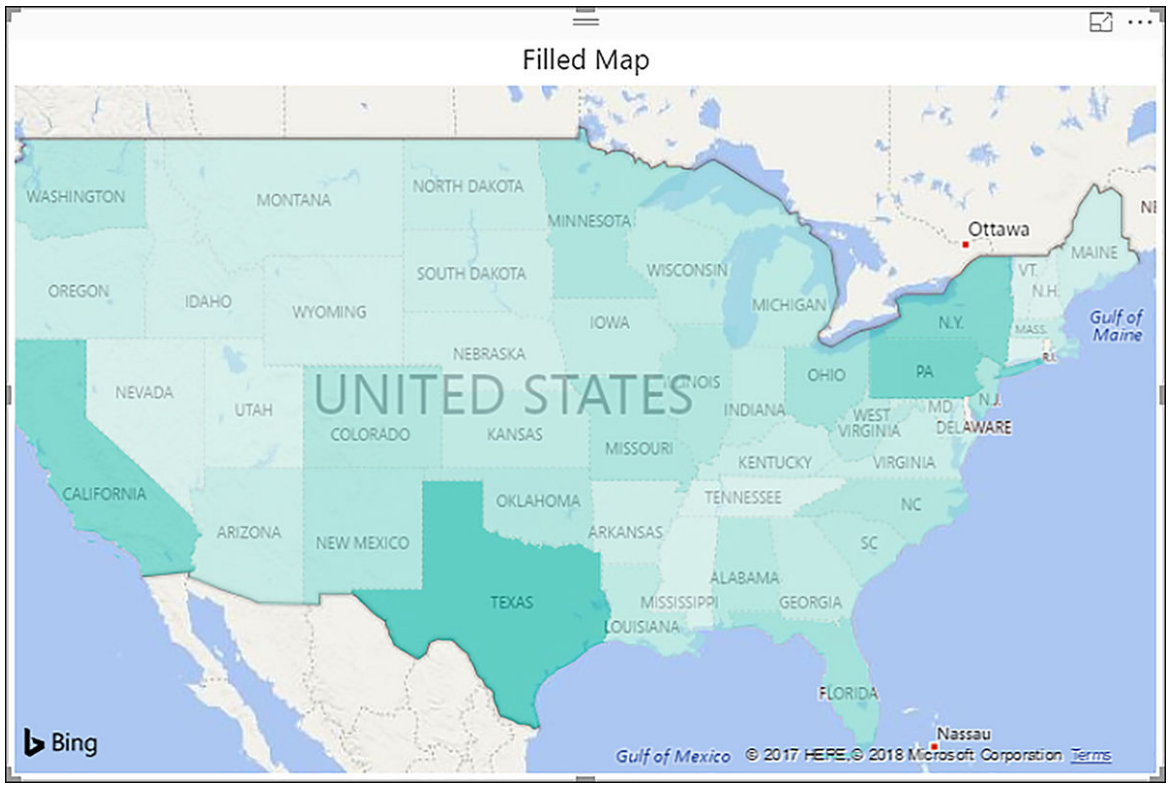


FIGURE 2.78 Filled Map

The map in Figure 2.78 uses Total Actual Amount for the color saturation and State Province as the location. This map only shows the contiguous U.S. states.

***MORE INFO* FILLED MAPS**

For more details on how you can use the Filled Map visual, see "Filled maps (choropleths) in Power BI (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-filled-maps-choropleths*.

***MORE INFO* OTHER MAPS**

There are many other options for visualizing geographical data in Power BI, including the ArcGIS Maps for Power BI and Shape Maps.

The ArcGIS Maps allow you to customize your maps beyond the standard map capabilities. For a tutorial on how you can create ArcGIS maps in Power BI, see "ArcGIS maps in Power BI service and Power BI Desktop by Esri" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-arcgis*.

For information on how you can interact with an ArcGIS map that has already been developed, see "Interacting with ArcGIS maps in Power BI" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualizations-arcgis*.

With Shape maps, you can use custom maps, which need not be geographical; for example, you can use a floor plan as a map. For more information on using Shape maps, see "Shape Maps in Power BI Desktop (Preview)" at *https://docs.microsoft.com/en-us/power-bi/desktop-shape-map*.

Custom visuals, which are covered later in this chapter, also allow you to visualize spatial data in many ways. For a comprehensive overview, see "10 Ways to Create Maps in Microsoft Power BI" at *https://dataveld.wordpress.com/2017/06/03/10-ways-to-create-maps-in-microsoft-power-bi/*.

Funnel

The final visual we are going to review is the Funnel chart, which looks similar to a bar chart with bars center-aligned. The visual has four field wells:

- **Group** You can use one or more categorical columns here.
- **Values** You can only use a numerical field here.
- **Color saturation** A numerical field may be used.
- **Tooltips** One or more fields can be used here.

An example of a funnel chart i shown in Figure 2.79.

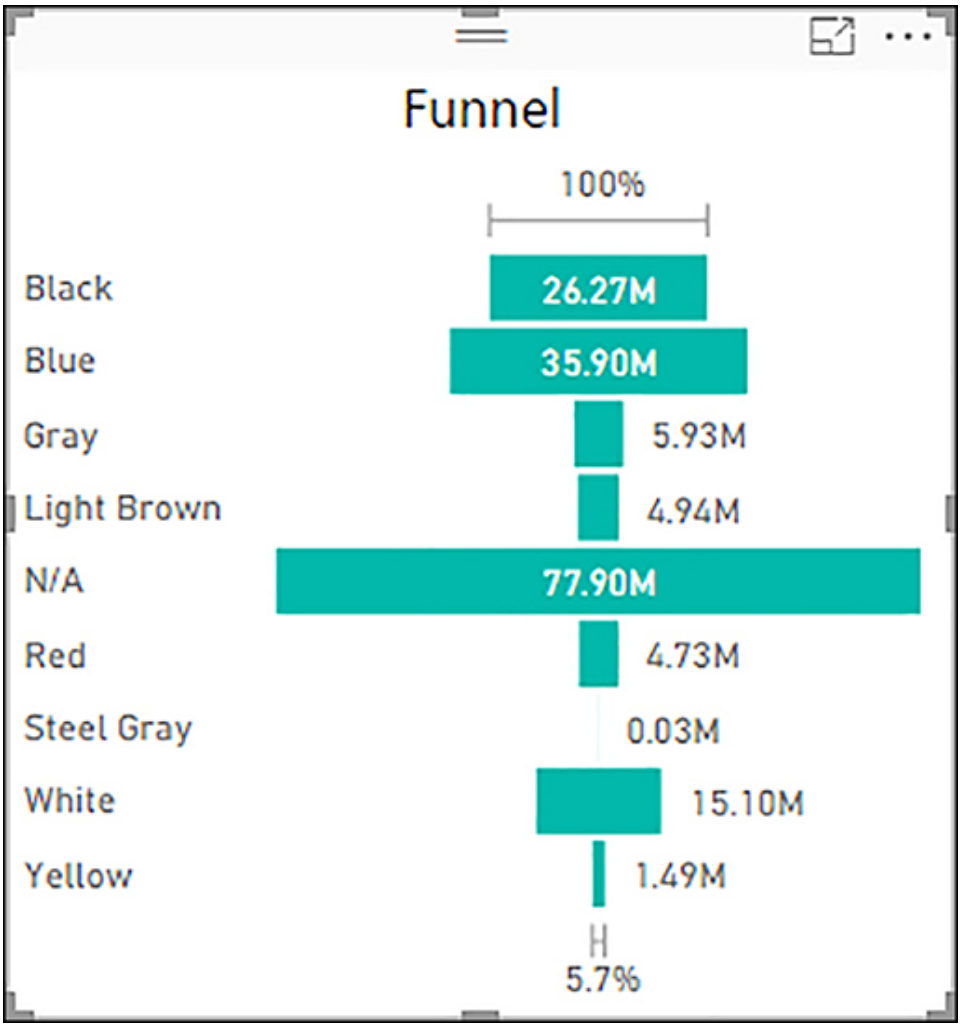


Figure 2.79 Funnel chart

Note that the Funnel displays the last item relative to the first item.

A funnel chart can be a proper choice for showing values by stages, for instance, by lead generation stages. The visual can also be used for revealing bottlenecks in a process or to track workflow.

> ***MORE INFO*** **FUNNEL CHART**
>
> For a video tutorial and more information on how you can create a funnel chart, including areas of application, see "Funnel charts (Tutorial)" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-funnel-charts*.

Configure page layout and formatting

By default, the size of each page in Power BI Desktop is 1280 by 720 pixels. You can modify this size by selecting the Format pane after making sure that no visual is selected. You will then see three sections:

- Page Information
- Page Size
- Page Background

In Page Information, you can give the page a new name and aliases, as well as choose whether Q&A is enabled for this page. You can also rename a page by either double-clicking on its tab or right-clicking on the tab and selecting Rename Page.

> ***MORE INFO* Q&A**
>
> Q&A is the natural language processing ability of Power BI, which can visually answer questions about your data. The Q&A feature is covered in Chapter 3.

In Page Size, you can select one of the preset page sizes, or specify a custom size in pixels. The preset sizes are as follows:

- 16:9 (1280 by 720 pixels)
- 4:3 (960 by 720 pixels)
- Cortana (296 by 592 pixels)
- Letter (816 by 1056 pixels)

In Page Background, you can choose the background color and specify transparency. Also, you can choose a background image and three options to scale it:

- **Normal** Displays the image in its original size.
- **Fit** Stretches the image to canvas size without keeping proportions.
- **Fill** Stretches the image to touch the canvas from inside, keeping proportions.

Besides page formatting, you can also adjust page view in **View** > **View** > **Page View**. You have three options:

- Fit to Page
- Fit to Width
- Actual Size

All three options keep the proportions of a page.

Configure interactions between visuals

One of the defining features of Power BI is the interaction of visuals with each other. For example, you can click on an item in a bar chart, and it will cross-highlight a column chart. This behavior is illustrated in Figure 2.80.

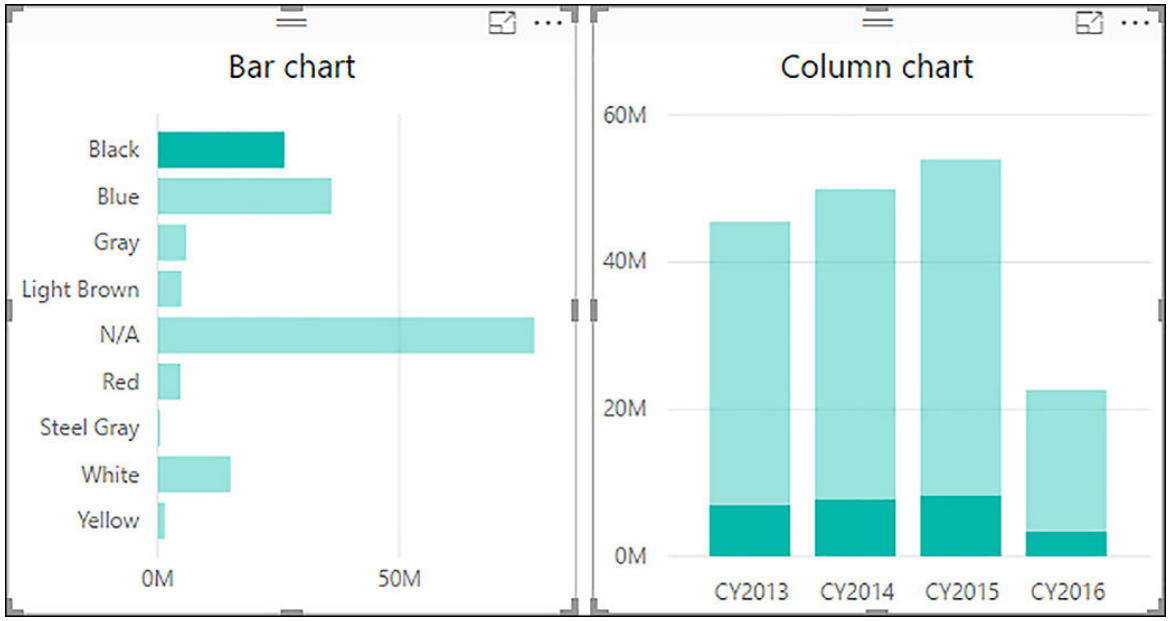


FIGURE 2.80 Cross-highlighting in action

In Figure 2.80, we clicked on the bar that displays Total Actual Amount for the Black color. The other bars in this graph then became dimmed, and portions of columns in the other graph became highlighted. These highlighted portions represent Total Actual Amount values in each calendar year that relate to the Black color. This behavior is called cross-highlighting: clicking on a visual highlights portions of other visuals.

To change this behavior, you can click on the bar chart, after which the Format tab in the top becomes visible. You can choose **Format** > **Interactions** > **Edit interactions**. You should then see new buttons appear in the top-right corner of the column chart, shown in Figure 2.81.

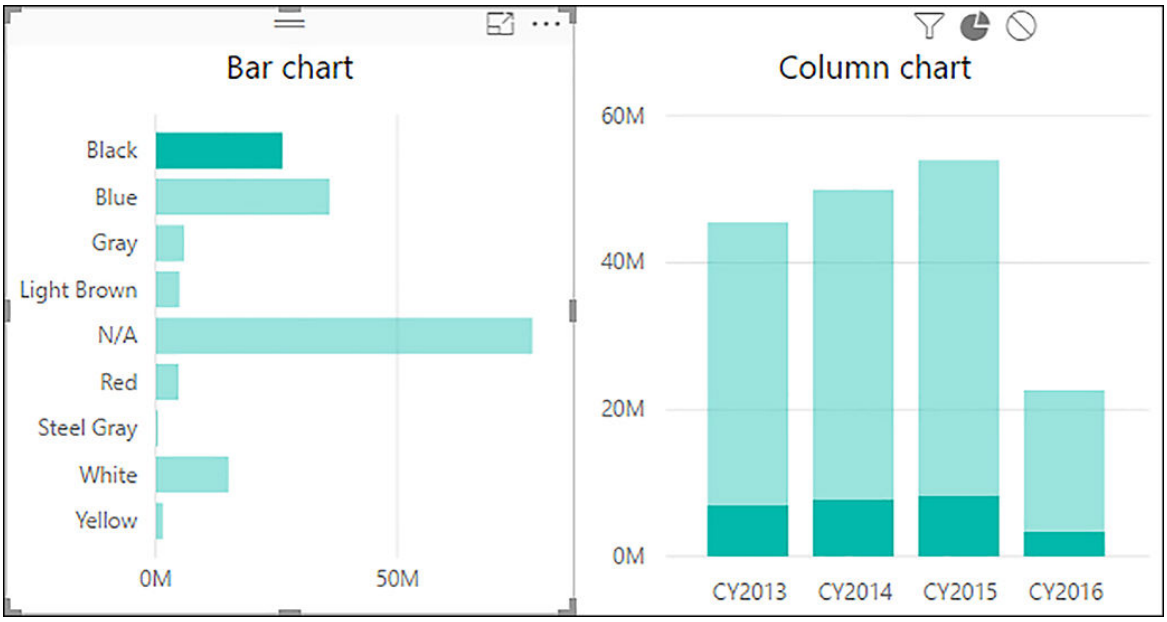


FIGURE 2.81 Interaction buttons

In this case, you see three buttons, left to right:

- Filter
- Highlight
- None

By default, the Highlight button is selected in Figure 2.81. In tooltips for the column chart, you will see both the total value and the highlighted one for each column.

If you select the Filter button instead, your visuals will look like in Figure 2.82.

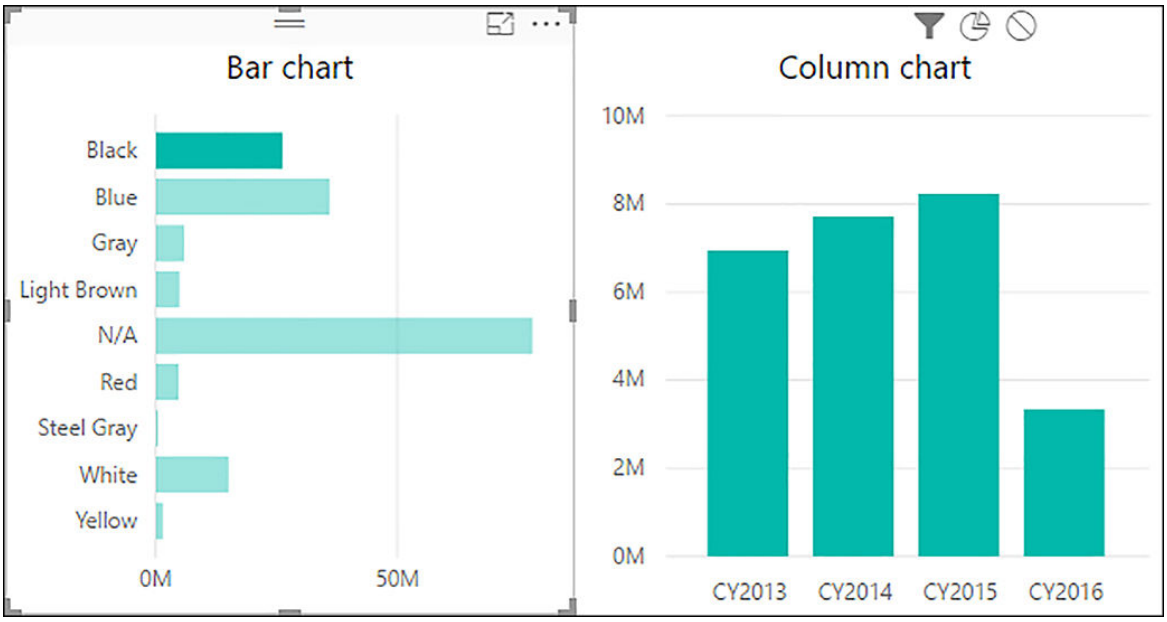


Figure 2.82 Cross-filtering behavior in action

Note how this time the columns are solid color; instead of highlighting, they are filtered to show different values.

> ***MORE INFO*** **FILTERING AND HIGHLIGHTING**
>
> For more information on the difference between filtering and highlighting, see "About filters and highlighting in Power BI reports" at *https://docs.microsoft.com/en-us/power-bi/power-bi-reports-filters-and-highlighting*.

The final option is the None button. If you select it, the column chart will ignore any selections made in the bar chart. You can employ the same technique to make a visual ignore other visuals, such as a slicer. You need to select how a visual affects others for each visual.

Not every chart has all three interaction options; some only have two, such as Filter and None. For example, it is not possible to highlight part of a slicer or a card. In these cases, the default behavior will be Filter instead of Highlight.

> ***MORE INFO*** **VISUALS INTERACTION**
>
> For a video overview and more details on how visuals can interact with each other, see "Visualization interactions in a Power BI report" at *https://docs.microsoft.com/en-us/power-bi/service-reports-visual-interactions*.

Configure duplicate pages

In some cases, you might want to show two similar pages with slight changes. For example, you might want to display several visuals that relate to revenue, and you would like another page to show costs with the same visuals. One way to accomplish this task is to duplicate a page and then switch the values. Doing so saves time by not having to rebuild and reformat all of the visuals.

You can duplicate a page by right-clicking on its tab in the bottom and selecting Duplicate Page. This will create a new page with the "Duplicate of" prefix. Alternatively, when you are on the page you want to copy, you can select **Home** > **Insert** > **New Page** > **Duplicate Page**.

Handle categories that have no data

By default, Power BI hides all categorical items that have blank corresponding measures, whether implicit or explicit. You can change this behavior by clicking on the down arrow of a categorical field in its field well and selecting Show items with no data. Now, you will see items even if they have no data. You can see the difference this makes in Figure 2.83.

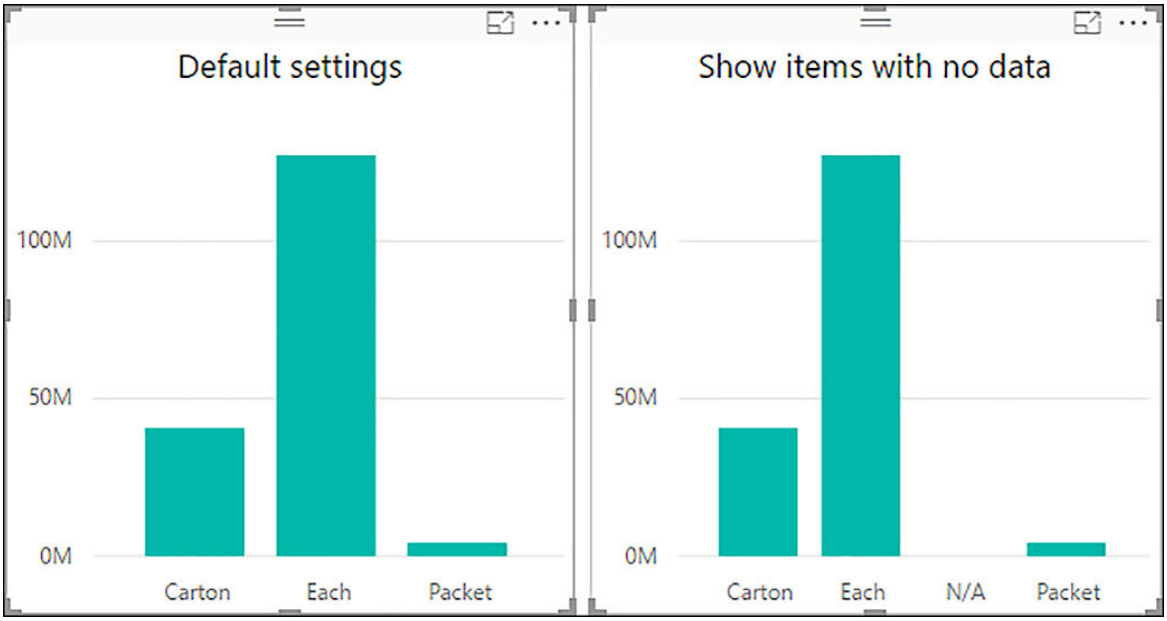


FIGURE 2.83 Showing items with no data

An alternative way to solve the same problem is to add a zero to a measure formula. For instance, you could create the following measure:

```
Always Show Amount = [Total Actual Amount] + 0
```

Adding a zero to measure formula shows 0 instead of a blank value when there is no data.

Configure default summarization and data category of columns

You can further enrich your data model by assigning certain properties to columns, which include default summarization and data category.

Default summarization

When you put a field into the Values field well, its values might need to be summarized. For example, you can put the Profit column from the Sale table into a card visual, and the latter will display the total profit in the current filter context. When Power BI sees a numerical column, it will assign a default summarization; you can see the default summarization if you select the field in the Fields pane and select **Modeling** > **Properties** > **Default Summarization**. In the case of Profit, Sum was selected by Power BI Desktop automatically. If necessary, you can change it to any of the following:

- Don’t summarize
- Sum
- Average
- Minimum
- Maximum
- Count
- Count (Distinct)

Not all default summarization options are available for every column. Columns of text and datetime type only have the following options:

- Don’t summarize
- Count
- Count (Distinct)

Columns of True/False data type can only be counted or not summarized.

The Don’t Summarize option makes Power BI treat a column like a categorical one. For example, if you select Don’t Summarize for the Tax Rate column from the Sale table visual and create a table with it, Power BI will show the distinct values of the column instead of its total sum. This behavior can be especially desirable for numeric columns that are not meant to be operated on, such as year number.

You can also override the default summarization by changing the summarization type in each field well. To do that, you need to right-click on a field in a field well or click on its down arrow in a field well and select the summarization type. For numeric columns, you can select the following summarization types:

- Don’t summarize
- Sum
- Average
- Minimum
- Maximum
- Count (Distinct)
- Count
- Standard deviation
- Variance
- Median

For datetime type columns, you can choose from the following:

- Don’t summarize
- Earliest
- Latest
- Count (Distinct)
- Count

Columns of type text can be summarized in the following ways:

- Don’t summarize
- First
- Last
- Count (Distinct)
- Count

The summarization types for the True/False type columns are the same as the defaults for that type.

Data categories

Besides default summarization, you can assign data categories to columns, which is also done in **Modeling** > **Properties**. Categorizing your data can help Power BI

with treating it properly in visuals. The following data categories are currently available:

- Address
- City
- Continent
- Country/Region
- County
- Latitude
- Longitude
- Place
- Postal Code
- State or Province
- Web URL
- Image URL
- Barcode

All but the last three data categories are related to geography; they might be helpful when your data is ambiguous. For instance, a two-letter code can stand for either a country code or a U.S. state code, which means categorization will improve mapping accuracy. Once you categorize a column as a geographical column, it will have a globe icon next to it.

The Web URL data category can be used to display clickable hyperlinks in a table or in a matrix visual.

> ***MORE INFO* WEB URLS**
>
> For more information on how you can display hyperlinks in some visuals, including displaying a hyperlink icon instead of text in tables, see "Hyperlinks in tables" at *https://docs.microsoft.com/en-us/power-bi/power-bi-hyperlinks-in-tables*.

With the Image URL data category, you can display images in some visuals, including Multi-Row Card, Slicer, Table, and Matrix.

The Barcode data type allows you to scan an item with the iPhone app and filter your report for this item.

***MORE INFO* TAGGING BARCODES**

For more information on working with barcodes in Power BI, see "Tag barcodes in Power BI Desktop for the mobile apps" at *https://docs.microsoft.com/en-us/power-bi/desktop-mobile-barcodes*.

***MORE INFO* DATA CATEGORIES**

To read more about how you can categorize data in Power BI Desktop, see "Data categorization in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-data-categorization*.

Position, align, and sort visuals

You can alter the way your visuals are presented by moving and sorting them.

Positioning visuals

There are three ways in which you can position your visuals:

- First, you can drag a visual around with your mouse.
- Second, you can go to the Format tab and specify new X and Y coordinates in the General section.
- Third, you can select a visual and then use the arrow keys on your keyboard to position the visual.

If you want to disable the ability of visuals to be moved, you can select **View** > **Show** > **Lock Objects**. You can also focus on a single visual by clicking the Focus Mode icon in its top-right corner, next to the ellipsis.

Aligning visuals

To make it easier to align the visuals, you can click **View** > **Show** > **Snap Objects to Grid**. This will increase the grain in which you can move your visuals. To see gridlines, you can choose **View** > **Show** > **Show Gridlines**.

Power BI also has several options for alignment of visuals by selecting **Visual tools Format** > **Arrange** > **Align**:

- Align left
- Align center
- Align right
- Align top

- Align middle
- Align bottom

Also, you can distribute visuals horizontally or vertically by selecting **Visual tools Format** > **Arrange** > **Distribute**. For this function to work properly, you need to select at least three visuals.

Sorting visuals

You can sort many Power BI visuals either alphabetically by the names of categorical items, or in ascending or descending order by values.

For instance, if you have a bar chart that shows Total Actual Amount by Color, you can sort the colors in descending order by Total Actual Amount. To do that, you would need to click on the ellipsis in the top-right corner and select Sort by Total Actual Amount. You can see the menu in Figure 2.84.

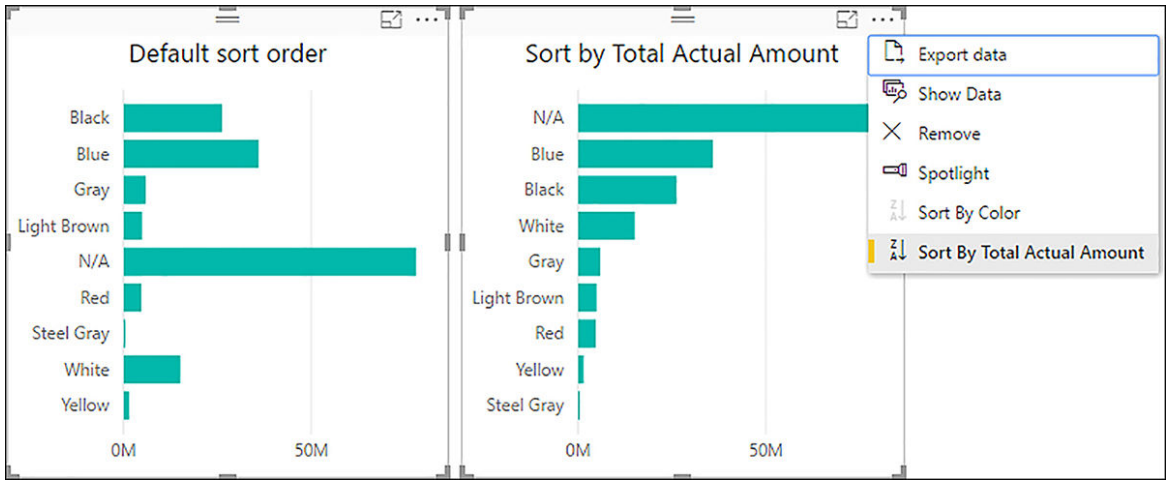


FIGURE 2.84 Sorting visuals

If you need to sort in ascending order instead of descending one, you can click on the ZA icon next to Sort by Total Actual Amount.

> ***MORE INFO*** **SORTING VISUALS**
>
> For more information on how you can sort visuals, including sorting using other criteria, see "Change how a chart is sorted in a Power BI report" at *https://docs.microsoft.com/en-us/power-bi/power-bi-report-change-sort*.

Note that from the same menu, you can export data from the visual in a CSV file by clicking Export Data, or you can look at the data behind the visual by

clicking Show Data. You can also delete the visual by clicking Remove.

Enable and integrate R visuals

In Power BI Desktop, it is possible to visualize data with R, though you need to have the R engine installed on your computer first.

> ***NOTE* DOWNLOADING R**
>
> You can download the R engine from several websites, including CRAN at: *https://cran.r-project.org/bin/windows/base/*, and MRAN at *https://mran.revolutionanalytics.com/download*.

Once you have R installed, you can go to **File** > **Options and settings** > **Options, then Global** > **R scripting**. You need to make sure the correct R home directory is specified, and you must select the preferred R IDE (Integrated Development Environment). You can then create visuals powered by R.

When you click on the R visual icon for the first time, you will see the Enable script visuals dialog box, shown in Figure 2.85.

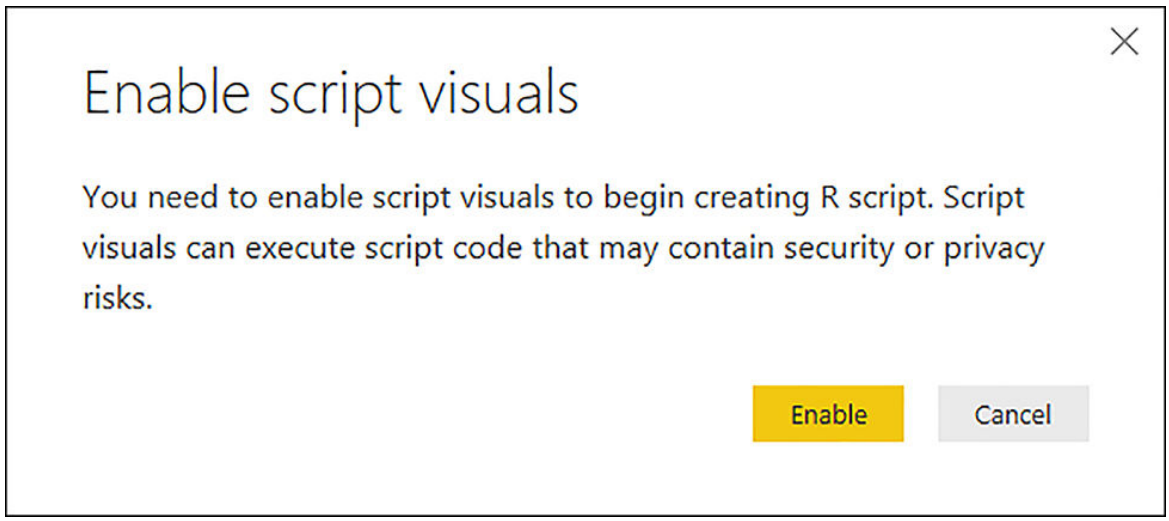


FIGURE 2.85 Enable script visuals dialog box

Clicking Enable creates a placeholder R visual and opens the R script editor, which you can see in Figure 2.86.

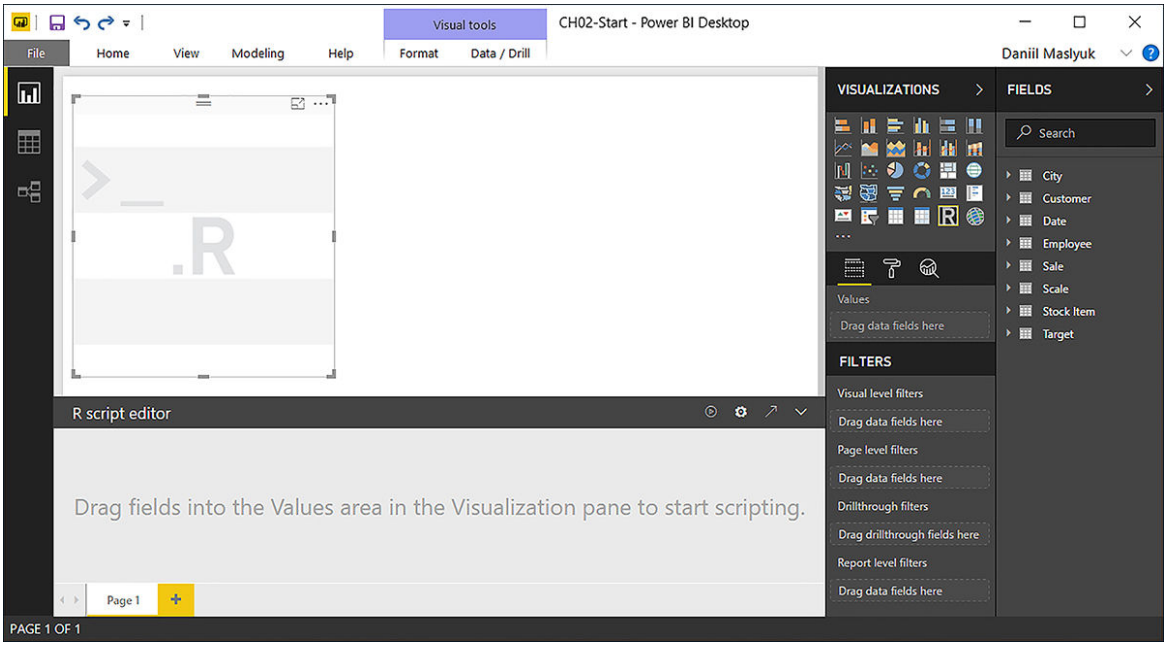


Figure 2.86 R script editor

For example, use the Color column from the Stock Item table and the Total Actual Amount measure with the following R script to produce a bar graph that shows Total Actual Amount, in millions, by colors. The result is shown in Figure 2.87.

LISTING 2-16 R script

Click here to view code image

```
Colors <- dataset$Color
Amounts <- dataset$'Total Actual Amount' / 1000000
par ( mar = c ( 2, 6, 2, 2 ) + 0.1 )
barplot ( Amounts, horiz = TRUE, names.arg = Colors, las = 1 )
```

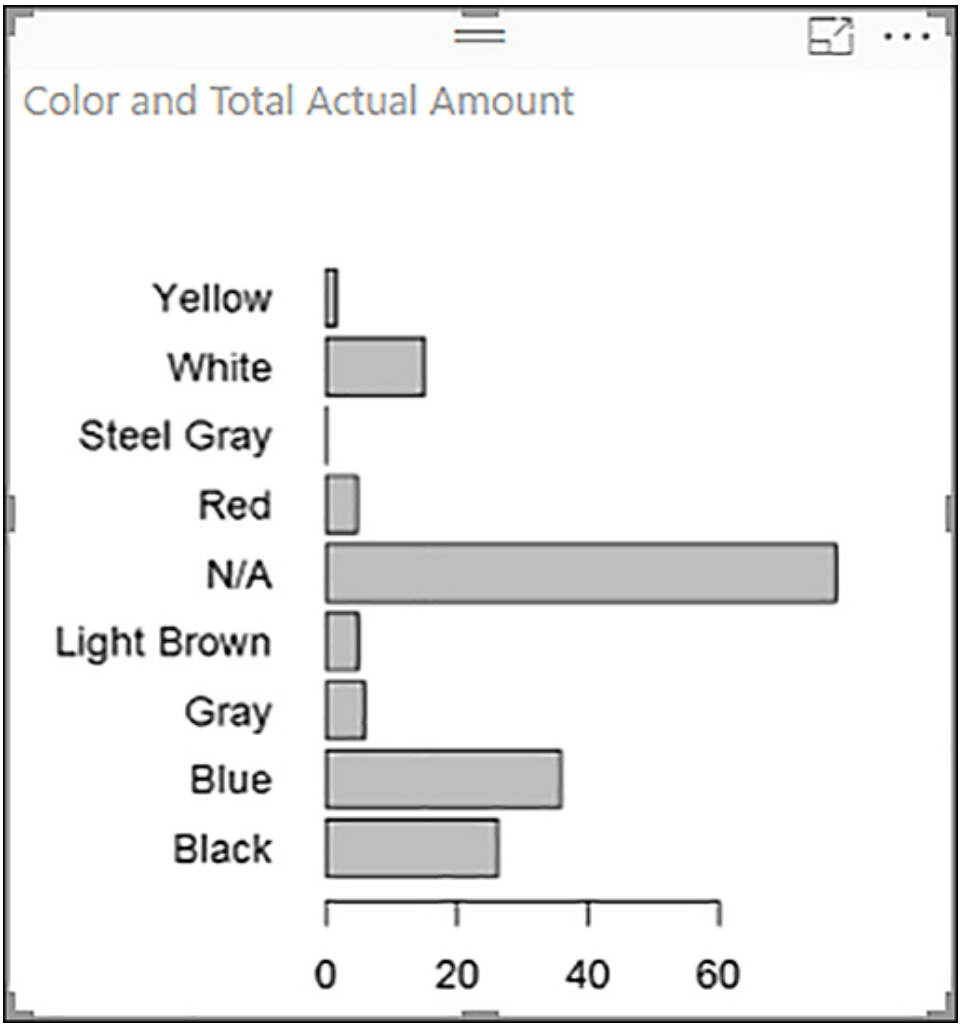


FIGURE 2.87 R visual

MORE INFO **USING R IN POWER BI DESKTOP**

R can be a powerful addition to the standard visuals in Power BI. For more information on using R to create visuals in Power BI Desktop, including limitations, see "Create Power BI visuals using R" at *https://docs.microsoft.com/en-us/power-bi/desktop-r-visuals*.

You can use any R library that is installed locally on your machine when you work in Power BI Desktop. If you decide to publish or share your report, which is covered in Chapter 3 under "Publish and embed reports," you should be aware of the limitations—namely, the available libraries. For more information, see "Creating R visuals in the Power BI service" at *https://docs.microsoft.com/en-us/power-bi/service-r-visuals*.

In addition to creating visuals, R can also be used as a data source in Get Data. For more details, see "Run R scripts in Power BI Desktop" at

https://docs.microsoft.com/en-us/power-bi/desktop-r-scripts.

Format measures

In Power BI Desktop, you can format both columns and measures, and the process is identical. You can select formatting in **Modeling** > **Formatting** > **Format**. Depending on the data type that a measure returns, you might see some of the following options available when choosing formatting:

- General
- Currency
- Date Time
- Decimal Number
- Whole Number
- Percentage
- Scientific
- Text
- Binary
- True/False

Both Currency and Date Time have additional options available. Additionally, you can display or hide the thousands separator and set the number of decimal places.

For example, you can format the Total Actual Amount measure as Currency General by selecting the measure in the Fields pane, then selecting **Modeling** > **Formatting** > **Format** > **Currency** > **Currency general**. To set a fixed number of decimal places, specify a number instead of Auto, such as 0. This can be done by either clicking on the arrows or typing. As a result, the measure will display values with a dollar sign prefixed, with thousands separator and no decimal places.

Use bookmarks and themes for reports

By using bookmarks in Power BI, you can create compelling data stories. Bookmarks allow you to save filters and the state of visuals; bookmarks can be arranged in order, and you can step through them in a presentation to highlight insights. You can spend less time on formatting by using report themes, which override the default formatting of visuals.

Bookmarks

Though one of the defining features of Power BI is its interactivity, in some cases you may prefer to draw users' attention to specific aspects of your reports. For example, you might want to highlight a certain visual with applied filters that show an interesting insight. Bookmarks in Power BI allow you to do exactly that: they let you save the states of your pages and switch between them in presentation mode or by linking to them.

Bookmarks can save states of pages, which corresponds to the following:

- Filters, including cross-filtering and selections in slicers
- Current page
- Objects visibility
- Sort order
- Drill down level

To create a bookmark, you need to open the Bookmarks pane first: click **View** > **Show** > **Bookmarks Pane**. Once you apply filters or otherwise achieve the desired state of the page you are on, click **Add** in the Bookmarks pane. The bookmark will then appear in the Bookmarks pane with a default name of **Bookmark 1**. You can rename a bookmark by double-clicking on its name or by clicking on the ellipsis next to it and selecting **Rename**. The ellipsis menu also contains the following items:

- **Update** If you decide to change the state of a bookmark, you can update an existing bookmark instead of creating a new one.
- **Delete** This option deletes the bookmark.
- **Data** This option determines whether the filter context of the page is saved in the bookmark. By default, this option is checked. Unchecking it can be useful when you only want to save the visibility of visuals, which is covered below in more detail.
- **Display** This option, which is checked by default, determines whether the bookmark saves the visibility of visuals. This setting can be useful when you only want to change filters with your bookmark without changing the visibility of visuals.
- **Current page** By default, this option is checked; it determines whether clicking the bookmark takes you to the page where it was created. Unchecking the option can be useful when you want to change a report-level filter without going to a different page.

- **All visuals** This option determines whether the bookmark is applied to all visuals or selected visuals only. By default, a bookmark is applied to all visuals.
- **Selected visuals** You can create a bookmark and only apply it to the visuals, which were selected at the time you created the bookmark. It is important to note that you cannot create a bookmark without selecting visuals, then select some visuals, click Selected visuals and then Update – this change will have no effect.

When creating bookmarks, you can not only apply filters but also change the visibility of visuals; the latter is done in the Selection pane. To open the pane, click **View** > **Show** > **Selection Pane**. In the pane, you will see a list of your visuals, which will all be visible by default. Note that the names of the visuals correspond to their titles, even if they are hidden. If a visual has no title, the visual will have the same name as the chart type. As a result, you may see multiple visuals with the same name. In this case, you can give your visuals meaningful names by adding user-friendly titles and then hiding them. To do this, select a visual either by clicking on it in the report canvas or by selecting it in the Selection pane, then click **Format** > **Title** > **Off** (this will turn it to On), enter **Title Text** and click **On** to hide the title.

The Selection pane allows you to hide visuals by clicking on the eye icon next to a visual. Note that hiding a visual does not delete it; instead, the visual becomes invisible, but it is still part of the page. One of the implications of this is that hidden slicers can still filter other visuals.

In a bookmark, you can highlight a single visual by clicking on the ellipsis in its top-right corner and selecting **Spotlight**. Once you do this, the visual will be highlighted with a shade around it, and the rest of the report canvas will be tinted. You can see this effect in Figure 2.88. At this stage, we can create a bookmark by clicking Add in the Bookmarks pane. If you created a bookmark before, the new bookmark would be called Bookmark 2.

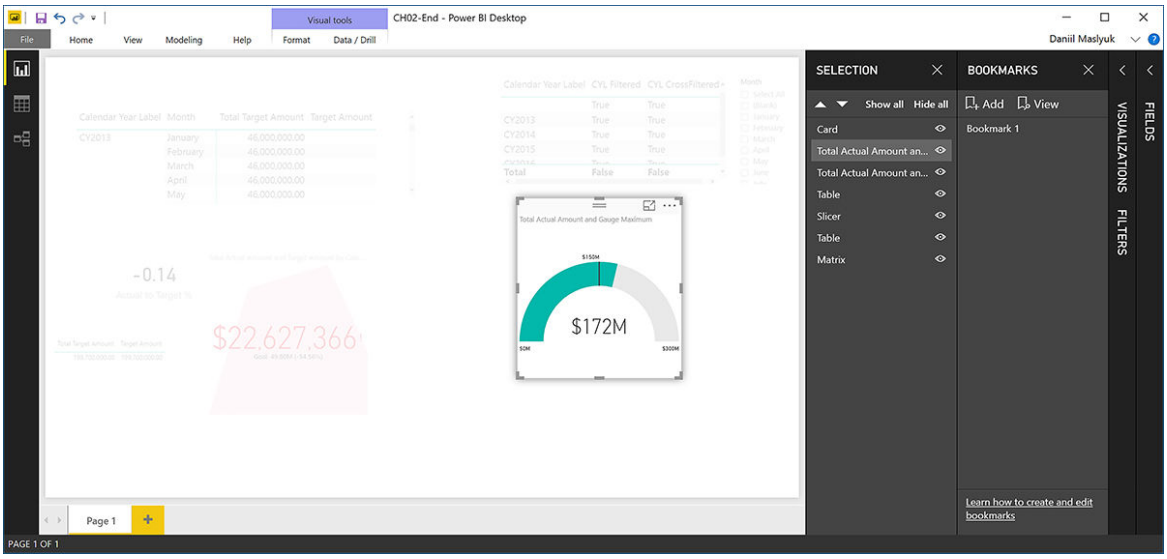


FIGURE 2.88 Spotlight

To undo the Spotlight effect, you can click anywhere else on the report canvas or select another visual in the Selection pane.

To review the effect of hiding visuals, you can hide some visuals by clicking the eye icon next to the visuals in the Selection pane. Once you do this, create another bookmark. The eye icons of the hidden visuals will be replaced with dashes. Figure 2.89 shows all but two visuals hidden.

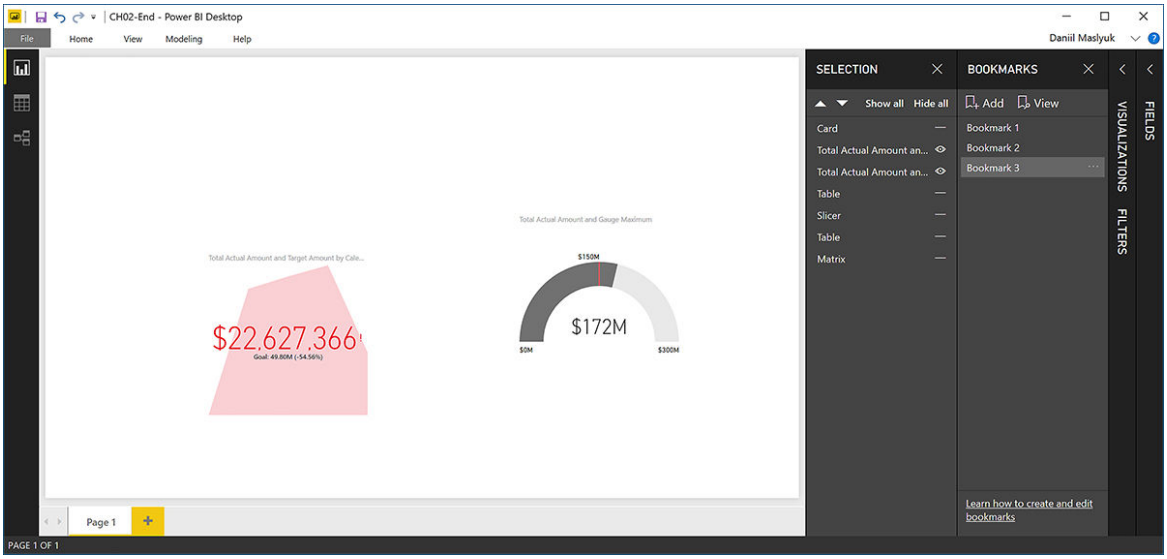


FIGURE 2.89 Hidden visuals

Note that in the Selection pane, you can hide or show all visuals at once by clicking the Hide All or Show All button, respectively. You can change the z-order, also known as z-index, of visuals by clicking the up or down arrows at the

top of the pane. The z-order of a visual determines which visual gets selected when you click on an area where two visuals overlap. To switch between different visuals, you can press the Tab button on your keyboard after highlighting a visual.

> ***MORE INFO*** **ACCESSIBILITY AND KEYBOARD SHORTCUTS**
>
> You can move to the previously selected visual by pressing Shift + Tab. To find out about the other accessibility features in Power BI, including keyboard shortcuts, see "Accessibility in Power BI Desktop reports" at *https://docs.microsoft.com/en-us/power-bi/desktop-accessibility*.

When you have created some bookmarks, you can change their order by dragging and dropping them in the Bookmarks pane. You can navigate between bookmarks by clicking them in the Bookmarks pane, or you can click the View button, which will replace the pages bar at the bottom with the bookmark title bar. When you are in the View mode, also known as presentation mode, you can close the Bookmarks and Selection panes to have more room for your report; the panes can be closed by clicking the cross in the top-right corner in each. The Visualizations and Filters panes can be collapsed by clicking on their names, and the Ribbons pane can be collapsed by double-clicking on any tab or by clicking on the arrow in the top-right corner of the pane. Figure 2.90 shows the bookmark title bar in the View mode.

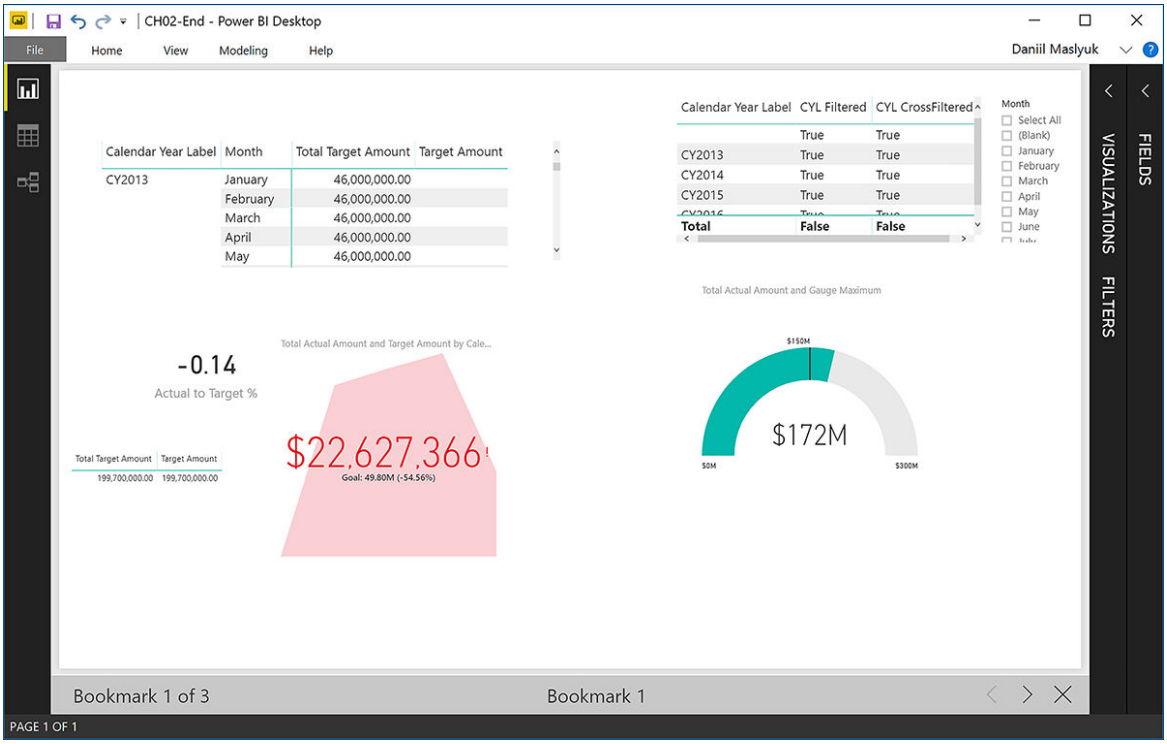


FIGURE 2.90 Hidden visuals

You can exit from the View mode by clicking Exit in the Bookmarks pane, or by clicking the cross in the bookmark title bar.

To navigate to a specific bookmark, you can also link a shape or an image to a bookmark. To create a shape, click **Home** > **Insert** > **Shapes** and select the shape you want to create. You can choose one of the following:

- Rectangle
- Oval
- Line
- Triangle
- Arrow

You can also link an image to a bookmark. To insert an image, click **Home** > **Insert** > **Image** and select an image from your computer. Once you have a shape or an image you want to turn into a link, select the visual. Note that the Visualizations pane will be replaced by the Format Shape pane. Click **Off** in the **Link** section of the Format Shape pane; once Off turns into On, click the drop-down list and select **Bookmark**. This will open the Bookmark drop-down list, where you can choose a Bookmark you want to link to. For example, you can pick **Bookmark 2**. At this stage, if you hold the Ctrl key on your keyboard and click the shape or image, you will be taken to Bookmark 2. You can create more

links to other bookmarks for more interactive navigation using the same technique. Note that when you are in Reading view in Power BI service, you do not need to hold the Ctrl key when you click on linked shapes or images.

> ***MORE INFO*** **USING BOOKMARKS**
>
> For more information on how you can use bookmarks in Power BI, see "Use bookmarks to share insights and build stories in Power BI" at *https://docs.microsoft.com/en-us/power-bi/desktop-bookmarks*.

Report themes

As discussed earlier in this chapter, Power BI allows you to apply custom formatting to your visuals. For example, you can change line colors in a line chart or change font size of slicer items. If you set the same formatting options for different charts repeatedly, you can benefit from using report themes, which allow you to set your own default format settings for each formatting option. In addition to this, report themes can contain a custom color palette.

> ***NOTE*** **ENABLING REPORT THEMES**
>
> Depending on the version of Power BI Desktop you are using, you may need to enable custom report themes first. For this, click File > Options and settings > Options > Global > Preview features > Custom report themes > OK.

Report themes are created as JSON files and then imported in Power BI Desktop. Listing 2-17 shows an example of a report theme that sets new default colors and a new default text size and color for all visuals. The code is available in the companion files folder as ReportTheme.json.

LISTING 2-17 Report theme JSON

Click here to view code image

```
{
    "name": "ER778",
    "dataColors": ["#717171", "#F1595E", "#79C369", "#589AD3",
"#F9A659", "#D77FB2", "#CE7058", "#9E67AB"],
    "visualStyles": {
        "*": {
            "*": {
```

```
                "*": [{
                    "fontSize": 10,
                    "color": {
                        "solid": {
                            "color": "#000000"
                        }
                    }
                }]
            }
        }
    }
}
```

> ***NOTE*** **CREATING REPORT THEMES**
>
> While you can create a report theme file manually by typing the code and saving it as JSON, there are tools that facilitate the creation of report themes. For example, see "Theme Generator" at *https://powerbi.tips/tools/*.

To import a theme into your report, click **Home** > **Themes** > **Switch Theme** > **Import Theme**. After selecting your Power BI report theme file, click **Open**. If the code of the theme is correct, you will then see a message saying the theme was imported successfully; otherwise, you will get an error message. To remove a theme from a report, click **Home** > **Themes** > **Switch Theme** > **Default Theme**.

It is important to note that themes do not override any formatting that you have already set in the Format pane. For instance, if your line chart has one line and you specified your own color for it, the theme will not change the color. If you kept the default color of the line, the report theme will change the color to a new color; more precisely, the line color will be set to the first color you specified in the dataColors array in your report theme JSON file.

As you will see in Chapter 3 under "Configure a dashboard," the formatting from report themes can also be applied to dashboard tiles.

> ***MORE INFO*** **USING REPORT THEMES**
>
> For a general overview of report themes and details on which formatting options you can specify, see "Use Report Themes in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-report-themes*.

Skill 2.6: Manage custom reporting solutions

While Power BI already has powerful functionality out of the box, it can be further extended with custom reporting solutions. You can embed Power BI reports in custom applications and apply your own formatting to them, including branding. Furthermore, you can push data into Power BI datasets programmatically by using the Power BI REST API and create custom visuals to widen the available visualization options.

This section covers how to:

- Configure and access Microsoft Power BI Embedded
- Enable developers to create and edit reports through custom applications
- Enable developers to embed reports in applications
- Use the Power BI API to push data into a Power BI dataset
- Enable developers to create custom visuals

Configure and access Microsoft Power BI Embedded

To use Power BI Embedded, you need to have a dedicated capacity, which you can configure in Azure portal. The following is required for creating dedicated capacity:

- Azure subscription
- Azure Active Directory tenant
- Power BI account in the same Azure AD tenant

To create Power BI Embedded capacity in Azure portal, you need to complete the following steps:

1. Sign in to Azure portal at *portal.azure.com*.
2. Select New and search for Power BI Embedded.
3. Select Power BI Embedded and click Create.
4. Specify the following:
    - Resource name, which is the name of your Power BI capacity, and it will be used to identify your capacity in both the Azure portal and the Power BI admin portal.
    - Subscription in which to create the capacity.
    - Resource group of your capacity.
    - Power BI capacity administrator.

- Location Is the Azure region in which the capacity will be hosted.
- Pricing tier.

5. Click Create.

The capacity usually takes only a few seconds to create. Once it is created, you can assign the capacity to a workspace by selecting **Settings** > **Admin portal** > **Capacity settings** > **Power BI Embedded**, then click on the capacity name and choose how you want to allocate the capacity. Alternatively, you can go to settings of a workspace and enable Dedicated Capacity in Advanced Settings, then choose the capacity.

> ***MORE INFO*** **MANAGING CAPACITIES**
>
> For more details on how you can manage and assign capacities in Power BI, see "Manage capacities within Power BI Premium and Power BI Embedded" at: *https://docs.microsoft.com/en-us/power-bi/service-admin-premium-manage*.

If you don't need to use the capacity, pause it to stop incurring charges for it; go to the capacity's Azure portal settings to pause it.

> ***MORE INFO*** **CREATING POWER BI EMBEDDED CAPACITY**
>
> For a detailed walkthrough on how to create dedicated Power BI Embedded capacity, see "Create Power BI Embedded capacity in the Azure portal" at: *https://docs.microsoft.com/en-us/azure/power-bi-embedded/create-capacity*.

Enable developers to create and edit reports through custom applications

When you embed your reports in custom applications, developers can specify custom layouts. Custom layouts allow you to set your own report page sizes or visuals size and position, as well as its visibility. By default, embedded reports inherit the layout from Power BI Desktop, which you can override dynamically to better fit your application.

To create custom layouts in a report, you need to have a published report that is ready to be embedded, which is covered next.

***MORE INFO* CUSTOM LAYOUTS**

For more details on how you can specify custom layouts in embedded reports, see "Custom layouts" at *https://docs.microsoft.com/en-us/power-bi/developer/embedded-custom-layout*.

Enable developers to embed reports in applications

With Power BI Embedded, you can embed Power BI dashboards, reports, tiles, and visuals in custom applications. There are two main scenarios, and Power BI uses a single set of APIs for both:

- You can embed for users outside of your organization who do not have Power BI licenses.
- You can embed for users with Power BI licenses within your organization.

In the first scenario, end users interact with your app like any other app; the end user does not need to know it is Power BI. End users log in with the credentials you provide to them. In the second scenario, users need to have Power BI licenses allocated to them. A use case for this scenario is embedding in SharePoint Online or Microsoft Teams. This section covers embedding for users outside of your organization. The other scenario is covered in Chapter 3.

***MORE INFO* EMBEDDING WITH POWER BI**

For an overview of the ways in which you can embed your content with Power BI Embedded, see "Embedding with Power BI" at *https://docs.microsoft.com/en-us/power-bi/developer/embedding*.

To set up your Power BI Embedded environment, you need to make sure you have an Azure Active Directory (AD) tenant, a Power BI Pro account, and an Azure AD app with the necessary permissions.

***MORE INFO* GETTING STARTED WITH POWER BI EMBEDDED**

For more details on what is required for embedding Power BI content, see "Get started with Microsoft Power BI Embedded" at *https://docs.microsoft.com/en-us/azure/power-bi-embedded/get-started*.

Azure Active Directory tenant

To embed items from Power BI into your application, you need to have an Azure Active Directory tenant with at least one Power BI Pro user. You can either use your existing Azure AD tenant, create a new one just for embedding, or create a tenant for each of your customers.

> ***NOTE*** **CREATING AZURE AD TENANTS**
>
> This section assumes you already have an Azure AD tenant you can use. If you need to create an Azure Active Directory tenant, see "Create an Azure Active Directory tenant to use with Power BI" at *https://docs.microsoft.com/en-us/power-bi/developer/create-an-azure-active-directory-tenant*.

Power BI Pro user

Your Azure AD tenant needs to have a user that has a Power BI Pro license allocated to it. This account will be used by your application to generate embed tokens for integration of Power BI content into your application.

While you only need one such account, it is recommended to segregate permissions for security purposes. For instance, you may choose to create separate accounts for analysts responsible for creating Power BI content, and you may want to separate the application account from Azure AD tenant admin user. Both the analyst accounts and the app master user will need Power BI Pro licenses allocated to it; the organization/tenant admin user does not need a Power BI Pro license.

Azure AD app

Once you ensure you have an Azure AD tenant with a Power BI Pro user, you can register your app in Azure for embedding. There are two ways to register an app:

- Power BI App Registration Tool
- Azure portal

The first option requires only a few fields to fill, so it is quicker. If you need to make changes to your app, you will need to use Azure portal. To register your app with Power BI App Registration tool, you need to complete the following steps:

1. Visit *dev.powerbi.com/apps*.
2. Select "Sign in with your existing account."
3. Specify App Name.
4. Choose App Type:

- Server-side Web app is for Web APIs or web apps
- Native app is for an app that runs on a client device or when you embed content for your customers

5. Specify the Redirect URL and, if you are creating a server-side web app, the Home Page URL.
6. Choose the APIs for your app.
7. Select Register App. You will then be given a Client ID and, in case your app is a server-side web app, a Client Secret. You can retrieve a Client ID in Azure portal; if you lose the Client Secret, you will need to create a new one through Azure portal.

> ***MORE INFO*** **REGISTERING AN APP AND PERMISSIONS**
>
> For a detailed overview of the available Power BI permissions for apps, see "Power BI permissions" at *https://docs.microsoft.com/en-us/power-bi/developer/power-bi-permissions*.
>
> This section only covers app registrations with the Power BI App Registration tool. For more information on the process, including registration through Azure portal, see "Register an Azure AD app to embed Power BI content" at *https://docs.microsoft.com/en-us/power-bi/developer/register-app*.

App workspace

When you embed content for your clients, you need to publish your content into an app workspace. The app master user needs to be an admin of the app workspace. Creating an app workspace is going to be covered in Skill 3.5: "Configure apps and apps workspaces."

> ***MORE INFO*** **EMBEDDING POWER BI ITEMS**
>
> The exact steps on how to embed Power BI items in your apps are outside of scope of this book. For a detailed overview of how to embed Power BI content, see "Embed your Power BI dashboards, reports and tiles" at *https://docs.microsoft.com/en-us/power-bi/developer/embedding-content*.
>
> For a tutorial on Power BI Embedded, see "Embed a Power BI dashboard, tile, or report into your application" at *https://docs.microsoft.com/en-us/power-bi/developer/embed-sample-for-customers*.

Use the Power BI API to push data into a Power BI dataset

You can push data into a Power BI dataset using Power BI REST API. For that, you need to have a native app registered with the following redirect URL:

`https://login.live.com/oauth20_desktop.srf`

The only API access you need to grant to the app is Read and Write All Datasets. Once you click Register App, you will be given a Client ID, which you should save for later.

> ***MORE INFO*** **REGISTERING AN APP FOR PUSHING DATA**
>
> For a detailed walkthrough on how to register an app for pushing data, see "Step 1: Register an app with Azure AD" at *https://docs.microsoft.com/en-us/power-bi/developer/walkthrough-push-data-register-app-with-azure-ad*.

In the following example, you are going to create a dataset using Power BI REST API and push data into it using the same API. If you want to follow the example, you need to have Visual Studio 2017 installed and complete these steps:

1. In Visual Studio, select **File** > **New** > **Project** > **Console App (.NET Framework)**.
2. Give your project a name and click OK.
3. Install the Azure AD Authentication Library for .NET NuGet package. For that, click **Tools** > **NuGet Package Manager** > **Package Manager Console** and type the following, and then press Enter:

   ```
   Install-Package
   Microsoft.IdentityModel.Clients.ActiveDirectory -
   Version 2.21.301221612
   ```

4. Using the same procedure as in the previous step, install the Newtonsoft.Json NuGet package by typing the following in Package Manager Console, and then press Enter:

   ```
   Install-Package Newtonsoft.Json
   ```

5. Replace all Program.cs code with the code from PushData.cs from the companion files of this book.
6. Make sure to replace {Client_ID} on line 32 with your actual Client ID that you acquired after registering your app.

7. Click Start or press F5 to run your app.
8. Sign in to your Power BI account.
9. At this stage, you will see your application window open, which will display the authentication access token.
10. After you press Enter, the window will display “Dataset Created.”
11. Pressing Enter for the second time will show the dataset ID.
12. When you press Enter for the third time, you will see “Rows Added.” You can see the application window in Figure 2.91.

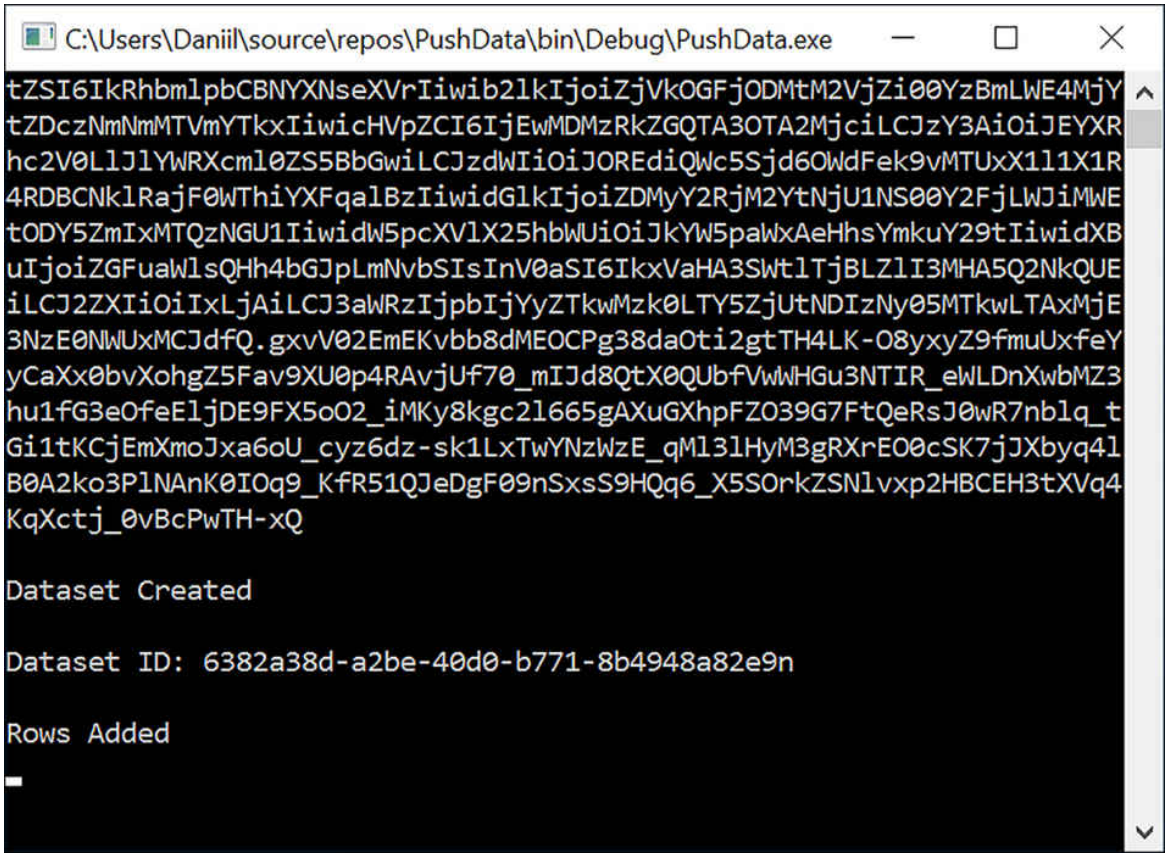


Figure 2.91 Application window

13. Pressing Enter for the fourth time will close the window.

You can now see the dataset in your workspace when you sign in to your account at *app.powerbi.com*, and you can even create a report from it.

> ***MORE INFO*** **PUSHING DATA INTO A DATASET**
>
> For a detailed overview of pushing data into Power BI datasets, including the description of Power BI dataset JSON object and table data

types, see “Push data into a Power BI dataset” at *https://docs.microsoft.com/en-us/power-bi/developer/walkthrough-push-data*.

Enable developers to create custom visuals

While Power BI already comes with many visuals that you can use right away, you can extend the number of available visuals by using custom visuals. You can download custom visuals created by others from custom visuals store, or you can develop your own visuals.

In Power BI Desktop, you can import a custom visual by selecting **Home** > **Custom visuals** > **From Marketplace, or From File**. Alternatively, you can click on the ellipsis that is displayed after the last visual icon in the Visualizations pane and select Import from File, or Import from Store. The first option, From Store, requires you to sign in to your account, after which you can browse the Power BI Custom Visuals store. If you already have a custom visual file, which has a .pbiviz extension, obtained from somewhere else, you can use the other option, From File. Once you import a custom visual using either option, it will appear in the Visualizations pane, and you will be able to use the new visual just like any other visual in your report. If you want to delete a custom visual, you can right-click on its icon in the Visualizations pane and select Delete Custom Visual. You can also delete multiple custom visuals from the ellipsis menu in the Visualizations pane.

To develop custom visuals, you need to have NodeJS installed with Power BI tools and a server certificate. To enable live preview of your custom visual, you will need to sign in to your Power BI account at app.powerbi.com and choose **Settings (gear icon)** > **Settings** > **General** > **Developer** > **Enable developer visual for testing**. This will add the developer visual to the list of available visuals in the Visualizations pane. The developer can then bind metrics to the new visual, testing the results live.

MORE INFO **DEVELOPING CUSTOM VISUALS**

For a detailed overview of what is required to develop custom visuals and how you can distribute them, see “Use developer tools to create custom visuals” at *https://docs.microsoft.com/en-us/power-bi/service-custom-visuals-getting-started-with-developer-tools*.

Thought experiment

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

You are the BI developer at Wide World Importers responsible for creating Power BI reports. You have already connected to data sources and imported data in Power BI Desktop. The loaded data can be found in CH02-ThoughtExperiment.pbix in the companion files folder for this book. The data model looks like Figure 2.92.

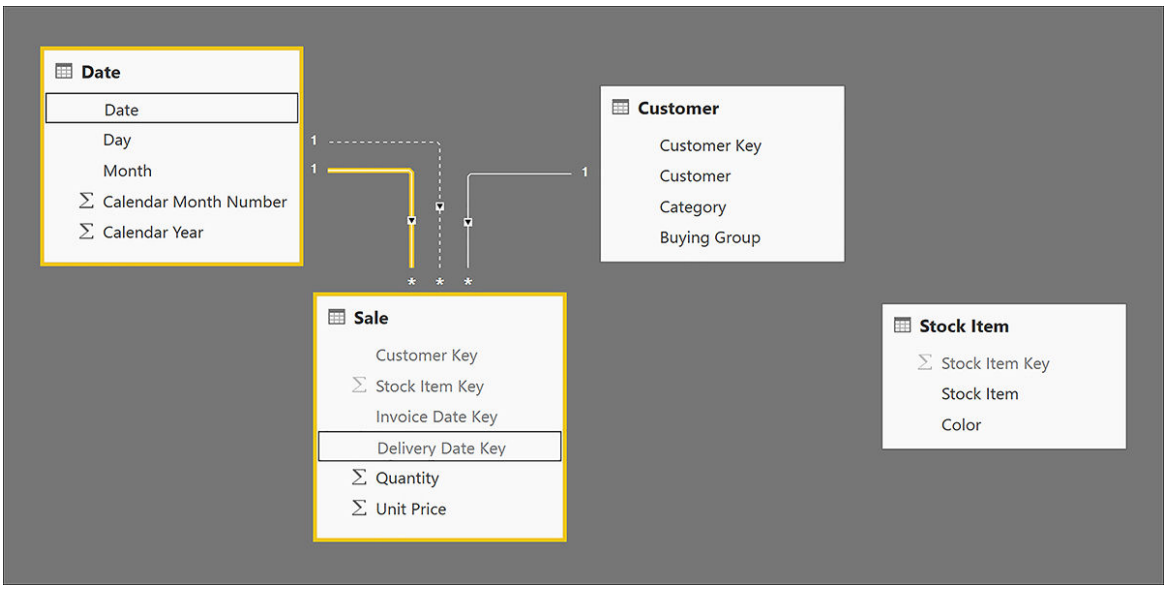


Figure 2.92 Wide World Importers data model diagram

The management requested a report based on historical data available. Based on background information and business requirements, answer the following questions:

1. You created a bar graph with Color on the axis and Quantity on the values. The visual is shown in Figure 2.93.

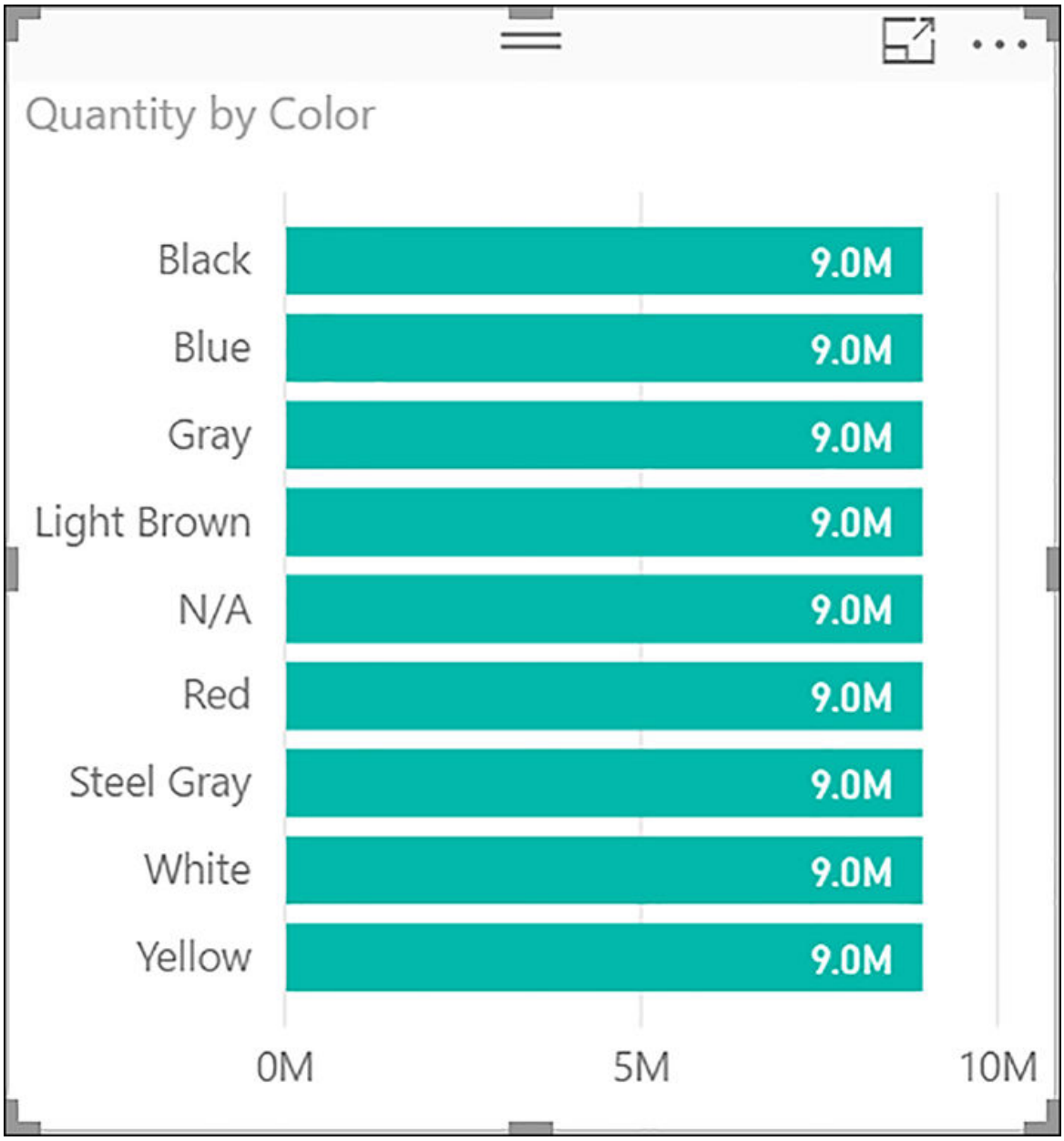


FIGURE 2.93 Bar graph showing Quantity by Color

Upon further checking, you find out that black stock items sold 1.1 million units. How can you fix the graph? The solution should use minimal effort and consider that you may want to analyze sales by metrics other than Quantity.

A. Create the following measure:

Click here to view code image

```
Total Quantity =
CALCULATE (
```

```
    SUM ( Sale[Quantity] ),
    TREATAS ( VALUES ( 'Stock Item'[Stock Item Key] ),
Sale[Stock Item Key] )
)
```

B. Create an active physical relationship between Sale and Stock Item.

C. Create the following measure:

Click here to view code image

```
Total Quantity =
CALCULATE (
    SUM ( Sale[Quantity] ),
    INTERSECT( Sale, 'Stock Item' )
)
```

D. Create a filtered calculated table for each color based on the Sale table.

2. You create a line chart that shows Quantity by Month. You notice that month values are sorted alphabetically. How can you remedy this? Your solution should involve minimal effort.

A. Use the ellipsis menu in the visual to change the sort order.

B. Create a new column that has a month number in front of month name .

C. Use the Sort by Column feature and sort Month by Calendar Month Number.

D. Create a measure for each month and place them in the right order.

3. You want to create a line chart showing Quantity by the invoice date. You notice that there is a relationship between the Date from the Date table and the Invoice Date Key from the Sale table, but the relationship is inactive. All other visuals in your report will be analyzing values by delivery date. How should you approach this problem?

A. Create the following measure:

Click here to view code image

```
Quantity by Invoice Date =
CALCULATE (
    SUM ( Sale[Quantity] ),
    USERELATIONSHIP ( 'Date'[Date], Sale[Invoice Date Key] )
)
```

B. Delete the relationship based on the delivery date and activate the relationship based on the invoice date.

C. Activate the relationship based on invoice date, keeping the other relationship as is.

D. Use the TREATAS function.

4. You need to create a column chart that displays Quantity by Color and can be drilled down to show Quantity by Stock Item. Which of the following solves the problem? More than one answer can be correct.

   A. Put Color and Stock Item on axis.

   B. Put Color on axis and Stock Item on legend.

   C. Create a Stock Item hierarchy that includes Color and Stock Item and use the hierarchy on axis.

   D. Put Color on Axis and Stock Item on Tooltips.

5. Is the following statement true? “The account you use for registering an app for embedding Power BI content does not need a Power BI Pro license allocated to it.”

   A. Yes

   B. No

6. You want to write a measure that calculates the ratio of a month of sales in a year. Which one of the following functions will you need to use?

   A. FILTER

   B. ENDOFYEAR

   C. WHOLEYEAR

   D. ALL

7. You add the Calendar Year column to a table visual and see only one row, as shown in Figure 2.94.

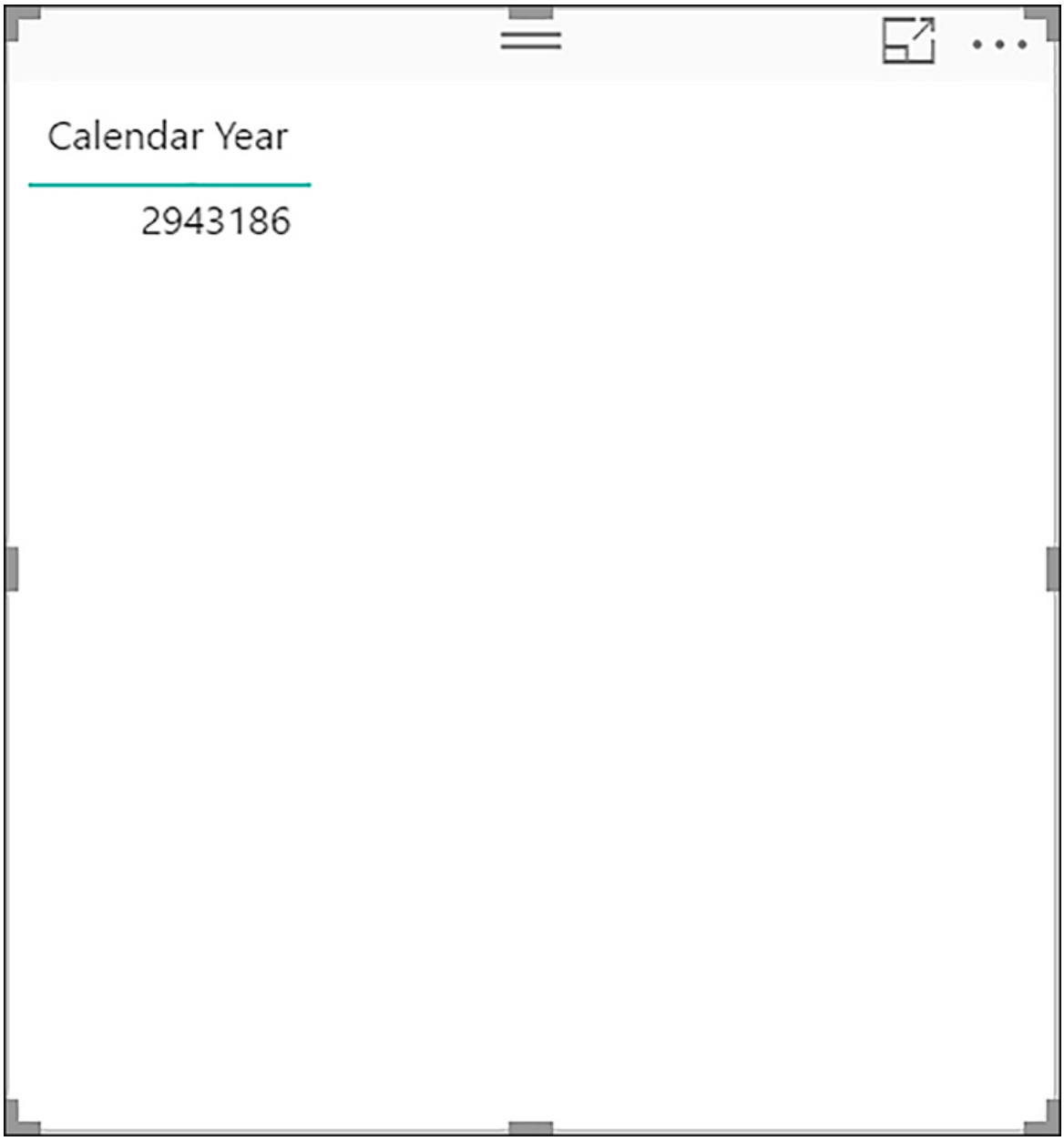


FIGURE 2.94 Calendar Year used in a table visual

Which of the following options solve the problem? More than one answer can be correct.

A. Right-click on Calendar Year in the Values field well and select Don’t Summarize.

B. Set the data type of the column to Text.

C. Set the default summarization to Don’t Summarize.

D. Add the Calendar Year field to the Rows field well of a matrix visual.

Thought experiment answers

1. The answer is **B**. Creating a physical relationship is the least arduous solution and will let you avoid creating virtual relationships like in answer A. While the measure from answer A will solve the problem of incorrect values, you will need to write a similar measure for each different metric you want to analyze. The measure formula from answer C will not work because Sale and Stock item have a different number of columns. Answer D involves too much effort and only solves the problem at hand; if you want to analyze quantity by individual stock items, this approach will fail.
2. The answer is **C**. When you sort Month by Calendar Month Number, the month names will appear in the right order in any visual where you use the Month column. The ellipsis menu discussed in answer A only allows you to change the sort order from ascending to descending or sort by Quantity instead of Month. The approach described in answer B may put month names in the right order, but it requires more effort and will add unnecessary clutter. Creating a measure for each month, like in answer D, will not solve the problem, because using multiple measures is like using a legend—you will see multiple dots instead of a sequence of values.
3. The answer is **A**. Because this is going to be a one-off visual, creating a measure that activates the relationship at query time solves the problem and involves the least effort. If you delete the relationship based on delivery date, as option B suggests, other visuals that use the Date table will display the wrong values. Following option C is not possible because no more than one relationship can be active at a time. Using TREATAS as per option D will result in the wrong figures.
4. The correct answers are **A** and **C**. With either option, you will be able to drill down in the hierarchy. Answer B is wrong because you cannot use the legend to drill down. If you follow the option D, you will not see stock item names in the visual.
5. The answer is **B**. You need to have a Power BI Pro user to register an app and embed Power BI content with Power BI Embedded.
6. The answer is **D**. With the ALL function, you can remove filter on month, allowing you to calculate the desired ratio. Neither FILTER nor ENDOFYEAR are required for the formula, so options A and B are incorrect. Option C is wrong because there is no such function as WHOLEYEAR in DAX.
7. Any of the answers will solve the problem.

Chapter summary

- In most Power BI data models, you need to create relationships between tables to produce the correct figures. In Power BI, it is only possible to create a relationship based on one column from each of the two tables you are relating. If you need to create a relationship between two tables based on more than one column per table, you can create a composite column in each table.
- You can create more than one relationship between two tables with no more than one active at a time. Inactive relationships can be activated programmatically with DAX. You can create one-to-many and one-to-one relationships in Power BI. A column that is on the one side of a relationship must contain unique values only without any blanks. To create a many-to-many relationship, you need to use a bridging table between two tables.
- Relationships can be either single-directional or bidirectional. If a relationship is bidirectional, then filters from one table reach the other table and vice versa. A single-direction relationship only passes filters in one direction.
- You can change the sort order of values in a column by using the Sort by Column feature. You can also apply a different sort order in some visuals by sorting by values instead of categorical items. This is done in the ellipsis menu of a visual. You can specify either ascending or descending order.
- Columns that exist only for technical purposes, such as keys and IDs, can be hidden from reporting view to make browsing the data model easier for report creators.
- Columns and measures can be formatted to make them appear different visually, such as having a percentage sign or a different number of decimal places.
- In addition to importing data from a data source, you can enter data manually using the Enter Data feature. Furthermore, you can create custom M queries in Power Query Editor or create custom DAX tables to achieve a similar effect.
- In addition to M, Power BI uses a language called DAX, which stands for Data Analysis Expressions. DAX can be used to enrich your data model with more columns, tables, and measures.

- A calculated column differs from a custom column created in Power Query Editor: the former is created from data that has already been added to the data model. A formula that you write for a calculated column is automatically applied for each row, and it must result in a scalar value. Once a calculated column is created, it is stored in memory and on disk.
- Many functions in DAX return tables, which can be materialized as calculated tables. You can often, but not always, nest table functions. Calculated tables, like calculated columns, also use RAM and disk space.
- Formulas that result in scalar values can be used to create measures in DAX. Unlike calculated columns and calculated tables, measures do not consume disk space and RAM; instead, they are calculated at the time the query is run, and they use the CPU.
- You must create a calculated column if you want to filter by its values in a slicer or elsewhere. Another reason to create a calculated column is to pre-compute and store the results of a very complex formula that would result in poor performance if it were used in a measure.
- There are two evaluation contexts in DAX, which always co-exist: row context and filter context. The row context can be thought of like the current row; in it, you can refer to values from other columns in the same row without aggregating them. The filter context filters the data model based on applied filters. The sources of filter context include, but are not limited to: slicers; visual-, page-, and report-level filters; axes; matrix rows and columns. In filter context, there is no concept of “current row.” To calculate a scalar value in filter context, you must use aggregation functions. Either context can be empty at a time.
- CALCULATE and its sister function, CALCULATETABLE, are the only functions in DAX that can change filter context. Another important role of the functions is context transition: they convert a row context into an equivalent filter context.
- You can use variables almost anywhere in DAX, which includes formulas for calculated columns, calculated tables, and measures. Variables are calculated only once in the context that they are defined in, and then they become immutable, meaning that even CALCULATE cannot change them. Variables can be useful to avoid code repetition and improve performance.
- With What If parameters, you can see how changing a parameterized metric affects some of your calculations.

- To compare the actual against the target, you can use a KPI or a gauge visual. Furthermore, you can visualize multiple metrics with a multi-row card visual. For a single metric, you can use the card visual.
- By default, Power BI Desktop creates a date hierarchy for each date or datetime type column. If this is undesirable, this feature can be turned off. You can create your own hierarchies based on business needs to make browsing the data model easier. Using a hierarchy is equivalent to using the columns that it is made of together. There are parent-child functions in DAX that enable you to flatten the parent-child hierarchies.
- Power BI Desktop currently contains about 30 built-in visuals, and you can extend your visualization options by using custom visuals. You can import custom visuals created by others, or you can develop your own.
- Page layout and formatting can be configured for each page individually. You can either use a predefined size or specify a custom one. In addition to this, you can duplicate pages.
- When you click on an item in a visual, it can cross-filter or cross-highlight other visuals. Alternatively, visuals can be set to ignore other visuals when interacting with them. Cross-filtering refers to filtering the data behind the visual; cross-highlighting means keeping all data and highlighting the filtered segments. Not all visuals can be cross-highlighted.
- By default, Power BI hides categorical items whose corresponding values are blank. If needed, you can display categories that have no data either using the visual settings, or you can do it programmatically with DAX.
- To enable the correct processing of data by Power BI, especially for use with a map visual, you can categorize it. You can also change the default summarization of a column.
- Visuals can be moved around with mouse or arrow keys, as well as by specifying coordinated in the Format pane. Visuals can also be aligned or distributed evenly on the canvas using the Visual Tools Format tab.
- You can use R visuals inside Power BI Desktop, for which you need to install R server on your machine. R visuals can also be used in reports published online, subject to the necessary libraries being available in Power BI service.
- Bookmarks allow you to draw the attention of users to specific aspects of your reports by saving page states. You can navigate between bookmarks using the View mode or by linking bookmarks to shapes or images.

- With report themes, you can apply your own color scheme to your reports, as well as specify your own default formatting values for each visualization type.
- To use Power BI Embedded, you need to configure dedicated capacity, which is done in Azure portal. For this, you need an Azure subscription and Azure Active Directory (AD) with at least one Power BI account in it.
- Developers can create custom report layouts that override the layouts inherited from Power BI Desktop.
- To create Power BI Embedded solutions, you need to have a Power BI Pro license and an Azure Active Directory app registered with the necessary permissions. You can use a single Azure Active Directory tenant or a different one for each customer. The reports you are embedding need to be published in an app workspace that uses a dedicated capacity.
- With Power BI REST API, you can create datasets and add rows to them programmatically. A registered Azure AD app is required to perform this task.

CHAPTER 3

Configure dashboards, reports, and apps in the Power BI Service

In previous chapters, we primarily reviewed the skills necessary to work in Power BI Desktop. In this chapter, we are going to review the skills you need to work with another product in the Power BI family—Power BI service. You can publish your reports to Power BI service and keep them up to date by enabling scheduled refresh of your datasets. In Power BI service, you can also create dashboards that may include visuals from multiple datasets, which you can then share with others either directly or by packaging them in apps. Furthermore, you can configure security in Power BI service to make sure that users can only access the data that they are supposed to access.

Skills in this chapter:

Skill 3.1: Access on-premises data

When you create your reports based on on-premises data and publish them online, to refresh your datasets, you will need a way to access your on-premises data sources. This can be achieved with a data gateway, which is covered in this section. We are also going to review how you can use Power BI Desktop to publish to and edit reports from Power BI service.

This section covers how to:

- Connect to a data source by using a data gateway
- Publish reports to the Power BI service from Power BI Desktop
- Edit Power BI service reports by using Power BI Desktop

Connect to a data source by using a data gateway

In most cases, to share the reports you created in Power BI Desktop, you need to publish them to Power BI service in the cloud, also known as PowerBI.com. Once this happens the mechanics of refreshing your dataset change, which means the cloud, not your machine, needs to have access to your data sources.

A gateway is a piece of software that acts as a bridge between the cloud and your on-premises data sources. With a data gateway, you can not only access your on-premises data sources but also schedule refresh for the datasets published to Power BI service.

Installing a data gateway

To install a gateway, you need to sign into PowerBI.com and click on the **Download** button in the top-left corner of the screen, then click **Data Gateway**. You can see the button in Figure 3.1.

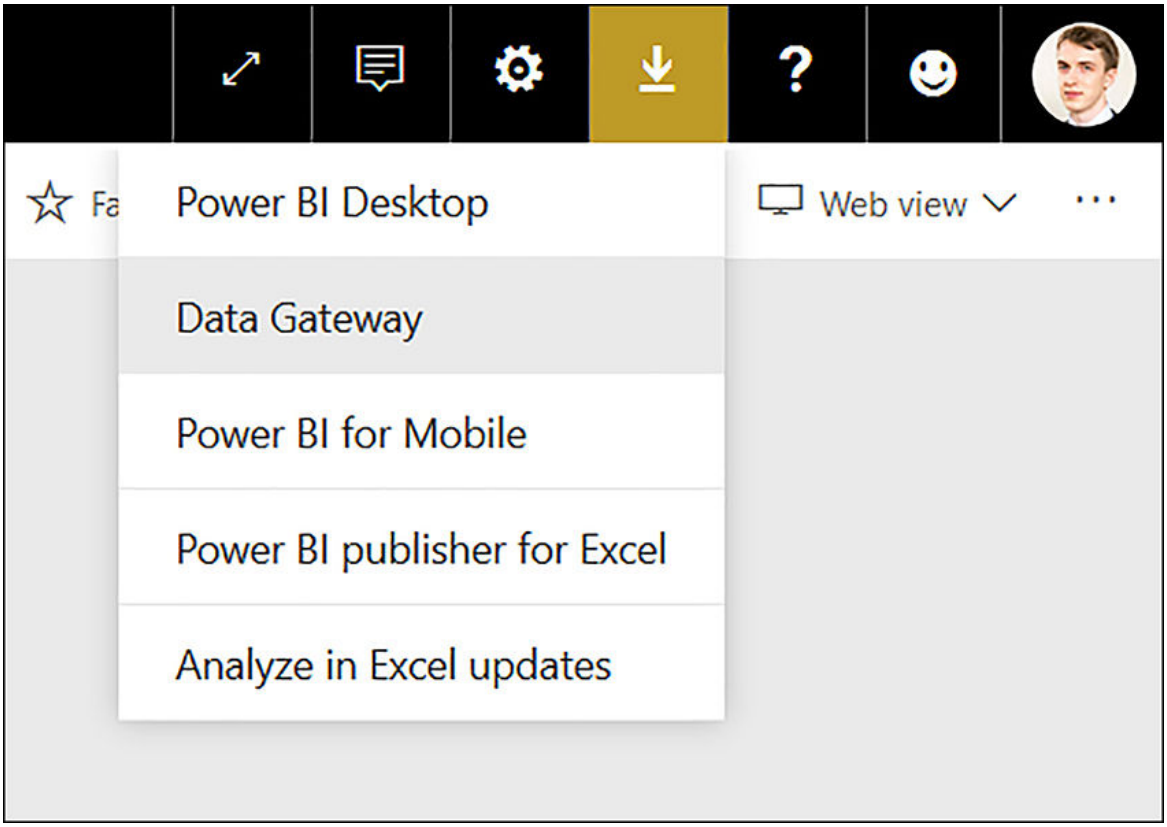


FIGURE 3.1 Data Gateway download button

The Power BI Gateway can be installed in two modes:

- **On-premises data gateway** This gateway can be used by multiple users that have access to the server onto which you install the gateway. It can be used for both scheduling refresh and live queries in Power BI. You can also use it for PowerApps, Logic Apps, and Microsoft Flow.
- **On-premises data gateway (personal mode)** Only you can use this, and you can use it only for scheduling refresh in Power BI. The Live Connection connectivity mode, PowerApps, Logic Apps, Microsoft Flow are not supported.

EXAM TIP

Be aware of the functionality difference between the two gateway installation modes and be prepared to decide which one is appropriate in any given situation.

In general, a data gateway should be installed on a machine that is always on and connected to the Internet, because a gateway cannot access on-premises data sources from a machine that is powered off. You can install up to one gateway in each mode on the same computer, and you can manage multiple gateways from the same interface on PowerBI.com. If you want to follow the examples later in this chapter, you should install an on-premises data gateway.

During the installation process, you will need to sign in to your Power BI account, and you will need to give your gateway a name. You will also need to specify a recovery key that you can use to move or recover the gateway. Once you are finished installing the gateway, you will need to add data sources and users to it.

MORE INFO **INSTALLING A GATEWAY**

For more details on the data gateway installation process, including requirements and considerations, see the "Install the on-premises data gateway" section of "On-premises data gateway" at *https://docs.microsoft.com/en-us/power-bi/service-gateway-onprem#install-the-on-premises-data-gateway*.

Adding data sources to a data gateway

Once you have a gateway installed, you can add data sources to it in Power BI service. For that, you need to start by clicking **Settings** > **Manage Gateways**. At

this point, you should see a page like the one shown in Figure 3.2.

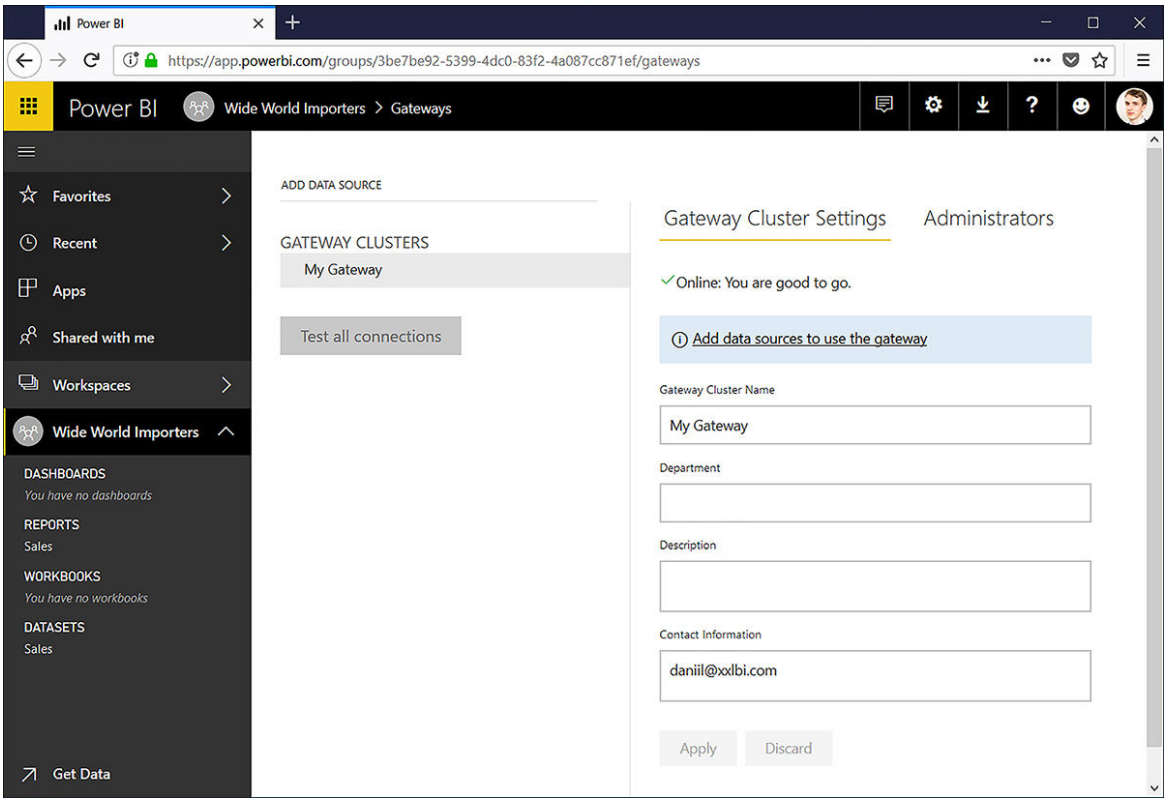


FIGURE 3.2 Gateway settings

If needed, you can enable others to administer the gateway in the Administrators tab. Note that this does not automatically grant access to each data source in the gateway; you will need to add users to each data source, which is covered later in this section.

You can add a data source by clicking on the **Add Data Sources To Use The Gateway** hyperlink. Alternatively, you can click on the **Add Data Source** button above the list of gateways, or you can click on the ellipsis next to a gateway name and select **Add Data Source**. You will then need to give your data source a name and select its type. Currently, Power BI gateways support over 20 data source types, including the following popular ones:

- SQL Server
- Analysis Services
- File
- Folder
- Web

- OData

In most cases, you do not need a data gateway if you want to access a data source that is on the web. One of the exceptions is the Web data source, for which you currently need a data gateway, even though it is not an on-premises data source.

> ***NOTE*** **DATA SOURCE TYPES AND GATEWAYS**
>
> Some data types can be used for DirectQuery or Live Connection, while others support importing data only. For more details, see the "List of available data source types" section in "On-premises data gateway" at *https://docs.microsoft.com/en-us/power-bi/service-gateway-onprem#list-of-available-data-source-types*,

In the previous chapters, we created a report in Power BI Desktop that combined data from SQL Server and files. If we want to keep our dataset up to date, we need to schedule refresh, which can be done by clicking on the ellipsis next to the dataset and selecting **Schedule Refresh**. If you have not yet configured your data sources properly, you will see the error shown in Figure 3.3.

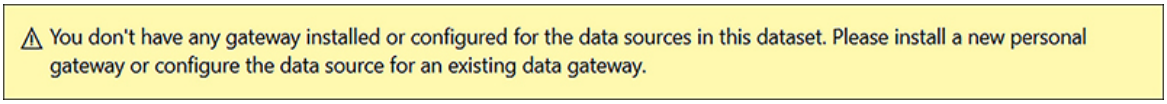


FIGURE 3.3 Data sources error

Even though we have a gateway installed, to schedule refresh, we need to add all data sources first. We can start with SQL Server. Once you choose the SQL Server data source type, you will need to specify the server and database name, authentication method, username, and password. In the advanced settings, you have an option to use SSO via Kerberos for DirectQuery queries and choose a privacy level setting for this data source. In our case, we can specify just the server and database name, authentication method and credentials, leaving the advanced settings as-is. Note that you cannot leave the database name blank, even though you can connect to a server without specifying a database in Power BI Desktop.

After you enter all settings and click **Apply**, the webpage is going to look like in Figure 3.4.

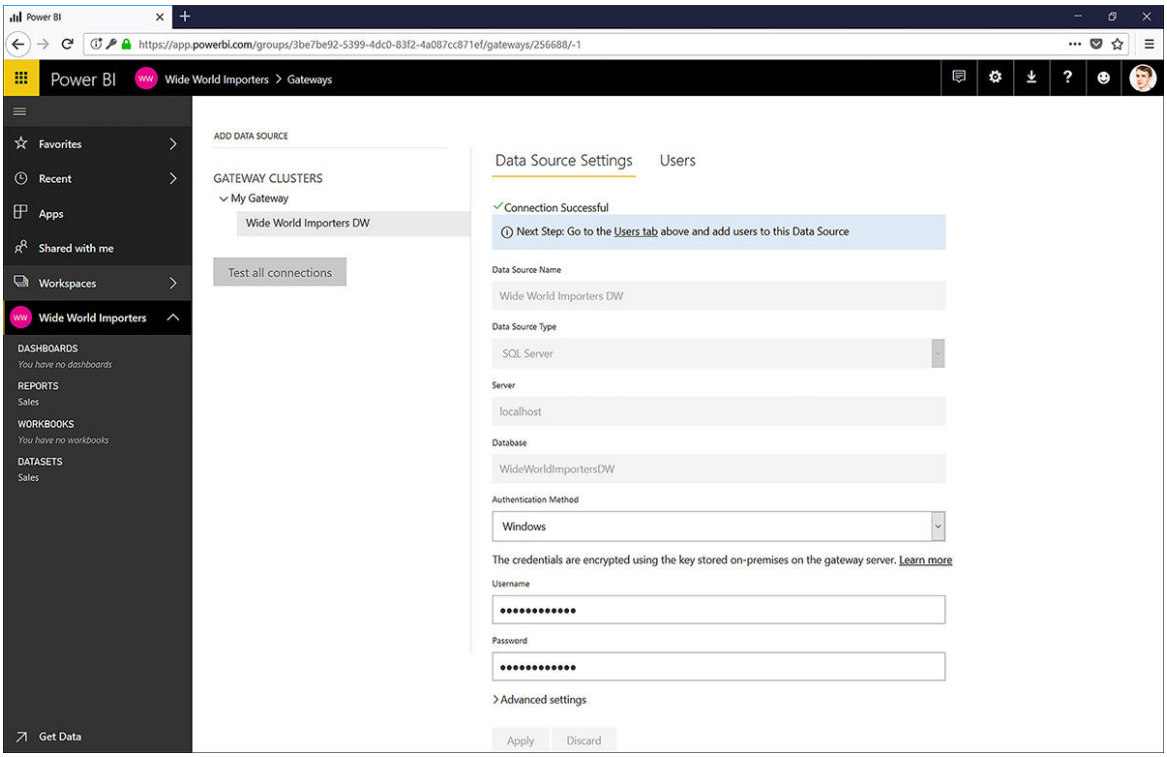


FIGURE 3.4 Data source added

At this stage, you can add users to your data source. In the **Users** tab in **Data Source Settings**, you add users by specifying their email addresses. You can see an example in Figure 3.5:

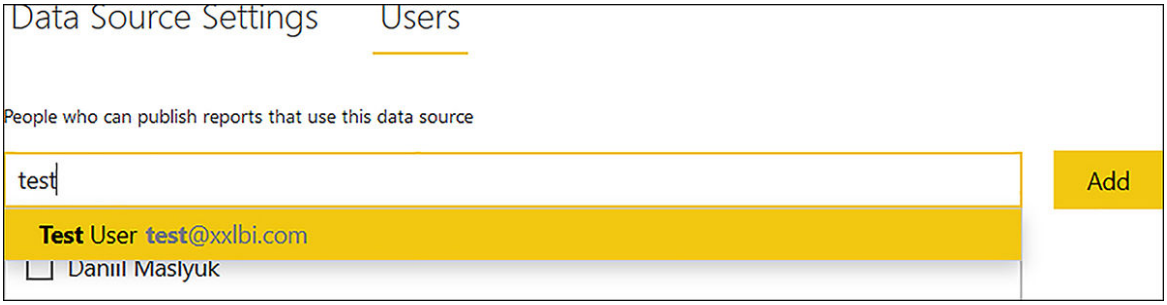


FIGURE 3.5 Adding a user to a data source

If you installed the gateway yourself, you are added as a user automatically. For the purposes of our example, you do not need to add anyone else.

> ***MORE INFO*** **SQL SERVER DATA SOURCE**
>
> For more details on adding and configuring a SQL Server data source, including DirectQuery considerations, see “Manage your data source—

SQL Server" at *https://docs.microsoft.com/en-us/power-bi/service-gateway-enterprise-manage-sql*.

Because SQL Server is not the only data source we are using in our report, we also need to make sure we can access our files. There are two ways of doing this:

- Add each file individually as a data source
- Add a data source of type Folder

The option you choose depends on your business requirements. In most cases, unless sensitive files are in the same folder as the files you want to grant access to, you can add a folder as a data source. In our example, we'll add "C:\Companion" as a data source.

Scheduling refresh

Once all of the necessary data sources are added to our gateway, click on the ellipsis of our dataset and select **Schedule Refresh**. We then need to expand the **Gateway Connection** section and select the **Use A Data Gateway** radio button. It is important to click **Apply** afterward; otherwise, you will get an error in the Data Source Credentials section as shown in Figure 3.6.

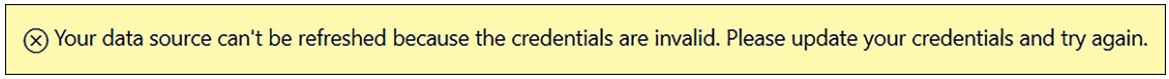


FIGURE 3.6 Credentials error

To determine whether the gateway can access your data sources successfully, you can click on the ellipsis next to a dataset and select **Refresh Now**. Clicking on the ellipsis again will either display an error (if the refresh attempt was unsuccessful) or the following message will appear: "A refresh is in progress. Please wait for the current refresh to finish before clicking Refresh Now." After some time—depending on the volume of data—you will see a message similar to this: "LAST REFRESH SUCCEEDED: Fri May 04 2018 11:08:53 GMT+1100 (AUS Eastern Daylight Time)."

At this stage, we can schedule the refresh in the Scheduled Refresh section. Start by toggling the **Keep Your Data Up To Date** option to **On** and then choose **Refresh Frequency** and **Time**. By default, refresh failure notifications will be sent to you via email. Click **Apply** for your settings to take effect.

***MORE INFO* SCHEDULING REFRESH**

For more information on how you can schedule refresh, see the "Using the data source for scheduled refresh" section of "Manage your data source - Import/Scheduled Refresh" at *https://docs.microsoft.com/en-us/power-bi/service-gateway-enterprise-manage-scheduled-refresh#using-the-data-source-for-scheduled-refresh*.

Publish reports to the Power BI service from Power BI Desktop

On the Power BI Desktop Home ribbon, you will see a **Publish** button, which allows you to publish your report to the Power BI service. If you click the **Publish** button and you are not signed into Power BI, you will be prompted to enter your credentials. If you have more than one workspace for publishing, you will need to choose one. Furthermore, if the workspace you are publishing to already contains a dataset with the same name, you will be asked if you want to replace it. By default, the report will be published to My Workspace.

For example, if you publish a report called Sales to an empty workspace, you will receive the Success message, which you can see in Figure 3.7.

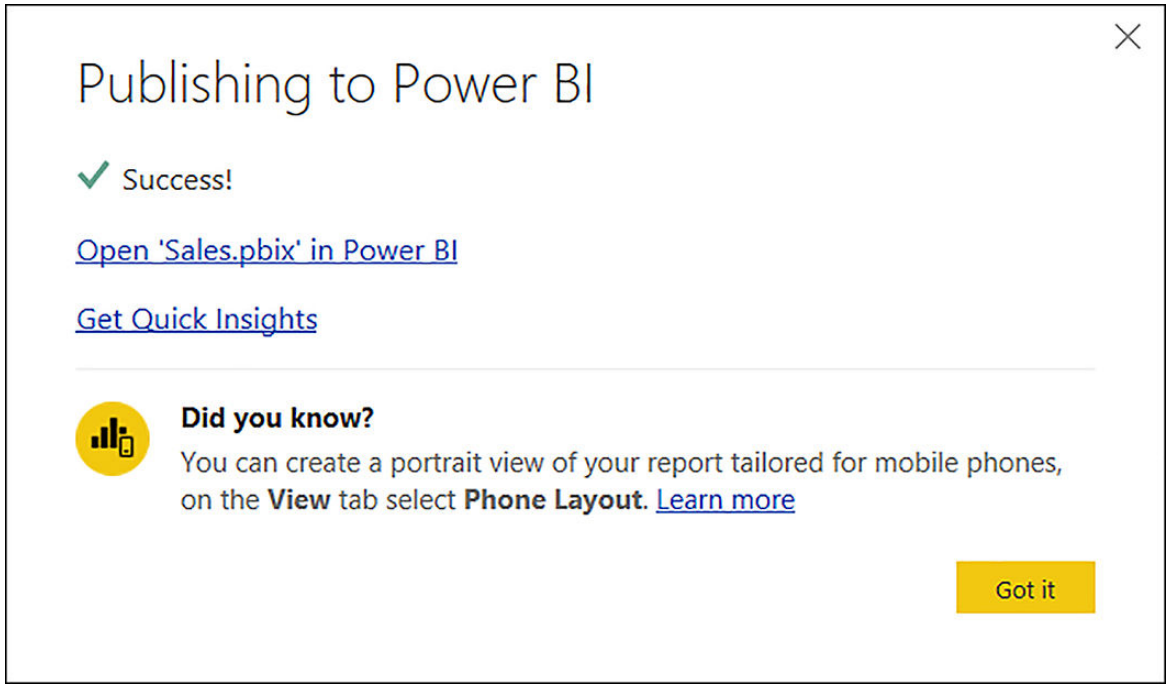


FIGURE 3.7 Publishing to Power BI window

Once you are done, you can go to Power BI service and see the report.

***MORE INFO* RE-PUBLISHING FROM POWER BI DESKTOP**

When you re-publish a report to Power BI service, you should be aware of certain considerations. For example, certain format changes may be ignored. For more details, see “Publish from Power BI Desktop” at *https://docs.microsoft.com/en-us/power-bi/desktop-upload-desktop-files*.

Edit Power BI service reports by using Power BI Desktop

The report editing experience in Power BI service is similar to Power BI Desktop, though you are limited to only editing the report itself and not the data model behind it. For instance, you can create new visuals, but you cannot create new measures.

If you want to edit a Power BI service report in Power BI Desktop, you can download it as a .pbix file. For that, you need to open your report in Power BI service, then click **File** > **Download Report**. You can see this option in Figure 3.8.

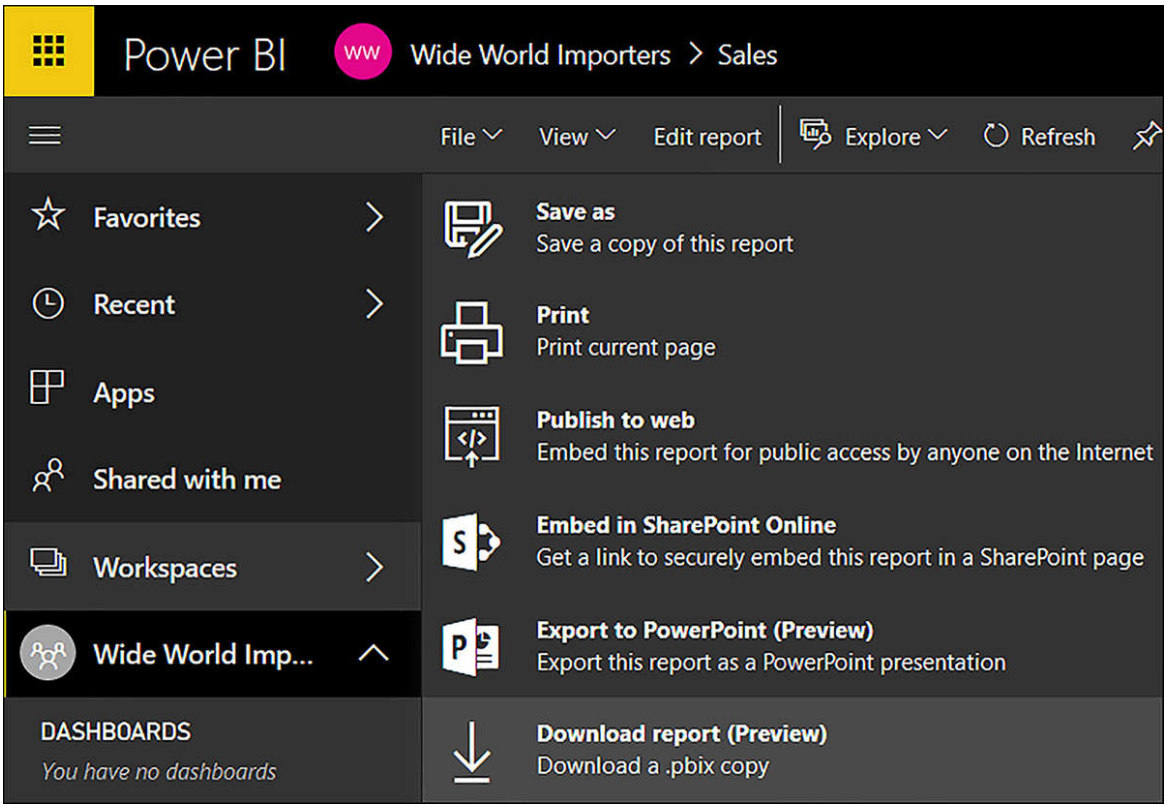


FIGURE 3.8 Download Report option

Clicking **Download Report** makes Power BI service export the current report as a .pbix file, which you can then save on your computer and open in Power BI Desktop. If your report is modified in Power BI service with the new features that

are not available in the Power BI Desktop version that you are using, you will get the following note when opening the report: “Report layout differences might exist.” You can see the full message in Figure 3.9.

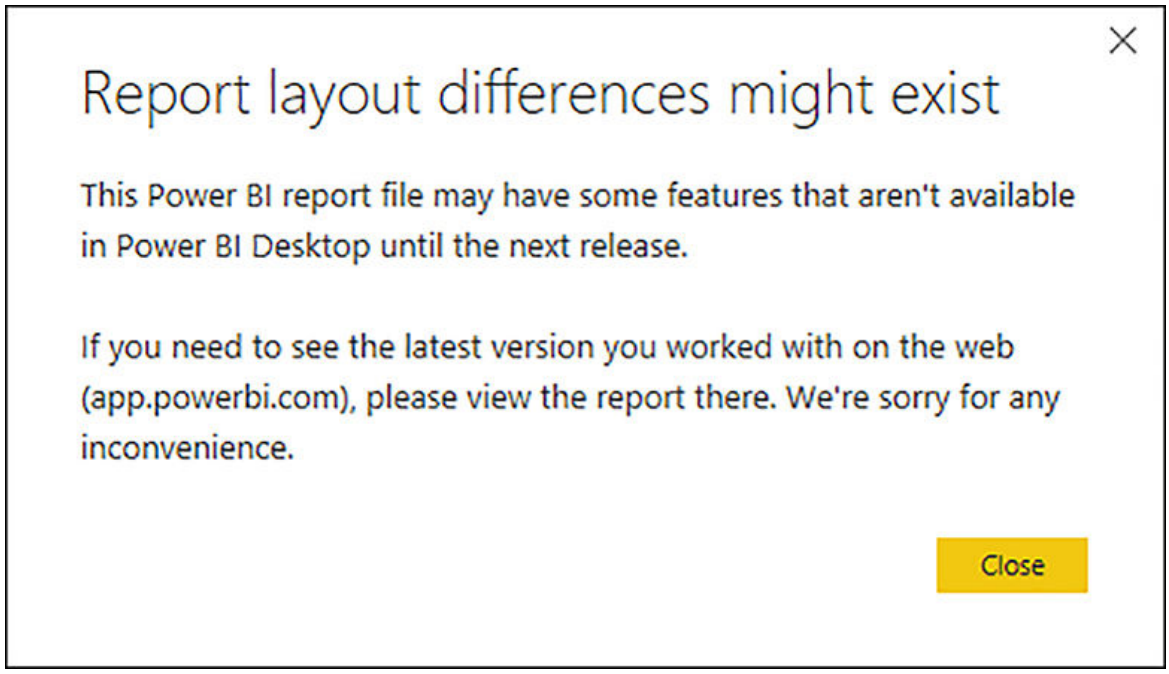


FIGURE 3.9 Report layout differences message

Not all reports can be downloaded from Power BI service. For example, you cannot download reports published to Power BI service before 23 November 2016. Also, you can only download the reports to which you have edit access.

MORE INFO **EXPORTING REPORTS FROM POWER BI SERVICE**

For more information on how to export reports from PowerBI.com, including a list of limitations and a video overview, see “Export a report from Power BI service to Desktop” at *https://docs.microsoft.com/en-us/power-bi/service-export-to-pbix*.

Skill 3.2: Configure a dashboard

So far in Power BI service, we have worked with datasets and reports. One of the limitations of a report is that it can contain data only from one dataset. When you want to show visuals from multiple datasets side-by-side, you can display them in

a dashboard. In this section, we are going to review dashboards and how they can be configured.

This section covers how to:

- Add text and images
- Filter dashboards
- Dashboard settings
- Customize the URL and title
- Enable natural language queries

Add text and images

A Power BI dashboard is a single-page collection of visuals, which can be visuals from reports, as well as other objects such as text and videos. Dashboards are a feature of Power BI service; they are not available in Power BI Desktop. In the context of dashboards, visuals are called tiles, and the process of adding a tile to a dashboard is called pinning.

***MORE INFO* DASHBOARDS IN POWER BI**

For a more detailed explanation of Power BI dashboard, how it can be used, and the related terminology, see "Dashboards in Power BI service" at *https://docs.microsoft.com/en-us/power-bi/service-dashboards*.

The dashboard that is created automatically after you publish a report contains one tile only, which also has the same title as the report. If you click on the tile, you will be taken to the report.

There are several ways in which you can add more tiles to a dashboard. When you are in a report and hover over a visual, you will see a pin icon. Clicking the **Pin Visual** button opens the **Pin To Dashboard** dialog box, which you can see in Figure 3.10.

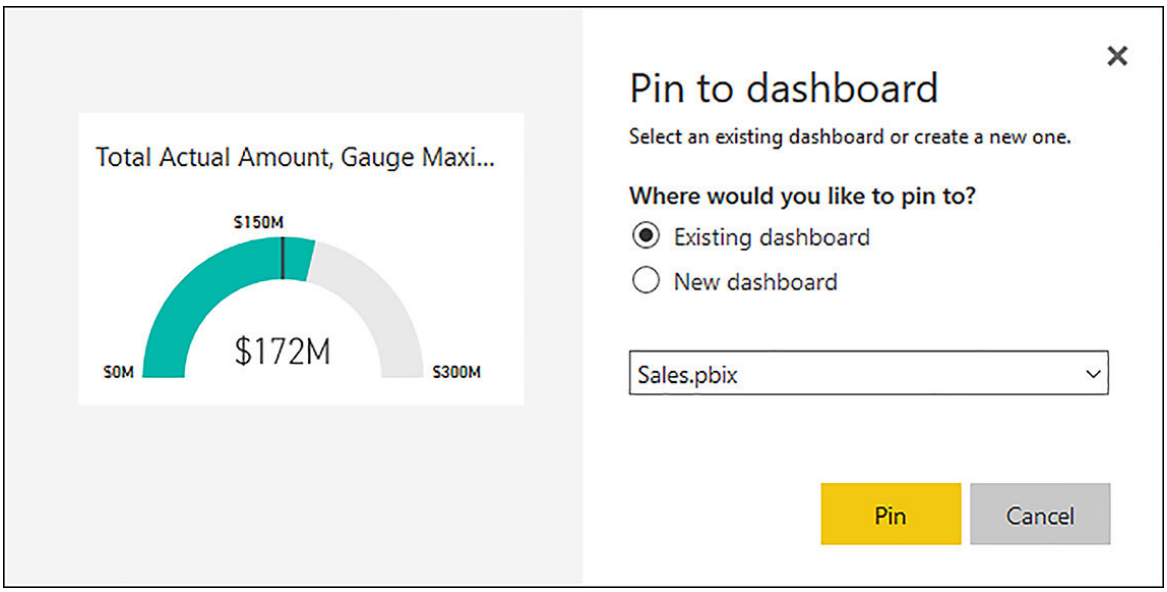


FIGURE 3.10 Pin to dashboard dialog box

> ***NOTE*** **REPORT THEMES**
>
> If you are using a custom theme in your report, you will also have an option of keeping the theme or using destination theme. Report themes are covered in more detail in Skill 2.5: "Create and format interactive visualizations."

If you choose to pin the tile to an existing dashboard, you can select a dashboard from a drop-down list. Alternatively, you can specify a new dashboard name to create a dashboard with the visual you are pinning. Once you pin a visual to a dashboard, you will see the Pinned to Dashboard message, and you will see a new tile in the dashboard.

> ***IMPORTANT CHANGES TO PINNED VISUALS***
>
> If you alter a pinned visual in the report—for example, you change formatting—the tile will stay the same and keep the original formatting. To update the tile, you will need to remove it and pin the visual again.
>
> By default, clicking on a tile takes you to the report that contains the visual. In contrast to report visuals, there is no interaction between tiles in a dashboard. If you want your visuals to be interactive, you can pin a live page to a dashboard. This pins a whole report page to a dashboard, and you can click on visuals for cross-filtering and cross-highlighting. To pin a live page, you need to go to a report page of your choice and

click **Pin Live Page** in the top menu. You will be presented with the same dialog box shown in Figure 3.10, only this time showing a preview of a live page with a **Pin Live** button instead of a **Pin** button. Changes in the report page will be reflected in a pinned live page.

***MORE INFO* PINNING FROM REPORTS**

For more information on how you can pin a visual or a live page to a dashboard, including a video overview, see "Pin a tile to a Power BI dashboard from a report" at https://docs.microsoft.com/en-us/power-bi/service-dashboard-pin-tile-from-report.

Another way to add a tile is to click on the **Add Tile** button when you are in a dashboard; this gives you the following options:

- Web Content
- Image
- Text Box
- Video
- Custom Streaming Data

While you can pin an image visual from a Power BI report, the **Add Tile** option allows you to pin an image that is not contained in any report. When you select **Image** and click **Next**, you will be able to specify the URL of an image. Optionally, you can specify a title and subtitle. Clicking **Apply** will pin the image to your dashboard.

***MORE INFO* SETTING CUSTOM LINKS**

For each tile type in the Add tile menu, you can specify a custom URL that the user will be taken to the tile is clicked. This feature is going to be covered later in this section.

To add text to your dashboard, you can either pin an existing text visual from a report or choose the Text Box option in the **Add Tile** menu in a dashboard. If you choose the latter, you will be presented with a text editor, where you can use rich text formatting and create hyperlinks.

Once a tile is added, you can edit its settings by hovering over the tile, clicking the ellipsis in the top-right corner and selecting **Edit Details**.

***MORE INFO* DASHBOARD TILES**

For a tutorial on how you can edit pinned tiles, see "Edit or remove a dashboard tile" at *https://docs.microsoft.com/en-us/power-bi/service-dashboard-edit-tile*.

For more information on what dashboard tiles are, how you can interact with them, and other ways in which you can pin a tile, see "Dashboard tiles in Power BI" at *https://docs.microsoft.com/en-us/power-bi/service-dashboard-tiles*.

***MORE INFO* ADDING TILES TO A DASHBOARD**

Details on how you can add videos, web content, and custom streaming data are outside of scope of this book. For more information on the topic, including a video overview, see "Add image, text, video, and more to your dashboard" at *https://docs.microsoft.com/en-us/power-bi/service-dashboard-add-widget*.

Filter dashboards

Currently, it's not possible to filter a dashboard in the same manner as you can filter a report. There are a few options to restrict what a user can see in a dashboard.

First, if you only want to remove some tiles, you can duplicate a dashboard and keep only the visuals you want. To duplicate a dashboard, you need to navigate to the dashboard, click on the ellipsis in the top-right corner and select **Duplicate Dashboard**. You will need to specify a new dashboard name, and you will then be able to remove the unwanted tiles. This option keeps the displayed values the same.

***MORE INFO* CREATING A COPY OF A DASHBOARD**

For more details on the process of copying a dashboard, see "Create a copy of a dashboard in Power BI service" at *https://docs.microsoft.com/en-us/power-bi/service-dashboard-copy*.

Second, you can filter visuals in a report and pin them in their filtered state to a dashboard. You do not need to save a filtered report; the tiles will inherit all applied filters. This option requires the creation of a new dashboard if you want to only display filtered tiles.

***MORE INFO* DASHBOARD FILTERS**

To read more about what is involved in filtering dashboard contents, see an article by PowerDAX, "Power BI Dashboard Filters" at *https://powerdax.com/power-bi-dashboard-filters/*.

Third, you can employ Row-Level Security, which filters underlying data depending on security rules defined for each user. This option allows you to keep a single dashboard while showing different values to different users.

***MORE INFO* ROW-LEVEL SECURITY**

Configuring Row-Level Security needs to be done in Power BI Desktop before it can be used in Power BI service. We are going to review this feature in Skill 3.4: "Configure security for dashboards, reports, and apps."

Dashboard settings

When you publish a report, a dashboard is created automatically, and it has the same name as the file you publish. You can rename a dashboard by right-clicking on it or clicking on the ellipsis next to its name and selecting **Rename**. Alternatively, you can select **Settings**, or, when you are in the dashboard, you can click on the ellipsis in the top-right corner of the screen and select **Settings**. You will then be able to specify a new dashboard name.

In dashboard settings, you can also enable or disable Q&A and dashboard tile flow; the latter option helps you with maintaining a contiguous tile layout in your dashboard and is disabled by default. We are going to cover the Q&A feature, which is enabled by default, later in this section.

Customize the URL and title

In a Power BI dashboard, you can set a title and subtitle for every tile, as well as display the last time the tile was refreshed. You can set these options if you hover over a tile and click **Edit Details**. You can turn off the title and subtitle by unchecking **Display Title And Subtitle**. If you enable the title and subtitle but leave both fields blank, the tile inherits the title from the report visual, if any. If you tick the **Display Last Refresh Time** checkbox in the Functionality section, the title and subtitle will be forced to be displayed; you will need to specify a custom title. Otherwise, you will get the following error: "Title can't be empty."

In addition to this, almost any tile can be set to be a link. By default, if a tile is a pinned visually from a report, clicking on the tile will take you to the report that contains the visual. To set a custom link for a tile, you need to enable the **Set Custom Link** option in the **Functionality** section. You will then need to choose between two link types:

- **External Link** You can point to a URL outside of PowerBI.com.
- **Link To A Dashboard Or Report In The Current Workspace** This allows you to point to a dashboard or a report within the same workspace.

If you choose the first option, you will need to specify a URL. You will also have an option to open the link in the same tab; by default, the link will open in a new tab.

Choosing the second option allows you to select a report or dashboard to link to from the current workspace. Note that the report or dashboard you select from the drop-down list does not need to be related to the tile; it can be any report or dashboard from the current workspace. You can see the Tile Details pane in Figure 3.11.

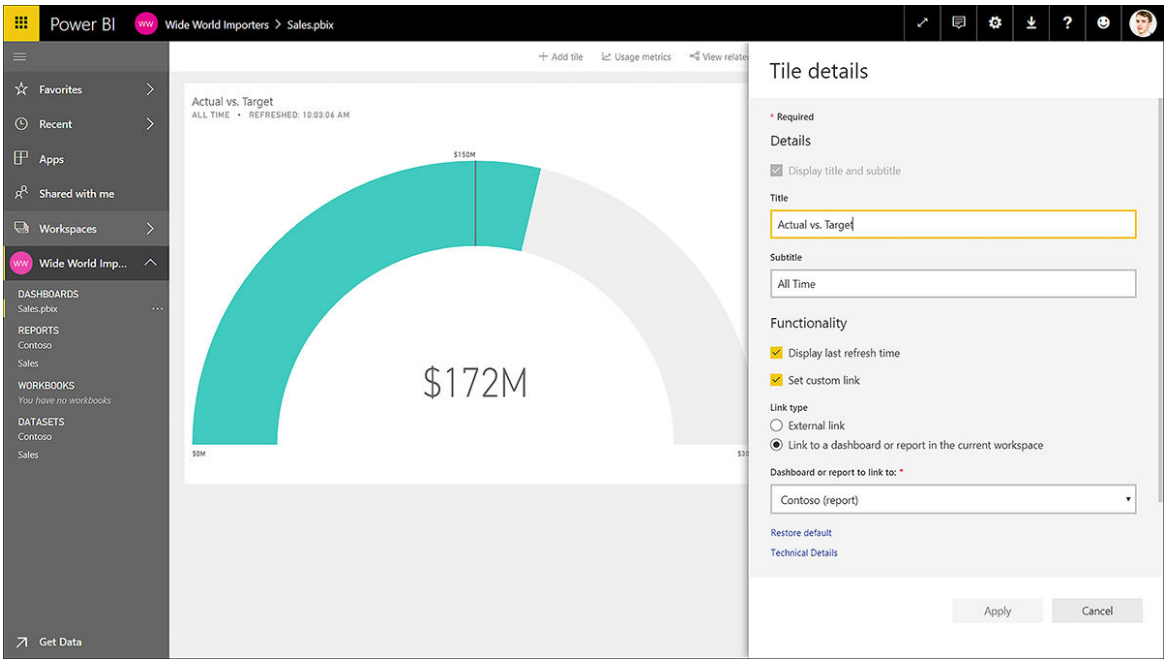


Figure 3.11 Tile details

Once you click **Apply**, you can use the tile as a link.

NOTE **HYPERLINKS IN TEXT BOXES**

If you have a hyperlink as part of your text box, you will still be able to click on it even if you set a custom URL for your tile.

Enable natural language queries

Both Power BI Desktop and Power BI service allow you to create visualizations that provide answers to specific questions. This often requires picking fields and placing them in the relevant field wells of a visual of your choice. While this gives you great control over formatting, it will not work if you have read-only access to content.

Another way to explore data in Power BI is to use Q&A (also known as natural language queries). This feature enables you to get answers to your questions by typing them in natural language. Even users with read-only access can query datasets in a natural language.

Q&A in Power BI service

To use the Q&A feature in a dashboard, you need to make sure that it is enabled in dashboard settings. If it is enabled, the Q&A box appears above all visuals in the dashboard canvas. You can see the question box as it appears for a user with read-only access in Figure 3.12.

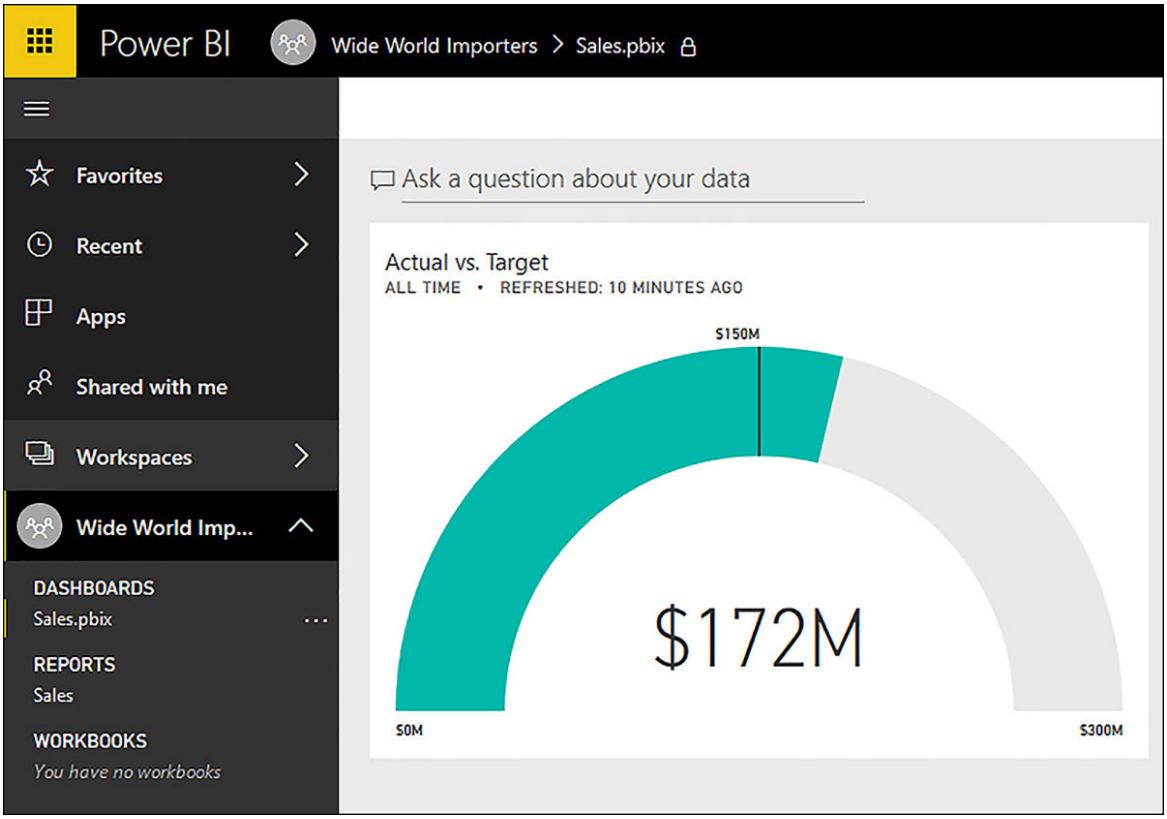


Figure 3.12 Q&A question box

In a natural query, you can define categorical and numerical values, as well as the chart type. For example, you can type the following query:

```
total profit by calendar year label as column chart
```

Once you click on the Q&A box, a list of terms that appear in the dataset will be displayed. These may include fields, hierarchies, tables, and page names. It is possible to add your own featured questions to this list so that they are displayed when a user clicks on the Q&A box in the Power BI service. For that, you need to follow the next steps:

1. Click on the ellipsis next to the dataset.
2. Click **Schedule Refresh**.
3. Expand **Featured Q&A Questions**.
4. Click **Add A Question**.
5. Type your query in the box. In our example, we can type **top stock item by quantity**.
6. Click **Apply**.

Now your question will appear alongside the Q&A terms.

***MORE INFO* FEATURED QUESTIONS FOR Q&A**

For more details on how you can add featured questions, including a video tutorial, see "Create featured questions for Power BI Q&A" at *https://docs.microsoft.com/en-us/power-bi/service-q-and-a-create-featured-questions*.

You can see the Q&A area with a featured question in Figure 3.13.

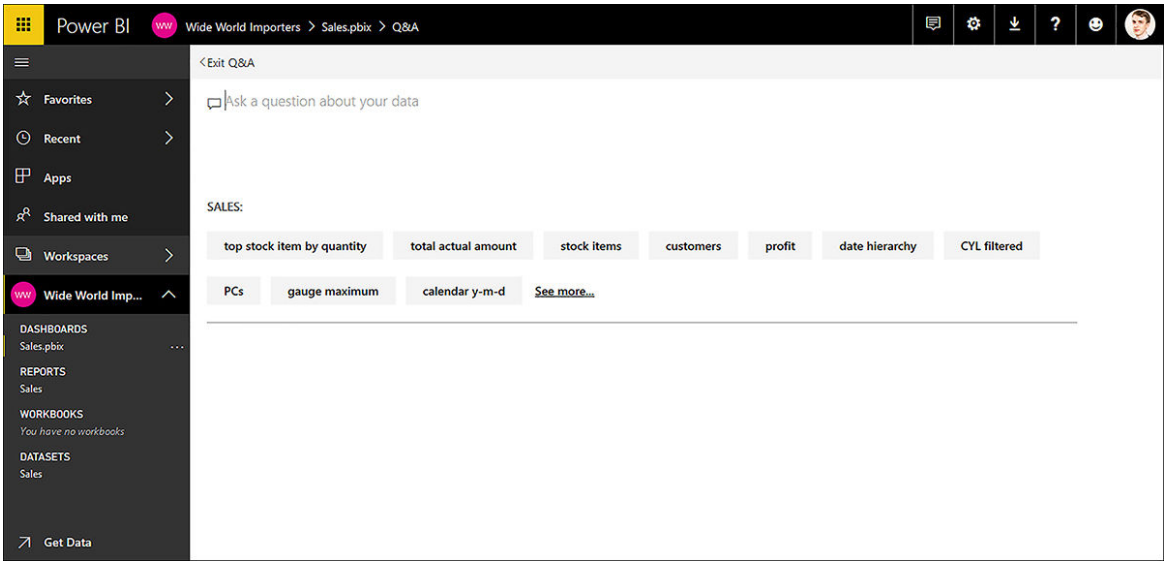


FIGURE 3.13 Q&A terms

As soon as you type **total profit**, you will get a card visual. Once you add **by calendar year label**, you will see a bar chart. Finally, typing **as column chart** will transform the bar chart into a column chart. You can see the results of the above query in Figure 3.14.

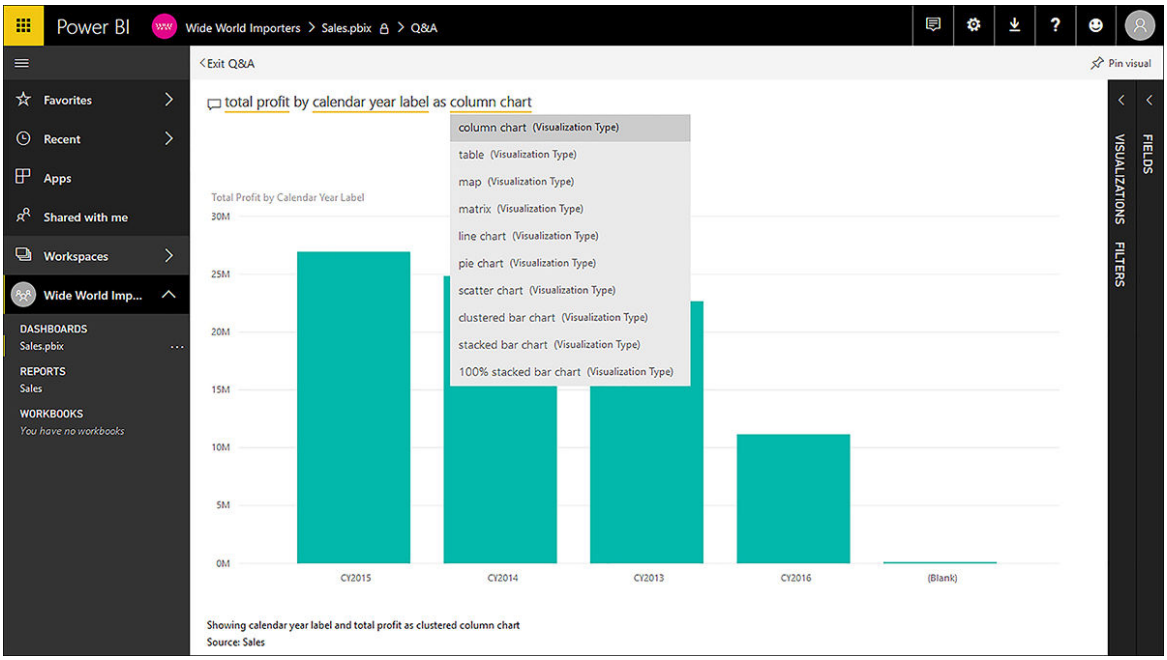


FIGURE 3.14 Q&A in action

Each keyword or set of keywords will be underlined and can be clicked on to see the alternatives. For example, instead of **column chart,** choose **pie chart**. If needed, you can expand the Visualizations and Fields panes for more granular control over the visual.

When you ask questions of your data, Power BI relies on built-in heuristics to answer them. For instance, “color” and “colors” are going to be deemed equivalent even if a column is named “Color” in your data model.

> ***MORE INFO* Q&A TERMINOLOGY**
>
> If you see a grayed-out keyword, it means that Power BI does not recognize it. In cases like this, you can try phrasing your question differently. For more details on the words and terminology that Q&A understands, see “Tips for asking questions in Power BI Q&A” at *https://docs.microsoft.com/en-us/power-bi/service-q-and-a-tips*.

Once you are satisfied with the visual created using the Q&A feature, you can pin it to your dashboard clicking the **Pin Visual** button in the top-right corner.

Q&A in Power BI Desktop

Q&A is also available in Power BI Desktop, and it works in a similar way to Power BI service. Depending on the version of Power BI Desktop you are using,

you may need to enable Q&A in preview features by clicking **File** > **Options And settings** > **Options** > **Preview Features** > **Q&A**. Once you make sure that Q&A is available, you can click **Home** > **Insert** > **Ask A Question**. Alternatively, you can double-click on empty space in a canvas, which will create a placeholder visual with a question box above it in which you can type. You can see the Q&A visual in Figure 3.15.

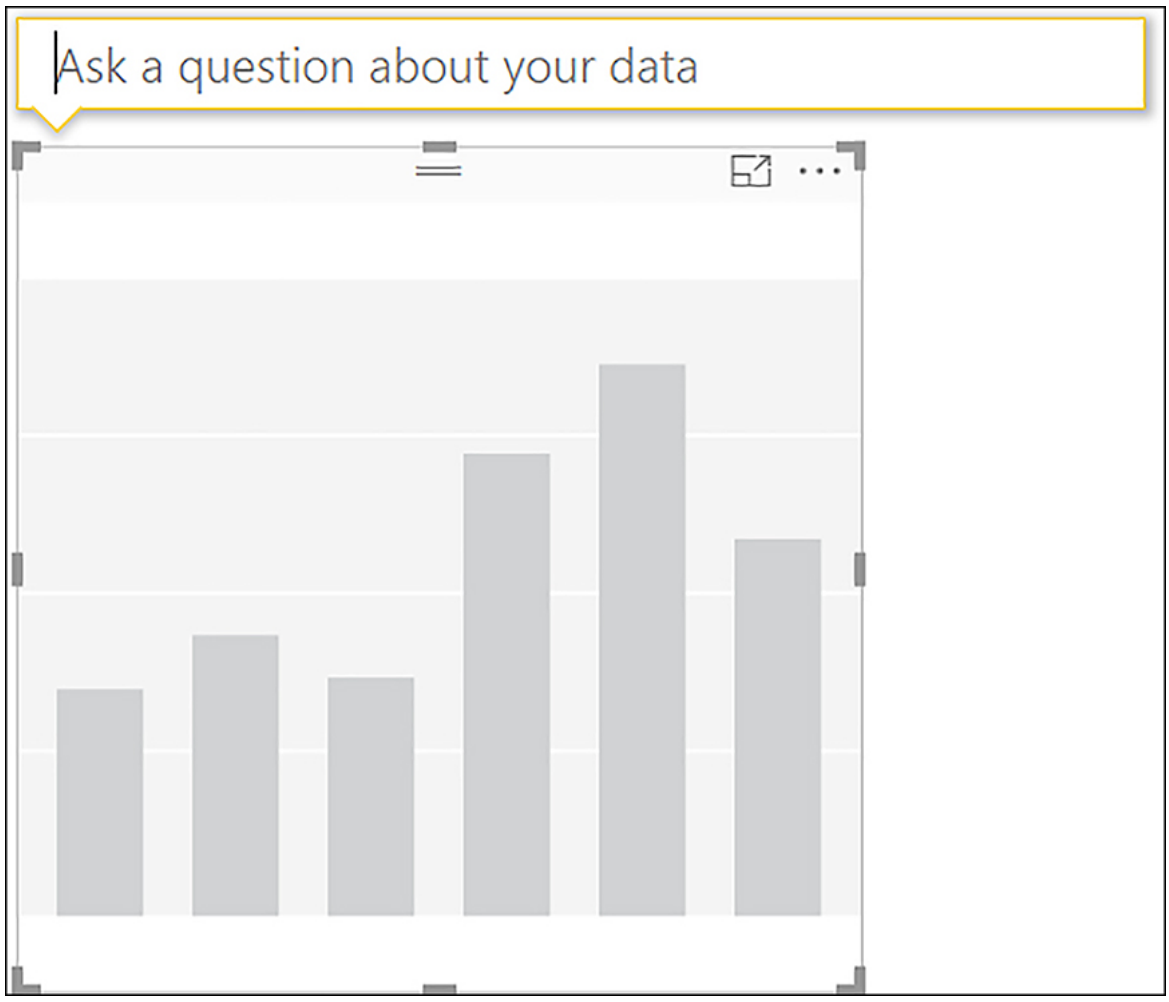


FIGURE 3.15 Ask a Question

After you type your question, you can refine the visual by formatting it or choosing a different visual type. Once you click away, you will not be able to modify your question; if you need to ask a different question, you will need to start from the beginning.

MORE INFO **Q&A IN POWER BI**

For a general overview of Q&A in Power BI service and Power BI Desktop, including details on how Q&A answers questions, see “Q&A in Power BI service and Power BI Desktop” at *https://docs.microsoft.com/en-us/power-bi/power-bi-q-and-a*.

If you want to follow a video tutorial on how to use Q&A, see “Tutorial: How to use Q&A to create visualizations and build reports” at *https://docs.microsoft.com/en-us/power-bi/power-bi-tutorial-q-and-a*.

Synonyms

You can introduce your own keywords and make Power BI recognize them. For example, if business users often substitute *pieces* for *quantity*, you can create a synonym for this. To do that, you need to follow the next steps:

1. Open your report in Power BI Desktop.
2. Go to the **Relationships** view.
3. Select **Modeling** > **Synonyms**.
4. Select the table that contains the field for which you want to create a synonym. In our example, select the **Sale** table.
5. In the Synonyms pane, scroll down to the **Quantity** field.
6. Click on the input field; a comma will be automatically inserted after **quantity**.
7. Type **pieces**.
8. Save your changes.
9. Re-publish your report.

You can see the Synonyms pane with an added keyword in Figure 3.16.

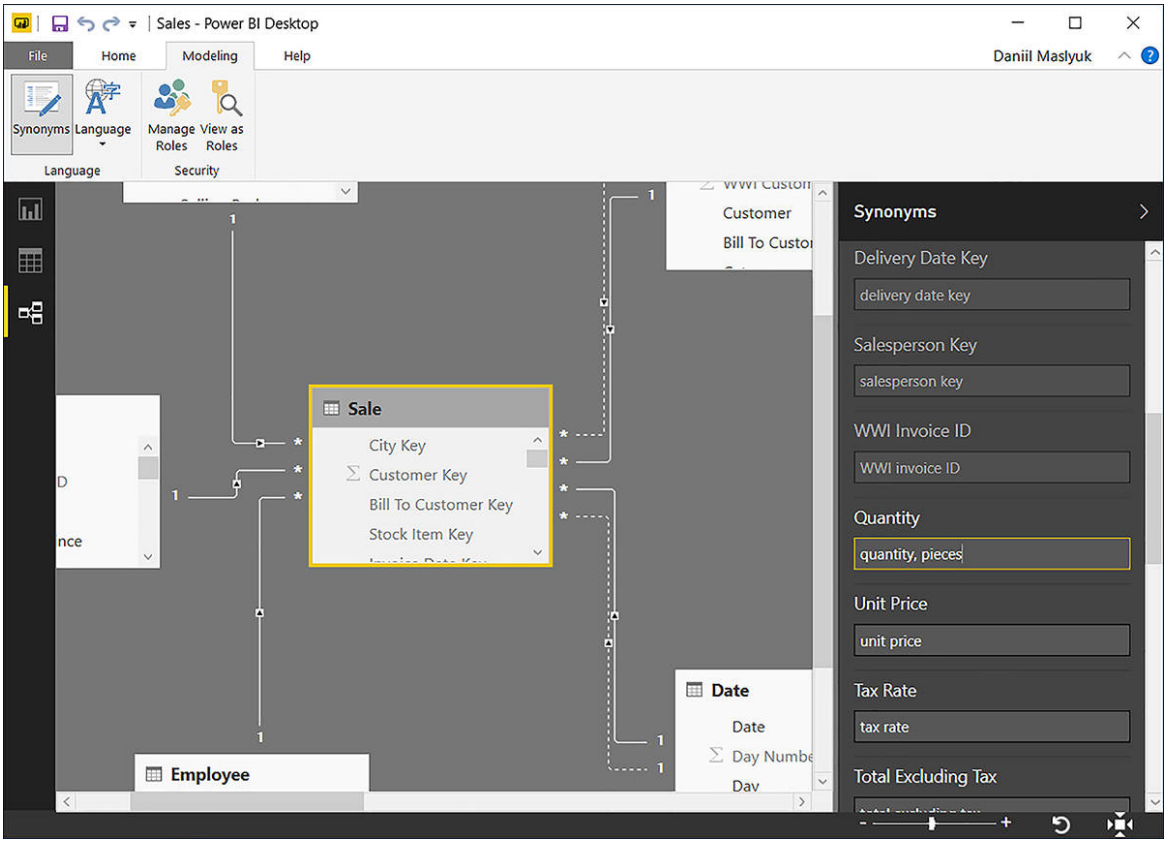


Figure 3.16 Synonyms

You can now use the **pieces** keyword. For instance, the following natural language query will create a bar chart that shows top three colors by quantity sold:

```
top 3 colors by pieces
```

Even though you typed **pieces**, you still see **Quantity** in the chart, which can be seen in Figure 3.17.

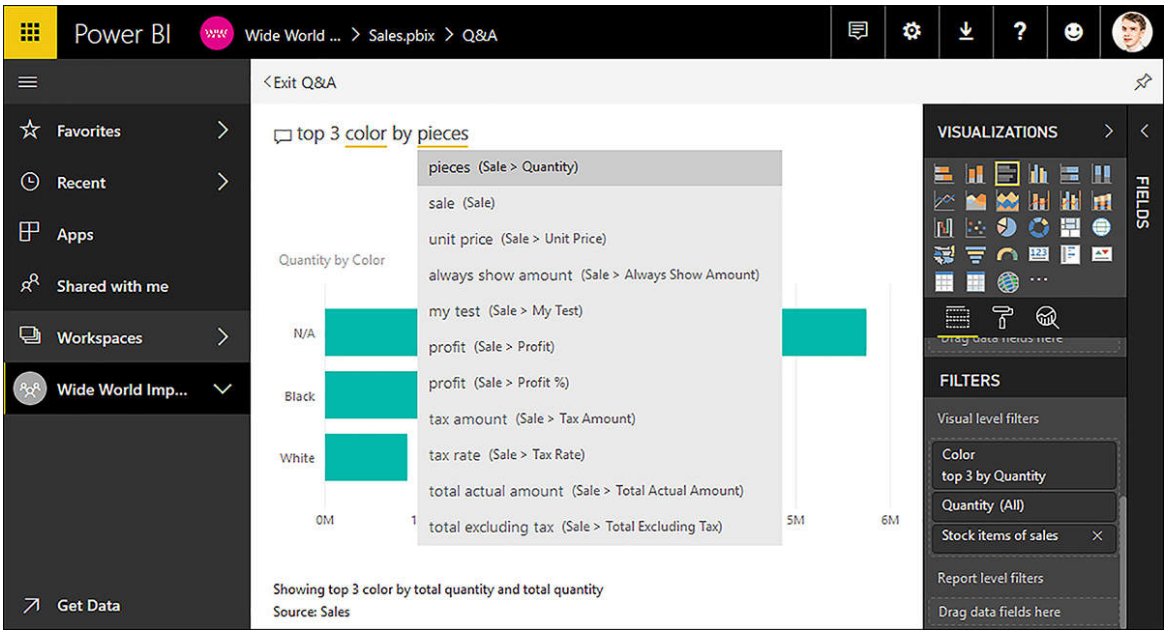


Figure 3.17 Using a synonym in a query

Note that one of the visual level filters is "top 3 by Quantity." Q&A can automatically filter data if the query asks for it.

MORE INFO USING SYNONYMS

You can use synonyms to simulate a translation of your data model. For example, if you have colleagues in France who are going to use your data model, you can create synonyms for each field in French, which will make it easier for your colleagues to write natural language queries. For more details on the approach and using synonyms in general, see "Power BI Synonyms, Take Q&A Experience to the Next Level" by Soheil Bakhsi at http://biinsight.com/power-bi-synonyms-take-qa-experience-to-the-next-level/.

MORE INFO Q&A IN POWER BI SERVICE AND CORTANA

Cortana is a virtual assistant created by Microsoft. It can answer your questions, including questions regarding your data in Power BI service. It can also display custom answer pages specifically designed for Cortana. Enabling Cortana to access Power BI datasets is outside the scope of this book; for more information on making Cortana and Power BI work together, including technical requirements, see "Quickly find and view your Power BI data using Cortana for Power BI" at *https://docs.microsoft.com/en-us/power-bi/service-cortana-intro*.

Skill 3.3: Publish and embed reports

There are many ways to share your Power BI report with others. In this section, we are going to review the steps to make a report publicly available to anyone and how to publish a report to SharePoint and Power BI Report Server.

This section covers how to:

- Publish to web
- Publish to Microsoft SharePoint
- Publish reports to a Power BI Report Server

Publish to web

You can make your Power BI report public using the Publish to Web feature in Power BI service. This option gives you an HTML code that you can use to embed a report in a blog post, on a web page, or elsewhere.

To publish your report to the web, you need to navigate to your report in Power BI service and click **File** > **Publish To Web**. You will then see the Embed In A Public Website window, where you will need to click **Create Embed Code** and then **Publish**. Doing so will take you to the Success window shown in Figure 3.18; from here, you can copy the link for sharing the report as well as the HTML code to paste into your blog or website.

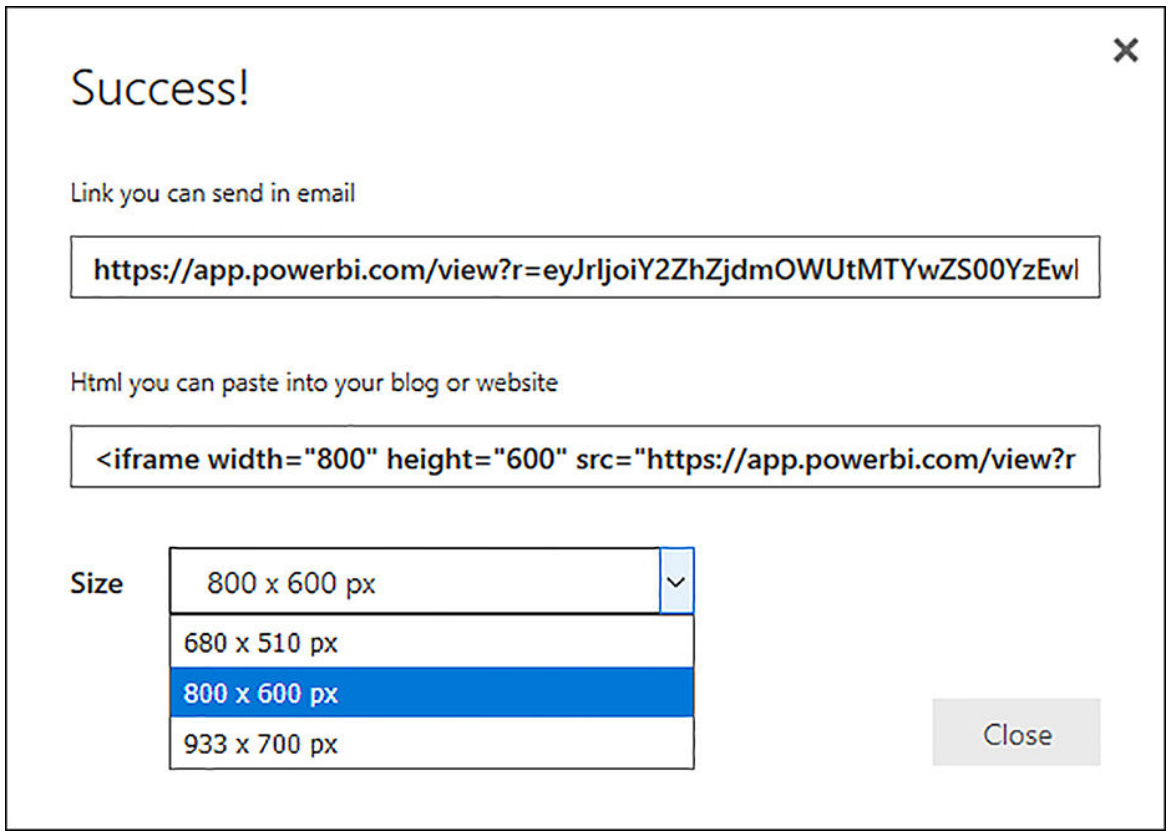


FIGURE 3.18 Publishing to the web

When publishing, you can choose from one of the three pre-defined sizes:

- 680 x 510 px
- 800 x 600 px
- 933 x 700 px

If needed, you can manually adjust the report size in the HTML code. You can also adjust the view mode in Power BI service by clicking **View** and then clicking one of the options below. Alternatively, you can adjust view mode in Power BI Desktop by clicking **View** tab > **View** grouping > **Page View** and then one of the three options:

- **Fit To Page** This option fits a report page into the window. Because the window may also contain extraneous elements such as a navigation bar, and the aspect ratio of the report and the window may be different, this option may result in grey bars shown outside of report content to fill the empty space in the window.
- **Fit To Width** This option only fits the width of a report page to the screen, which may result in a vertical scrollbar. The original aspect ratio is preserved.

- **Actual Size** This option makes sure that the size of the report is preserved from the report settings. This might result in scrollbars.

Your report will contain a navigation bar that is 56 px tall. You can adjust the frame dimension size by this figure for a better fit. For example, if you wanted to embed a report that is 640 by 480 px, you could change that to 640 by 536 px.

> ***IMPORTANT PUBLISH TO WEB SECURITY***
>
> There is absolutely no security when you use the Publish To Web feature. There is no authentication when viewing the report, and anyone on the web who has a link, including people who find your report through search engines, can access your published report. You should only use Publish To the Web when the underlying data, including the lowest detail level, is not sensitive.

If you have previously published a report to the web, you will see the same window as shown in Figure 3.18, except it will be titled Embed Code. Only one embed code can be created per report. Unlike pinned visuals, reports that are published to web are updated after you make changes to them, although it may take up to an hour for updates to propagate.

Manage embed codes

All embed codes that you created previously can be managed in Power BI service by clicking **Settings**, **Manage Embed Codes**. You can see the Manage Embed Codes page in Figure 3.19.

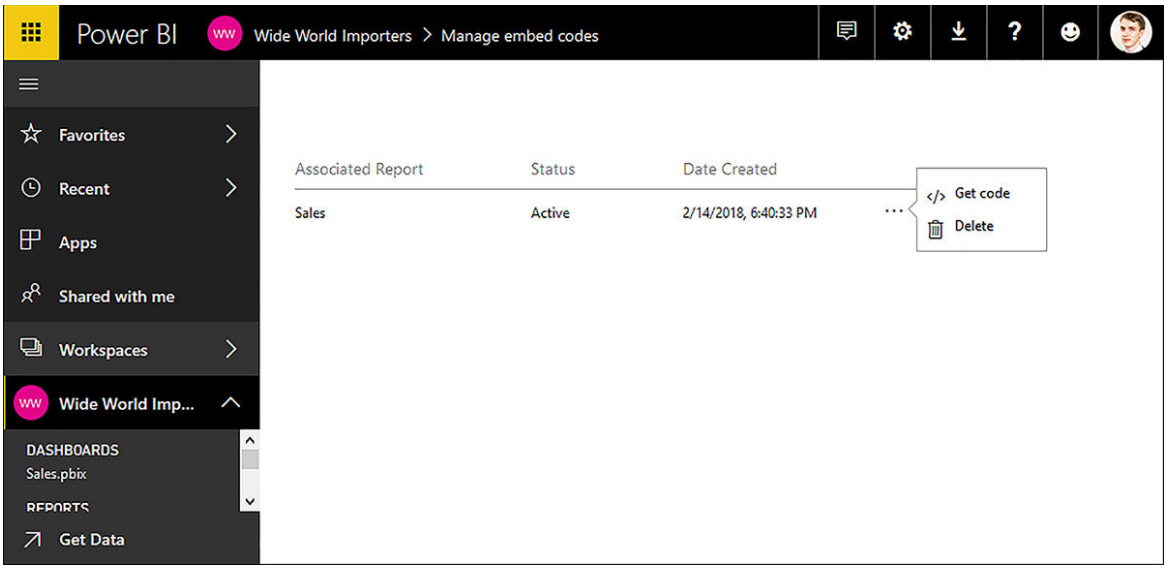


Figure 3.19 Manage embed codes

On the Manage embed codes page, you can see a list of reports for which you created embed codes, along with their status and the date when you created a link. The Status column displays one of the following statuses:

- **Active** The report is available for anyone on the web.
- **Blocked** The report has been blocked by Microsoft; if you believe this was an error, you can contact support.
- **Not Supported** The report uses one of the unsupported features discussed below.
- **Infringed** The tenant policy regarding **Publish To Web** was changed after the report had been published.

You can also copy the embed code by clicking on the ellipsis and then selecting **Get Code**. If you need to delete a code, you can click **Delete** after clicking on the ellipsis.

Limitations

Not all reports can be published to the web. The following are some of the unsupported scenarios when publishing to web:

- R visuals
- Reports with row-level security
- Reports that connect live to on-premises Analysis Services Tabular
- Reports shared with you

Also, a Power BI administrator can disable publishing to web for specific users or a group of users. If you are a Power BI administrator, you can click **Settings** > **Admin Portal** > **Tenant settings**, and expand **Publish To Web**; from here, you can disable the feature for the entire organization or for specific security groups.

> ***MORE INFO*** **PUBLISH TO WEB**
>
> For more information on the Publish to Web feature, including a video overview, security considerations, and more details on limitations, see "Publish to web from Power BI" at *https://docs.microsoft.com/en-us/power-bi/service-publish-to-web*.

Publish to Microsoft SharePoint

You can securely embed interactive Power BI reports in SharePoint Online. This option is drastically different from **Publish To Web** because it is fully secure. To embed Power BI content in SharePoint Online, you need Modern Pages.

> ***MORE INFO*** **MODERN PAGES**
>
> Modern Pages facilitate creation of content in SharePoint Online and have been available since 2016. For more information on Modern Pages, see "Customizing 'modern' site pages" at *https://docs.microsoft.com/en-us/sharepoint/dev/solution-guidance/modern-experience-customizations-customize-pages*

To embed a Power BI report in a SharePoint page, you need to get a report link and use it in a web part in SharePoint Online. To get a report link, you can navigate to a report and click **File** > **Embed In SharePoint Online**, which will open the **Embed Link For SharePoint** window, from which you can copy the link as shown in Figure 3.20.

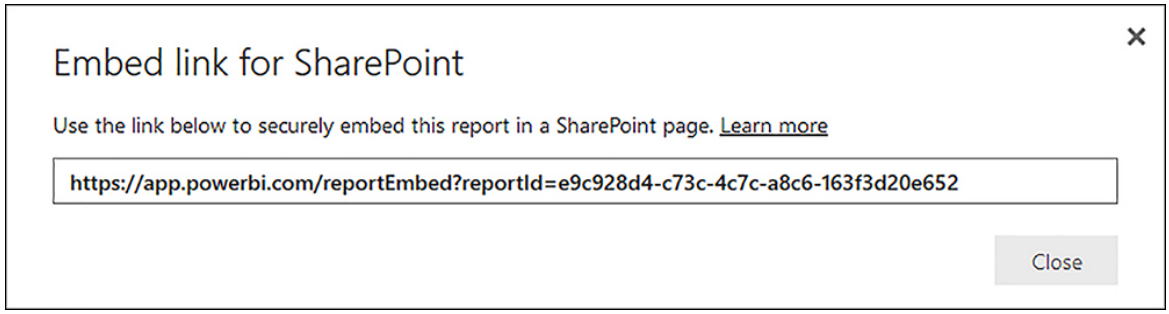


FIGURE 3.20 Embed link for SharePoint

Alternatively, you can copy the report URL from the navigation bar of your browser.

> ***IMPORTANT REPORT URL***
>
> When you copy the report URL from the navigation bar of your browser, it might contain a reference to a specific page of your report. If you want to embed only the page you selected, you can leave it as-is; otherwise, you need to remove the portion that starts with "/ReportSection." Or you can copy the link from the Embed Link For SharePoint window as described above.

Report embedding

Once you have the link, you can embed it in SharePoint online. To do so, you need to create a web part in a page. In the following steps, we are going to create a site page from scratch:

1. Navigate to your SharePoint Online site.
2. Click **New** > **Site Page.**
3. Click on the **Name Your Page** section and give your page a name.
4. Click the + icon (when you hover over it, a screentip reading **Add A New Web Part In Column One** when you hover over it.
5. Click **Power BI**. You can use the search bar to locate the button.
6. Click **Add Report.**
7. Paste your report link.
8. Select the page name from the **Page Name** drop-down list.
9. Click **Publish.**

Your site page will then be published, and SharePoint Online will provide you with a page link. You can edit your page by clicking **Edit** in the top-right corner of the page. If you want to edit the Power BI web part, you need to click on the Edit web part button that has a pencil icon and is to the left of the report page. In addition to the report link and report page name, you can change the following settings:

- **Display** You can choose either 4:3 or 16:9 ratio for your Power BI web part. Note that you cannot set custom size in pixels. The default selection is **16:9**.
- **Show Navigation Pane** You can either show or hide the navigation pane that contains page names and appears at the bottom. The default setting is **On**.

- **Show Filter Pane** You can either show or hide the filter pane that appears on the right. The users will not be able to add new fields to the pane, but they will be able to change the filters that are already in place, as well as remove them. The default setting is **Off**.

Report security

If a user has access to a SharePoint Online site page that contains a Power BI report, the user does not automatically have access to the report. If you do not have access to a report, the report will not load, and you might see the message shown in Figure 3.21.

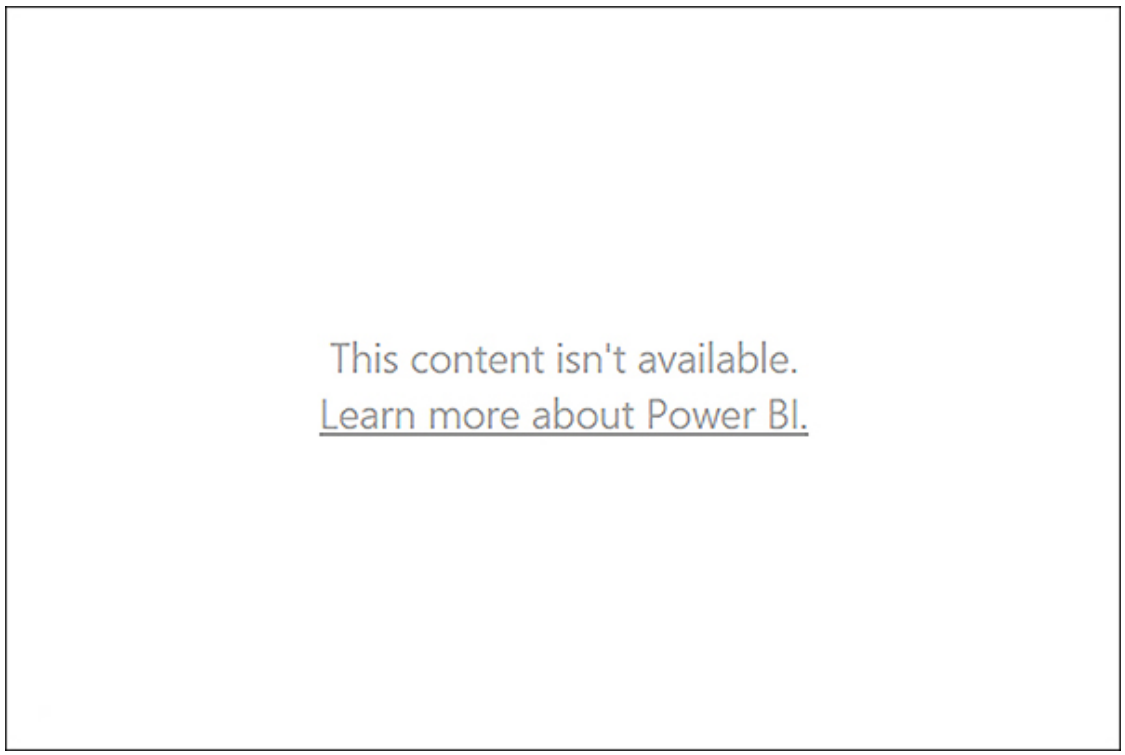


FIGURE 3.21 Inaccessible report

> ***NOTE*** **REPORT NOT LOADING**
>
> Also, a report might not load in SharePoint Online because it has been deleted after embedding.

Report security is managed in Power BI service. There are two ways to grant a user access to an embedded report. First, if you use an Office 365 group for your SharePoint Online team site, you can add a user to the app workspace of the report; doing so will grant the user access to all contents of the workspace.

Second, you can share a report directly by navigating to the report and clicking **Share** in the top-right corner.

> ***MORE INFO*** **SECURING POWER BI CONTENT**
>
> Configuring security for your dashboards and reports will be covered in more detail later in the chapter in Skill 3.4: Configure security for dashboards, reports, and apps.

> ***MORE INFO*** **SHAREPOINT ONLINE REPORT EMBEDDING**
>
> For more details on embedding a report in SharePoint Online, including considerations on multi-factor authentication and limitations, see "Embed with report web part in SharePoint Online" at *https://docs.microsoft.com/en-us/power-bi/service-embed-report-spo*.

Publish reports to a Power BI Report Server

So far in this chapter, we considered publishing reports in the cloud only. With Power BI Report Server, you have an option to publish your report on-premises. Power BI Report Server is included with Power BI Premium. It is also included with the purchase of SQL Server Enterprise Edition with Software Assurance.

> ***NOTE*** **CONFIGURING POWER BI REPORT SERVER**
>
> This section assumes that you already have Power BI Report Server installed. Power BI Report Server deployment and configuration are outside of the scope of this book. For a general overview of Power BI Report Server, see "Get started with Power BI Report Server" at *https://docs.microsoft.com/en-us/power-bi/report-server/get-started*.
>
> For details on how to install Power BI Report Server, see "Quickstart: Install Power BI Report Server" at *https://docs.microsoft.com/en-us/power-bi/report-server/quickstart-install-report-server*.

If you want to publish your report to Power BI Report Server, it is recommended that you install the Power BI Desktop version that is optimized for Power BI Report Server. This is because Power BI Report Server features are released on a different schedule from Power BI service and the regular Power BI Desktop, and not all features from the regular Power BI Desktop version are supported by Power BI Report Server, even if the two are released at the same

time. You can have both versions installed on the same machine; by default, the version that was installed last will be the default app to open Power BI files.

To install Power BI Desktop optimized for Power BI Report Server, navigate to the Power BI Report Server web portal and click **Download** > **Power BI Desktop**. Once you install the software, you can author your reports. The report building experience is almost identical to the regular version of Power BI Desktop. The only difference is that some features might not be supported.

> ***MORE INFO*** **TWO VERSIONS OF POWER BI DESKTOP**
>
> There are some differences between the two versions of Power BI Desktop, such as the list of supported features, release schedule, and publishing options. For a comprehensive comparison of Power BI Desktop and Power BI Desktop optimized for Report Server, see this article by Samuel Lester, "Understanding the differences between 'Power BI Desktop' and 'Power BI Desktop Optimized for Power BI Report Server'" at *https://blogs.msdn.microsoft.com/samlester/2018/02/19/understanding-the-differences-between-power-bi-desktop-and-power-bi-desktop-optimized-for-power-bi-report-server/*.

You can also open an existing report that has a .pbix extension; if you open a report that was created in the regular version of Power BI Desktop, you might see the message shown in Figure 3.9. If, instead, you use a regular version of Power BI Desktop to open a report that was saved in Power BI Desktop optimized for Power BI Report Server, you might see the message shown in Figure 3.22.

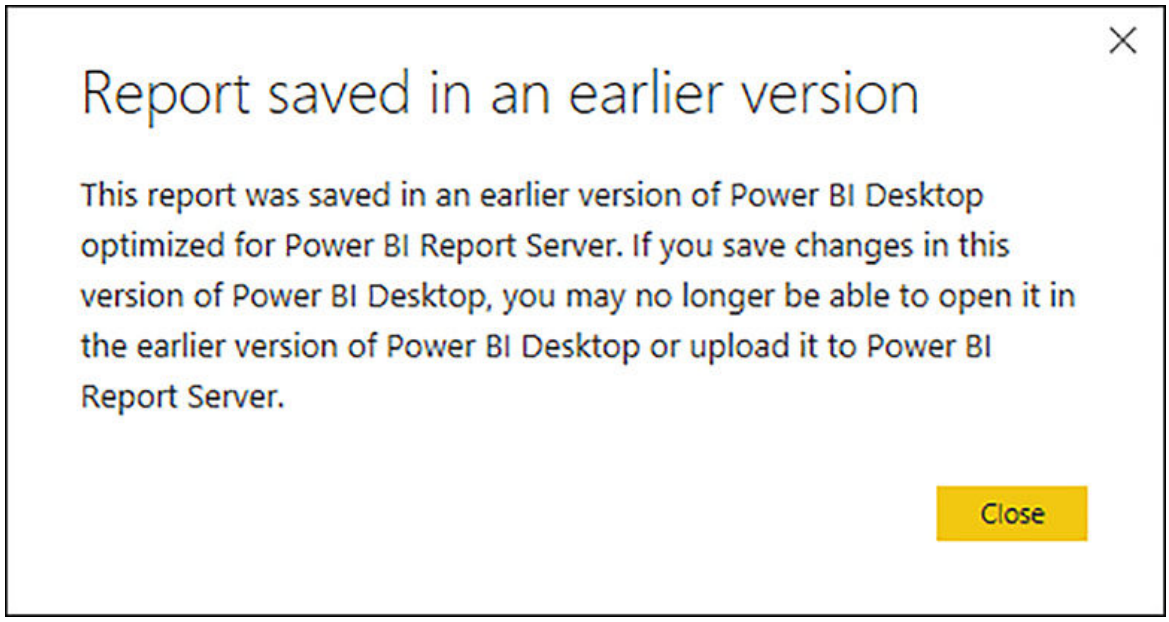


FIGURE 3.22 Report saved in an earlier version

The major difference between the regular version of Power BI Desktop and Power BI Desktop optimized for Power BI Report Server is the way you publish reports. Power BI Desktop optimized for Power BI Report Server does not have the Publish button. Instead, to publish a report to Power BI Report Server, you need to save your report to the server.

To publish a report to Power BI Report Server from Power BI Desktop optimized for Power BI Report Server, you need to click **File** > **Save as** > **Power BI Report Server**. You will then see the Power BI Report Server Selection window shown in Figure 3.23.

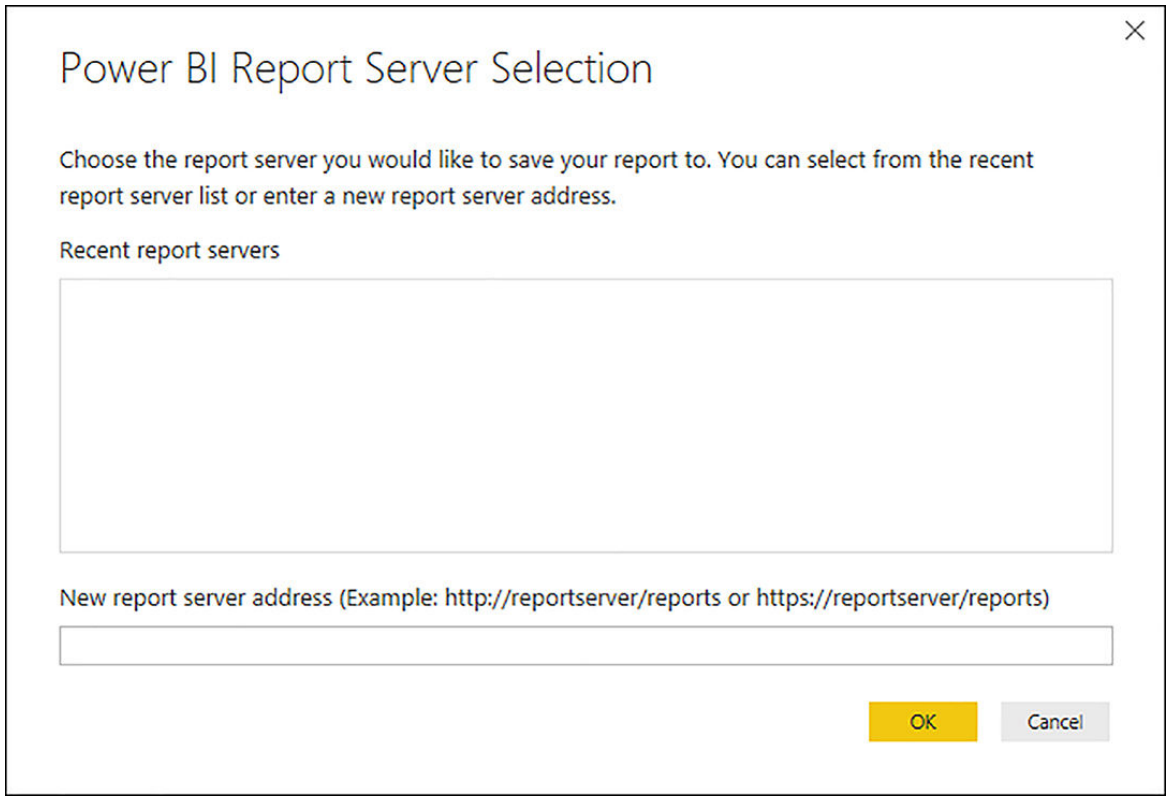


Figure 3.23 Power BI Report Server Selection

Clicking **OK** takes you to the Save report window, where you can choose a folder to which you want to save your report. In the next example, you will save a report called Sales.pbix to the Wide World Importers folder that has already been created in the report server located at *http://localhost/Reports/*. You can see the **Save Report** window in Figure 3.24.

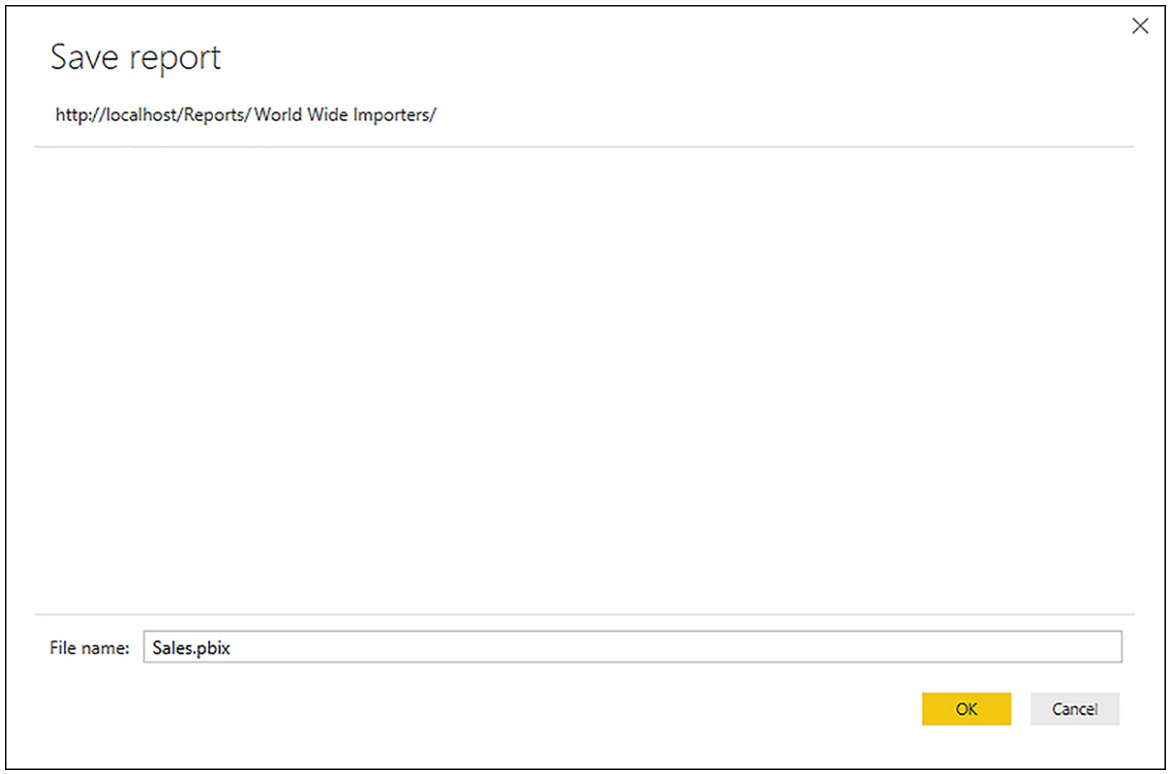


FIGURE 3.24 Save report window

Once you choose a folder and click **OK** in the **Save Report** window, the report will be saved, and you will see the **Saving To Power BI Report Server** window shown in Figure 3.25.

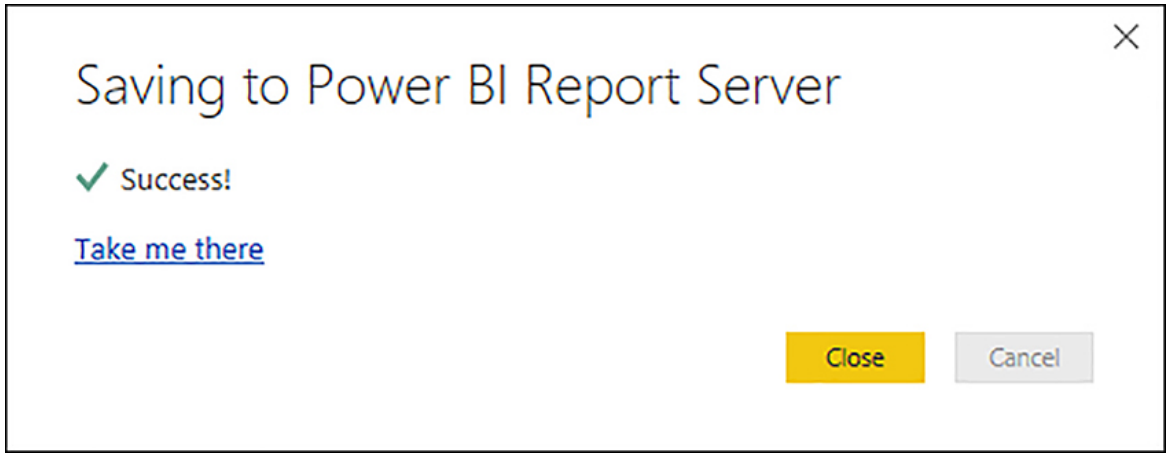


FIGURE 3.25 Saving to Power BI Report Server

Clicking on the **Take Me There** hyperlink takes you to the report in Power BI Report Server. You can see the report as it appears in Power BI Report Server in Figure 3.26.

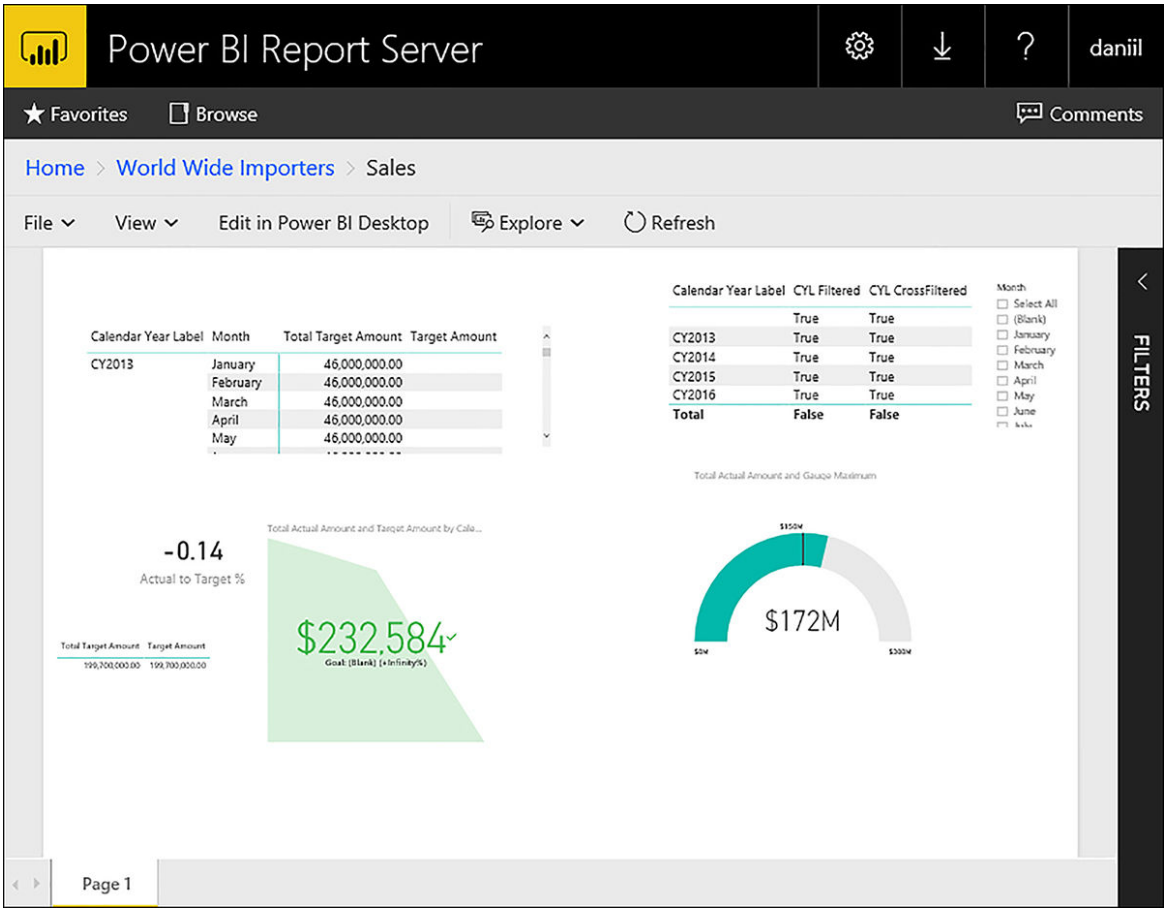


FIGURE 3.26 Sales report saved to Power BI Report Server

Editing existing reports

One of the differences between Power BI Report Server and Power BI service is that the latter provides an online report-editing experience that is similar to Power BI Desktop, while the former does not. Instead, you can edit your reports in Power BI Desktop.

If you need to make changes to an existing report, you can open the report in Power BI Desktop optimized for Power BI Report Server. There are three ways to open an existing report:

- **From the report page in Power BI Report Server** For this, you need to click **Edit In Power BI Desktop**, which will open the report in Power BI Desktop optimized for Power BI Report Server.
- **From folder in Power BI Report Server** When you are in a Power BI Report Server folder, you can click on the ellipsis next to a report and click **Edit In Power BI Desktop**, which will open Power BI Desktop optimized for Report Server. You can see the menu in Figure 3.27.

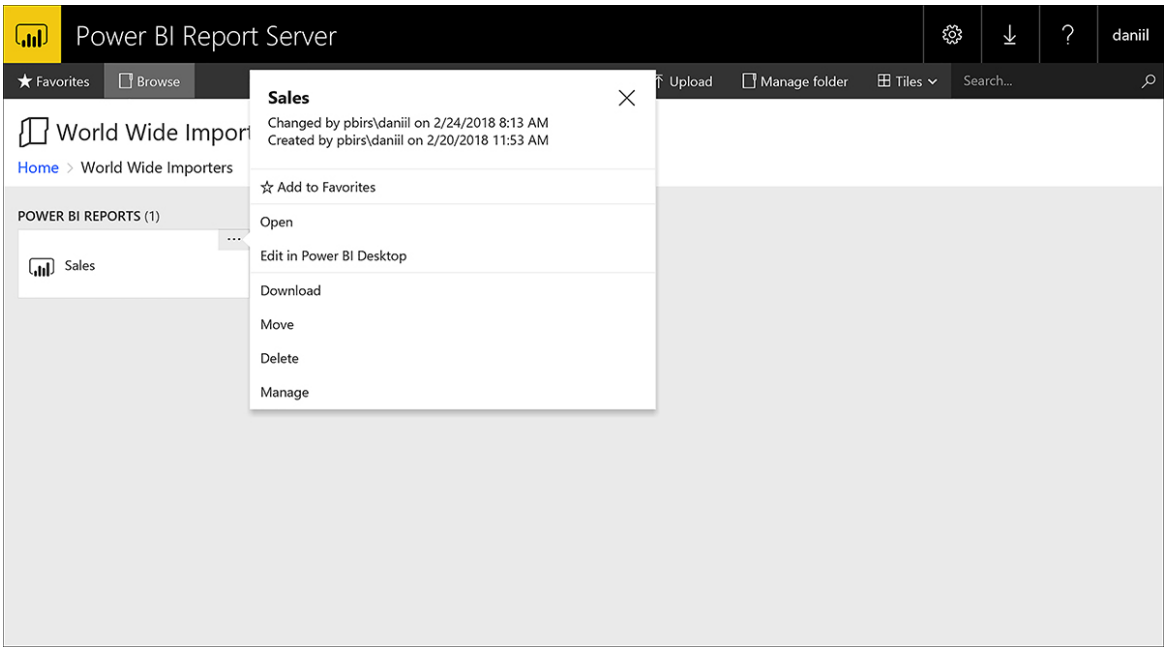


FIGURE 3.27 Power BI Report Server folder report menu

- **From Power BI Desktop** In Power BI Desktop optimized for Power BI Report Server, you can click **File** >**Open** > **Power BI Report Server**. You will then see the Power BI Report Server Selection window shown in Figure 3.23, followed by the **Open Report** window that is similar to the Save report window shown in Figure 3.24. In the Open report window, you will need to select the report you want to edit.

Once you make the desired changes and click **File** > **Save**, you need to wait for the **Saved successfully** message to appear in the bottom right corner of the main window. If you close Power BI Desktop before the message appears, the changes will not be saved.

Commenting reports

Power BI Report Server allows you to add comments to existing reports. When you are on a report page in Power BI Report Server, click **Comments** in the top-right corner to open the Comments pane.

In the Comments pane, you can type your comment in a text box, and you have an option to attach a file to your comment. Once you have posted your comment, you can edit its text, and others can reply to it. Comments can be sorted from newest to oldest and vice versa. You can see a comment and a reply to it as shown in Figure 3.28.

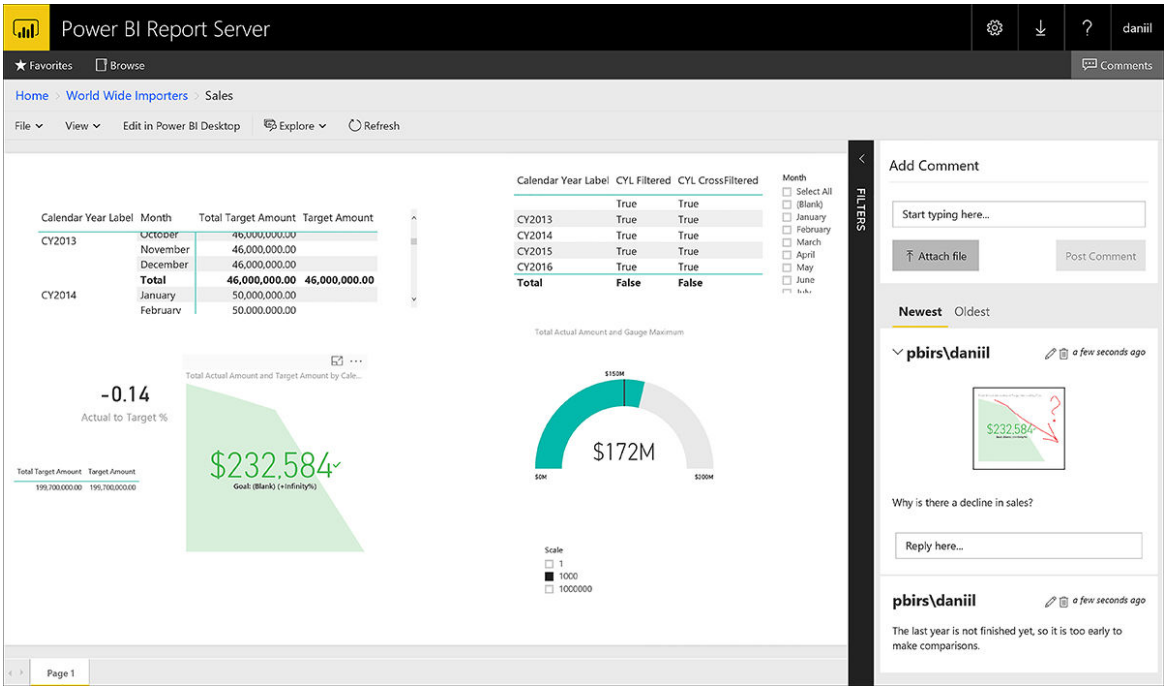


FIGURE 3.28 Comments in Power BI Report Server

In replies, you cannot include attachments. Comments and replies can be deleted if needed by clicking on the **Delete** button that is depicted by a trash can icon.

MORE INFO POWER BI REPORT SERVER REPORT COMMENTS

For more information on the commenting system in Power BI Report Server, including permissions, see "Add comments to a report in a report server" at *https://docs.microsoft.com/en-us/power-bi/report-server/add-comments*.

MORE INFO POWER BI REPORT SERVER REPORTS

For a detailed overview of how you can create reports for Power BI Report Server, see "Quickstart: Create a Power BI report for Power BI Report Server" at *https://docs.microsoft.com/en-us/power-bi/report-server/quickstart-create-powerbi-report*.

Skill 3.4: Configure security for dashboards, reports, and apps

When sharing Power BI content, you often need to configure security so that each user has the right level of access. For instance, you might want the regional managers to see only values from their own regions, while allowing the general manager to see all regions. In this section, we are going to review the steps you need to take to create a security group, configure access to Power BI content such as dashboards and app workspaces, and review the Power BI tenant security settings. Finally, we are going to review Row-Level Security in Power BI.

This section covers how to:

- Create a security group by using the Admin Portal
- Configure access to dashboards and app workspaces
- Configure the export and sharing settings of the tenant
- Configure Row-Level Security

Create a security group by using the Admin Portal

When you share content in Power BI, you can use the email address for each user. While this works, this solution can be hard to maintain: for example, if you share several reports with many salespeople and a new salesperson joins, you will need to update the sharing settings for each report. In cases like this, you can create a security group, add all salespeople to it and share your reports with the security group. When a new salesperson joins the company, you will only need to add them to the security group once, and all reports will be immediately available to them.

Security groups are created in Office 365 Admin Portal, also known as Admin center. To create a security group, follow these steps:

1. Sign in at *https://www.office.com/*.
2. In the Apps section, click **Admin**. You need to be an Office 365 administrator to access this app.
3. Click the **Expand Navigation Menu** button on the left, which has a right arrow icon.
4. Click **Groups** > **Groups**. At this point, you will see a page similar to Figure 3.29.

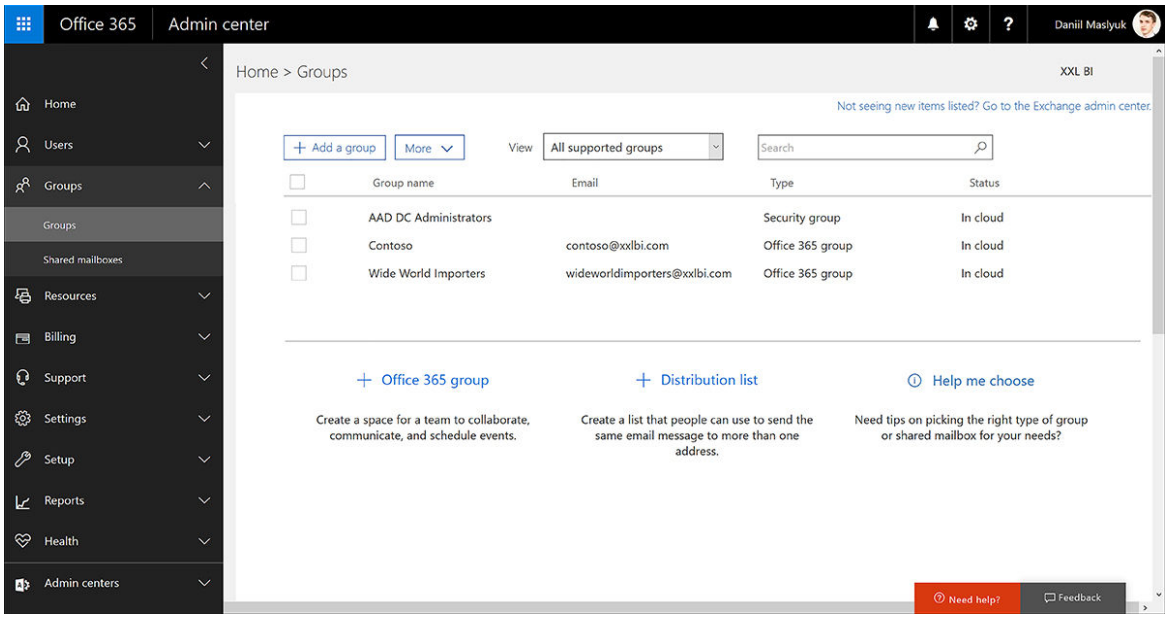


Figure 3.29 Admin center, Groups

5. Click **Add A Group**.
6. In the **Type** drop-down list, select **Security Group**.
7. Type a group name and, optionally, a description. For example, you can call the group **My Security Group**.
8. Click **Add** > **Close**.

Your new security group will now appear in the list of groups. You can edit a group by clicking on its row in the list of groups, which will open a settings pane where you can do the following:

- Delete group
- Edit name and description
- Add owners
- Add members

The settings pane can be seen in Figure 3.30.

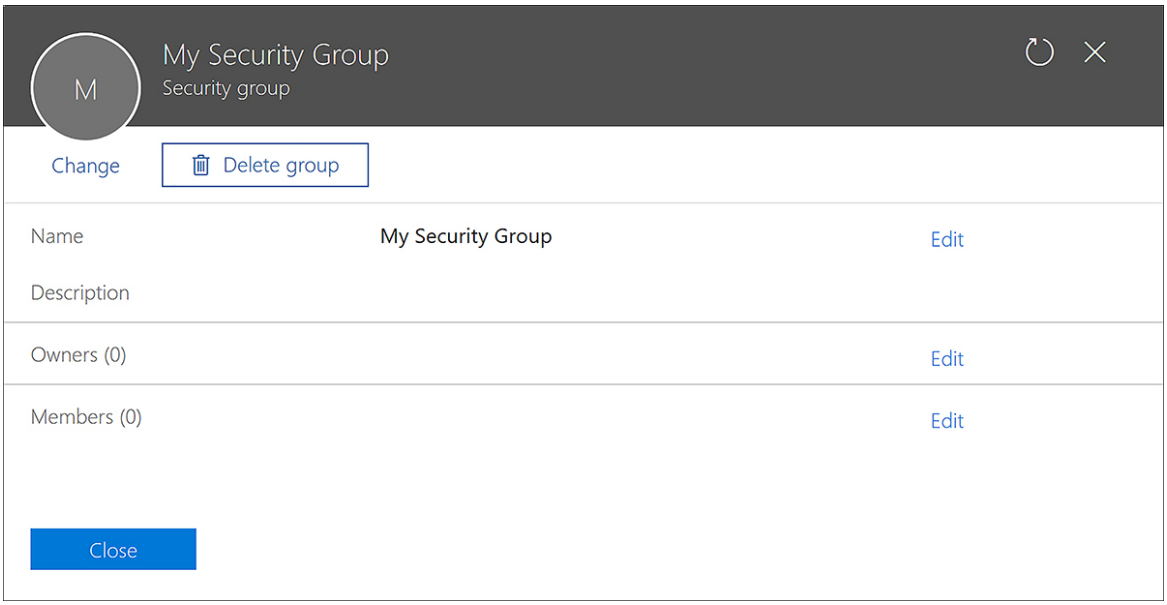


Figure 3.30 Group settings

To add a member to the group, you need to follow the next steps:

- Click **Edit** next to Members.
- Click **Add Members**.
- Search for members and pick them from the list by selecting the checkboxe next to the members you want to add.
- Click **Save** > **Close** > **Close**.

NOTE **SECURITY GROUP MEMBERSHIP**

Unlike distribution lists and mail-enabled security groups, security groups do not require its members to have mailboxes allocated to them. What this means is that you can have a user who has a Power BI license allocated to her and who is a member of a security group, but does not have a mailbox. This can save costs in case a user needs access to Power BI but does not need a working company email.

Distribution lists and mail-enabled security groups are outside of the scope of this book. For a comprehensive comparison of different group types available in Office 365, see "Compare groups" at *https://support.office.com/en-us/article/compare-groups-758759ad-63ee-4ea9-90a3-39f941897b7d*.

When you create an app workspace, an Office 365 group is created automatically. App workspaces are covered later in this chapter in Skill 3.5:

Configure apps and apps workspaces.

> ***MORE INFO*** **MANAGING SECURITY GROUPS IN OFFICE 365**
>
> For more information on working with security groups in Office 365, see "Create, edit, or delete a security group in the Office 365 admin center" at *https://support.office.com/en-us/article/create-edit-or-delete-a-security-group-in-the-office-365-admin-center-55c96b32-e086-4c9e-948b-a018b44510cb*.

Configure access to dashboards and app workspaces

Power BI content can be contained either in your own workspace, called My Workspace, or in app workspaces. You can share reports and dashboards individually, as well as grant other users access to app workspaces.

In all cases, sharing Power BI content requires a Power BI Pro license for the report developer and either Power BI Pro licenses for each report user or Power BI Premium capacity.

> ***MORE INFO*** **COLLABORATING IN POWER BI**
>
> For more information on how you can collaborate in Power BI, including ways other than sharing dashboards and reports and giving access to app workspaces, see "How should I collaborate and share in Power BI?" at *https://docs.microsoft.com/en-us/power-bi/service-how-to-collaborate-distribute-dashboards-reports*.

Sharing dashboards and reports

When you share a dashboard with a user, the user can see the dashboard as well as click through to any reports to which the dashboard tiles point. Sharing a report gives a user access to that report only, but does not provide access to the dashboards that use visuals from the report. In both cases, users will have read-only access.

To share a dashboard with a user, follow these steps:

1. Navigate to the dashboard you want to share and click **Share** in the top-right corner. You will see the Share Dashboard pane like in Figure 3.31.

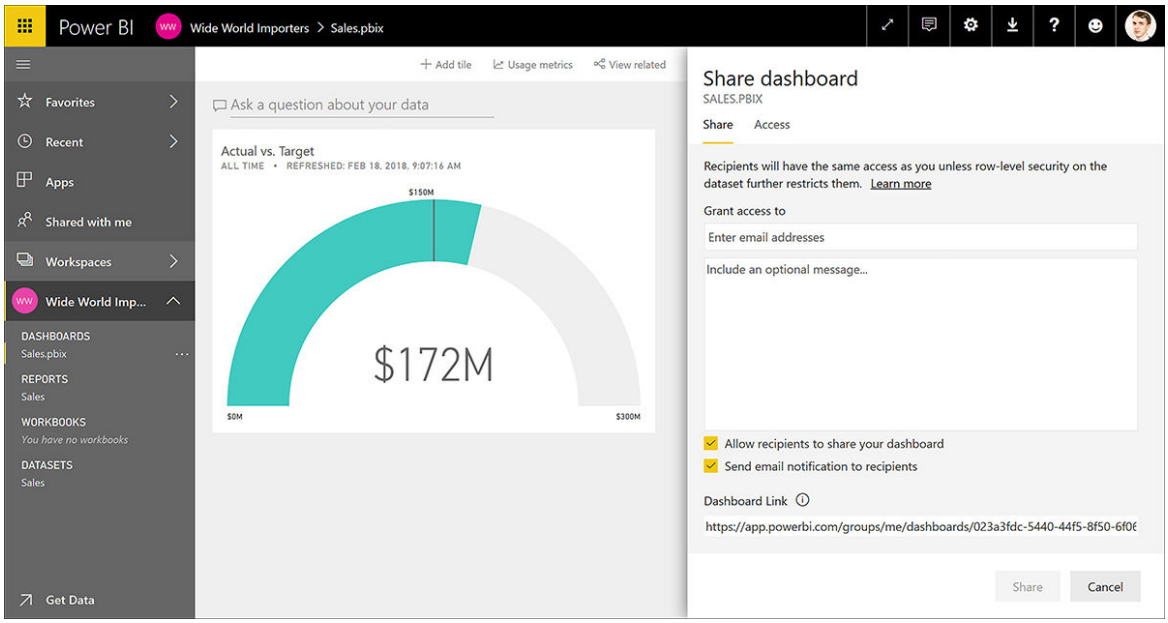


Figure 3.31 Share dashboard

2. Fill in the email addresses of the users with whom you want to share the dashboard. Alternatively, you can type the name of the security group that contains the users. In this example, we are going to share this dashboard with a user called Test User. You can share with users outside of your tenant, but you will receive a warning shown in Figure 3.32.

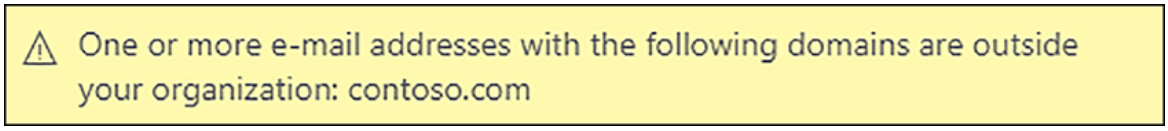


FIGURE 3.32 Sharing outside your organization

3. Optionally, you can include a message for the recipients.
4. By default, your recipients can share your dashboard. If you do not want your dashboard to be shared further, uncheck the **Allow Recipients To Share Your Dashboard** checkbox.
5. If you do not want to notify your recipients that you shared a dashboard with them, uncheck the **Send Email Notification To Recipients** checkbox.
6. Click **Share**.

At this stage, the users you shared your dashboard with will have immediate access to it. You can share a dashboard link with them directly, or they can click **Shared With Me** in Power BI service to see the content you shared with them.

To revise the access rights you previously granted, you can click on the **Access** tab in the Share dashboard pane (see Figure 3.33).

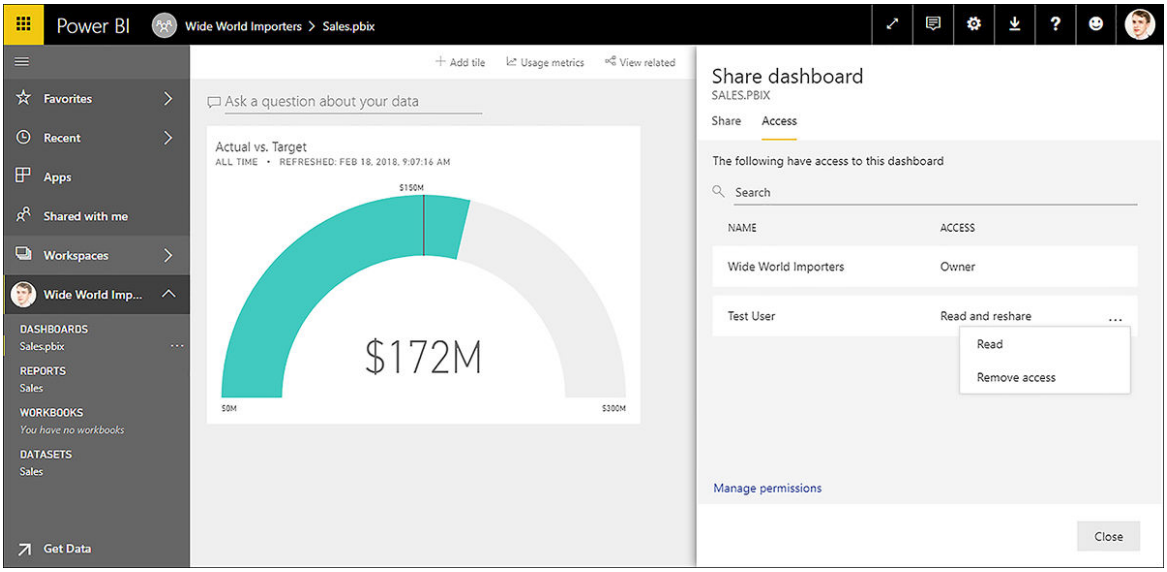


Figure 3.33 Share dashboard Access tab

By default, the app workspace in which the dashboard is published is listed as **Owner** in the **Access** tab. For every other user, you can change the level of access by clicking on the ellipsis. For example, the access level of Test User, who previously was granted **Read And Reshare** access, can either be removed or changed to Read.

If you click **Manage Permissions**, you will be taken to the Manage access page, where you will see the dashboard and the related content, such as reports and datasets. For each item, you will be able to revise the access level of each user.

> ***NOTE*** **SHARING A REPORT**
>
> The steps you need to follow to share a report are identical to sharing a dashboard.

> ***MORE INFO*** **SHARING DASHBOARDS AND REPORTS**
>
> For a video overview and more details on how you can share a dashboard or a report in Power BI service, including considerations on sharing with people outside of your organization, see “Share your Power BI dashboards and reports with coworkers and others” at *https://docs.microsoft.com/en-us/power-bi/service-share-dashboards*.

Configuring access to app workspaces

If you want to allow users to edit existing reports or dashboards, or create new ones from shared datasets, you will need to give them access to the app workspace with the content you want to share. In this section, we review the steps necessary to configure access to app workspaces; app workspaces are covered in more detail in Skill 3.5: “Configure apps and apps workspaces.”

If you have an existing app workspace, you can configure access to it in one of two ways: in Power BI service or in Office 365 Admin center. In Power BI service, you can click **Workspaces** and then click on the ellipsis next to the workspace you want to configure access to, and click **Edit Workspace** (see Figure 3.34).

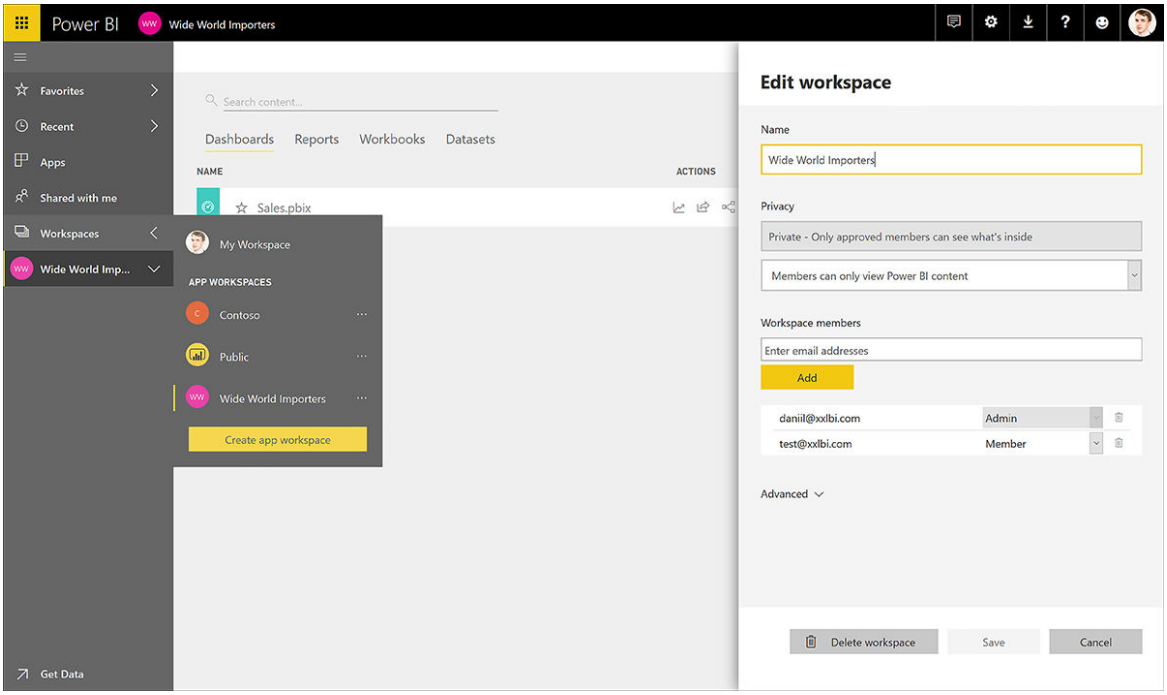


FIGURE 3.34 Editing workspace

> ***NOTE*** **EDITING APP WORKSPACES**
>
> Only workspace admins can see the Edit workspace option.

In the Privacy section, you can only change whether members can edit Power BI content; the other drop-down can only be used when you create an app workspace.

***MORE INFO* APP WORKSPACE PRIVACY**

The Privacy section of app workspaces is reviewed in more detail in "Skill 3.5: Configure apps and apps workspaces."

Below the Privacy section, you can see a list of members who already have access to the app workspace, along with their access rights, which can be either Admin or Member. Admins can edit workspaces and add other members to them, as well as see and edit all Power BI content in the workspaces. Members can view Power BI content and might also be able to edit it depending on the setting chosen in the Privacy section above.

To configure access to an app workspace from Office Admin center, you will need to follow these steps:

1. In Office Admin center, click **Expand Navigation Menu**.
2. Click **Groups** > **Groups**.
3. Click on the group that has the same name as the app workspace to which you want to configure access. Note that it will be of type Office 365 group.
4. Click **Edit** next to **Owners**, which is the equivalent of Power BI app workspace admins, or Members.
5. If you clicked **Edit** next to **Owners**, click **Add Owners**; if you clicked Edit next to Members, click **Add Members**.
6. Select the users you want to add; you can use the search bar if needed.
7. Click **Save**.
8. Click **Close** > **Close** > **Close**.

IMPORTANT APP WORKSPACE ADMINS

The users that you want to add as admins to an app workspace through Office Admin center need to be added as both Owners and Members to the relevant Office 365 group. If a user is added only as an Owner but not as a Member, she will not be able to see the app workspace in Power BI service.

Configure the export and sharing setting of the tenant

By default, Power BI content can be shared with external users and data can be exported using several options. To change the default export and sharing settings

in Power BI service, click **Settings** > **Admin Portal** > **Tenant Settings**. The Tenant settings tab can be seen in Figure 3.35.

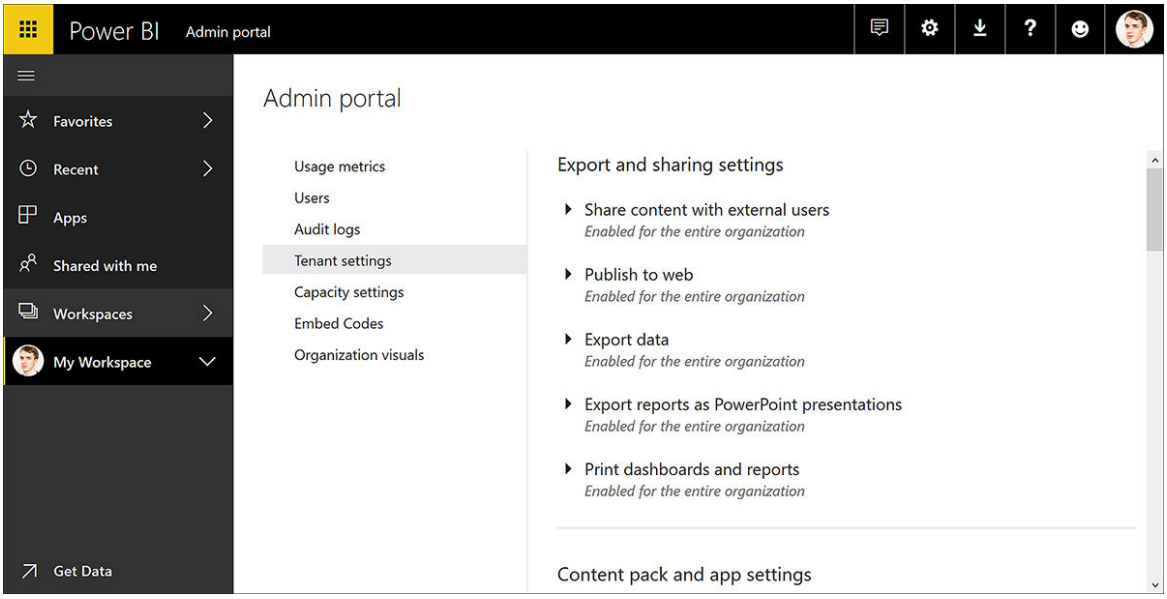


Figure 3.35 Tenant settings

In Tenant settings, you can see the following settings groups:

- Export And Sharing Settings
- Content Pack And App Settings
- Integration Settings
- Custom Visuals Settings
- R Visuals Settings
- Audit and Usage Settings
- Dashboard Settings
- Developer Settings

In this section, we review the Export And Sharing Settings. We also cover Content Pack And App Settings in Skill 3.5: "Configure apps and apps workspaces." Other settings are out of the scope of this section.

> ***MORE INFO* POWER BI ADMIN PORTAL**
>
> For an overview of Power BI admin portal (including other tabs) as well as details on the other Tenant settings groups, see "Power BI admin portal" at https://docs.microsoft.com/en-us/power-bi/service-admin-portal.

In the Export And Sharing settings, you disable or enable the following settings:

- **Share Content With External Users** This setting allows users to share Power BI content with users outside of the tenant.
- **Publish ToWeb** This setting allows users to publish reports to the web as detailed in Skill 3.3: “Publish and embed reports.”
- **Export Data** This setting controls whether users can export data from dashboard tiles and report visuals, as well as use the Analyze in Excel feature and connect live to Power BI service from Power BI Desktop.
- **Export Reports As PowerPoint Presentations** Disabling this setting prevents users from exporting Power BI reports as PowerPoint presentations.
- **Print Dashboards And Report** If you disable this setting, users will not be able to print dashboard or reports from Power BI service.

Each setting is enabled by default, and you can either disable it for the entire organization or enable for a subset of your organization. If you want to enable a setting for a subset of your organization, you have several choices:

- **Enable For Specific Security Groups Only** This option allows you to enable a setting only for the groups that you specify, excluding everyone else.
- **Enable For The Entire Organization Except For Specific Security Groups** This option allows you to disable a setting only for the groups that you specify, enabling the setting for everyone else.
- **Enable For Specific Security Groups While Excluding Other Security Groups** This option is relevant when you have users that are part of multiple security groups, and you want to enable access for one group but disallow it to another group of users who might be part of the first group. An example of this security setup can be seen in Figure 3.36.

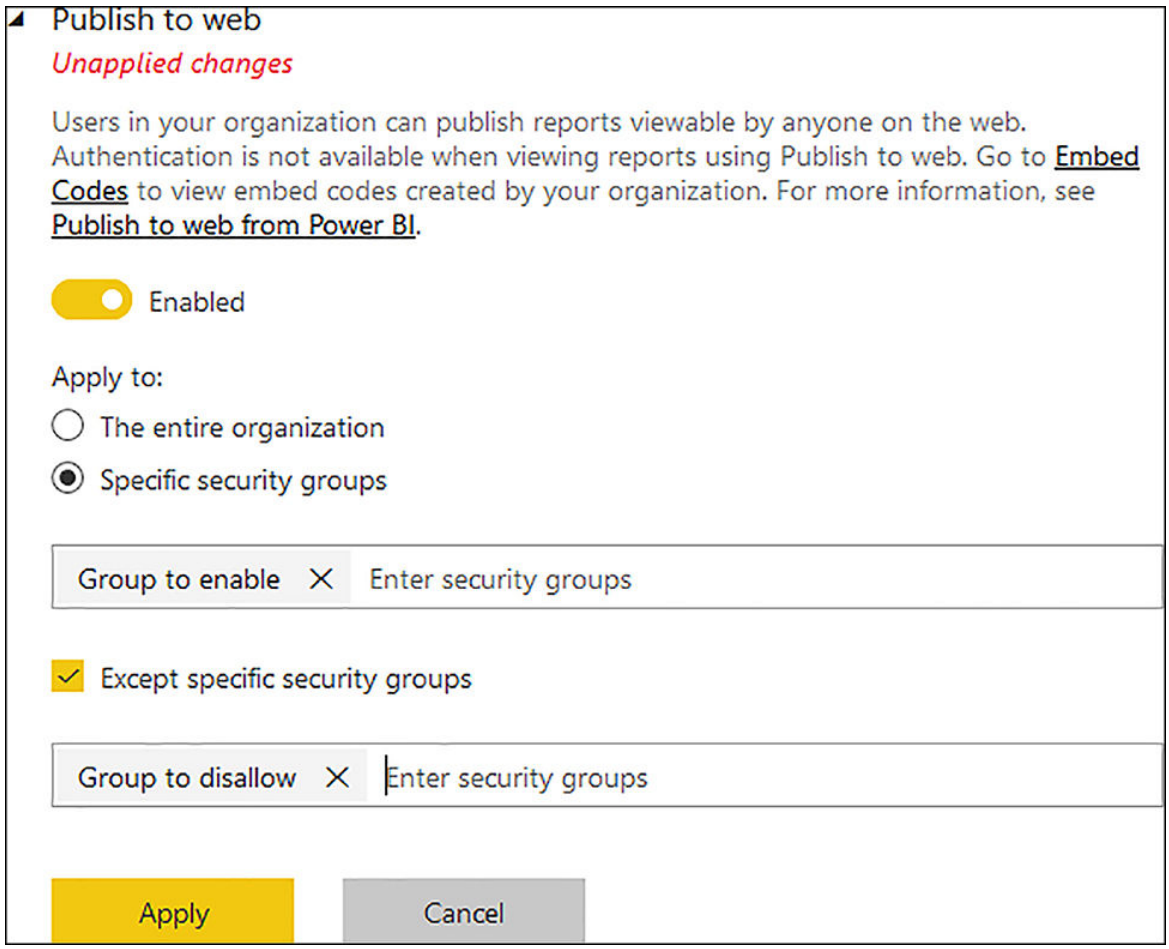


FIGURE 3.36 Tenant settings and security groups

If you modify settings for a feature, you will see a red **Unapplied Changes** message underneath the feature name. Settings are not saved automatically as you change them; to apply changes, you will need to click the **Apply** button.

> ***MORE INFO*** **EXPORTING DATA FROM POWER BI**
>
> For information on how you can export data from Power BI dashboard tiles and report visuals, see "Export data from visualizations" at *https://docs.microsoft.com/en-us/power-bi/power-bi-visualization-export-data*.

Configure row-level security

A common business requirement is to secure data so that different people who view the same report can see different subsets of data. In Power BI, this can be accomplished with the feature called row-level security, abbreviated RLS.

Row-level security restricts data by filtering it at the row-level depending on the rules defined for each user. To configure row-level security, you first need to define roles in Power BI Desktop and then assign users to the roles in Power BI service.

> ***NOTE* ROW-LEVEL SECURITY AND ANALYSIS SERVICES LIVE CONNECTIONS**
>
> Defining roles in Power BI only works for imported data and DirectQuery. When you connect live to an Analysis Services data model, Power BI will rely on row-level security configured in Analysis Services, and you cannot override it by creating roles in Power BI Desktop. Row-level security in Analysis Services works similarly to Power BI: you define roles and then allocate users to the roles; for more information on the topic, see "Roles" at *https://docs.microsoft.com/en-us/sql/analysis-services/tabular-models/roles-ssas-tabular*.

Creating roles in Power BI Desktop

To create a role in Power BI Desktop, you need to click **Modeling** > **Security** > **Manage Roles**, and then **Create** in the **Manage Roles** window as shown in Figure 3.37. This will create a new role with a default name of **New role**.

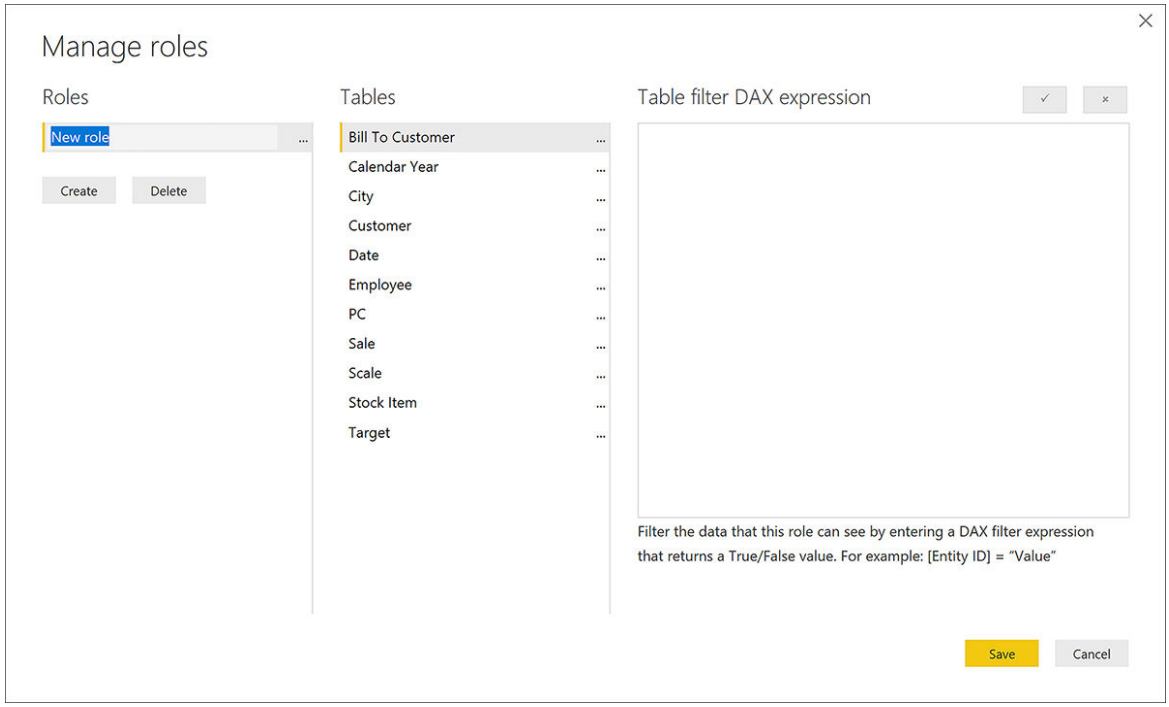


Figure 3.37 Manage roles

When you create a role, you have an option to change the default name to a new one. It is important to give roles user-friendly names because you will see them in Power BI service, and you will need to be able to assign users to the correct roles. All roles are listed in the Roles section of the Manage roles window.

If you right-click on a role or click on the ellipsis next to a role, you will be presented with the following options:

- **Create** This creates a new role and is an alternative to the Create button below the list of roles.
- **Duplicate** This will create a copy of the currently selected role.
- **Rename** This will rename the currently selected role; you can also rename a role by double-clicking on its name.
- **Delete** This will delete the currently selected role; this action can also be performed by clicking **Delete** below the list of roles.

For each role, you can define a table filter DAX expression for each table. When row-level security is configured, these expressions will be evaluated against each row of the relevant table, and only those rows for which the expressions are evaluated as `true` will be visible.

You can either type a table filter DAX expression from scratch or use the ellipsis menu next to each table to add an expression that you can then customize. The menu can also be accessed by right-clicking on a table. There are three options in the menu:

- **Add Filter** This option lists all columns available in the table, as well as an option called Hide all rows.
- **Copy Table Filter From** This option can copy a table filter DAX expression from another role that has a filter expression defined for the table.
- **Clear Table Filter** This option removes any table filter DAX expression from the table; this is a shortcut to erasing all text from the Table filter DAX expression area manually.

For example, in the Wide World Importers that we previously created, you can right-click **Date** > **Add Filter** > **[Calendar Year]**. This will insert an expression like the following one into the Table filter DAX expression area:

```
[Calendar Year] = 0
```

Depending on the data type of a column, you might see a different placeholder DAX expression, like the following ones:

Click here to view code image

```
// Column of type Date
[Date] < DATE(2018,05,23)
// Column of type Text
[Calendar Year Label] = "Value"
// Column of type True/False
[Is Latest]
```

If you decide to modify the expression, you can then validate it with the checkmark button in the top-right corner of the Manage roles window. If the expression is invalid, you will see a warning stating the syntax is incorrect below the Table filter DAX expression area, like the one shown in Figure 3.38.

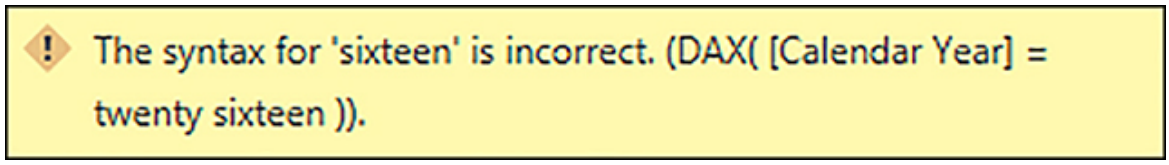


FIGURE 3.38 Row-level security syntax error

Next to the checkmark button, there is the cross button, which reverts any changes that have not been applied yet.

If you want to hide all rows in a table, you can right-click on it and click **Add Filter** > **Hide All Rows**. This will add the following table filter DAX expression:

```
false
```

Because `false` is never going to be `true` for any row, no rows will be shown in this case.

In our example, we can name our role **Year 2016** and create the following table filter DAX expression in the Date table:

```
[Calendar Year] = 2016
```

Also, we can duplicate the role by right-clicking on it and selecting **Duplicate**. We should then rename the new role to **Year 2015** and modify the Date table filter DAX expression as shown here:

```
[Calendar Year] = 2015
```

IMPORTANT DUPLICATING ROLES

If you duplicate a role before you validate the last added table filter, the table filter will not be copied to the duplicate role.

Once you make the necessary changes, click **Save** and publish the report. In this example, we are going to publish the report as **Sales RLS**.

Viewing as roles in Power BI Desktop

In Power BI Desktop, you can see what the users with specific roles will see even before you publish your report to Power BI service and assign users to roles. For this, you need to have at least one role defined and click **Modeling** > **Security** > **View As Roles**. You will then see the View as roles window shown in Figure 3.39.

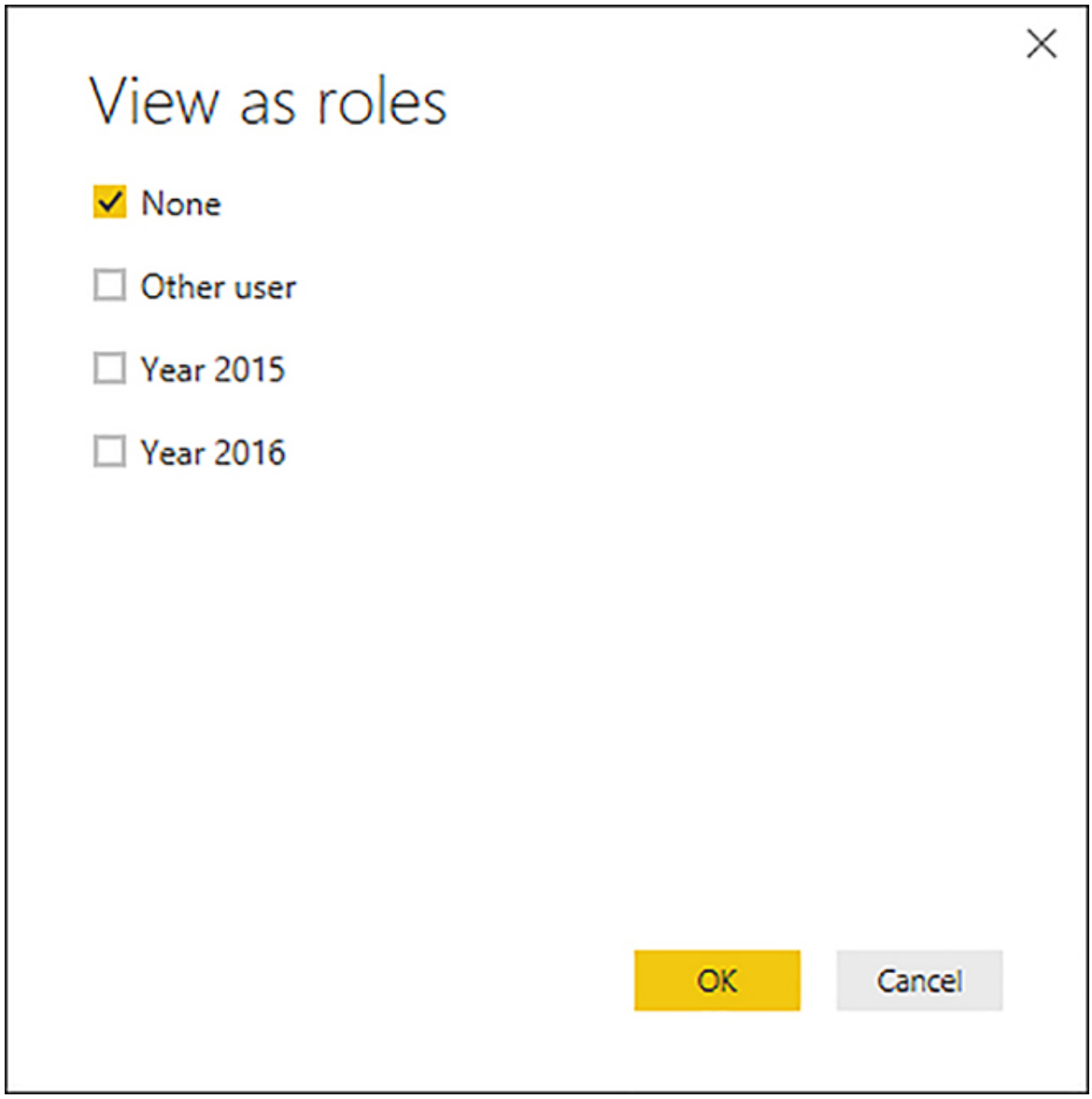


FIGURE 3.39 View as roles

Note that you can view as several roles simultaneously. This is because you can allocate a single user or a security group to multiple roles in Power BI service; in this case, the security rules of the roles will be combined using the AND logic. For example, you can select **Year 2015** and **Year 2016** and click **OK**. You will then see the report with the bar shown in Figure 3.40.

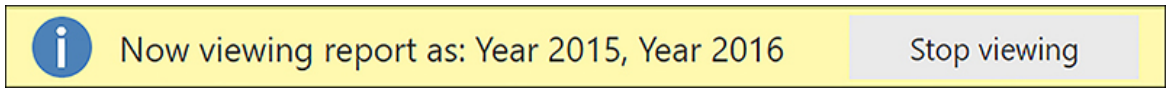


FIGURE 3.40 Now viewing report as

The report data will now be filtered to years 2015 and 2016. It is important to understand that the filters applied by row-level security are only applied at query time and not processing time. This behavior can be clearly demonstrated if you add a calculated column to the Date table with the following formula:

Click here to view code image

```
Number of years column = DISTINCTCOUNT ( 'Date'[Calendar Year] )
```

At this point, even if you view the report using the Year 2015 or Year 2016 role, or both, the column will still contain the value of **4**, regardless of your choice. On the other hand, if you define the following measure, it will display different values in a Card visual, depending on whether you chose one or two roles:

Click here to view code image

```
Number of years measure = DISTINCTCOUNT ( 'Date'[Calendar Year] )
```

Another option in the **View As Roles** window is **Other** user. With this option, you can test dynamic row-level security, which is covered next.

> ***MORE INFO*** **ROW-LEVEL SECURITY IN POWER BI DESKTOP**
>
> For more information on how you can create and configure security roles in Power BI Desktop, including the currently imposed limitations, see "Row-level security (RLS) with Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/desktop-rls*.

Dynamic row-level security

The roles we have created so far have been static, which means that all users within a role will see the same data. If you have many rules according to which

you need to secure your data, this approach might result in creating many roles, as well as updating the data model every time a new role should be introduced or an old one should be removed.

There is an alternative approach, called dynamic row-level security, which allows you to show different data to different users within the same role. For this, your data model must contain the usernames of people who should have access to the relevant rows of data. You will also need to pass the active username as a filter condition. Power BI has two functions that allow you to get the username of the current user:

- **USERNAME** Returns the domain and login of the user in the domain\login format.
- **USERPRINCIPALNAME** Depending on how the Active Directory was set up, this function usually returns the email address of a user.

> ***NOTE*** **USING USERNAME AND USERPRINCIPALNAME**
>
> If your computer is not part of an Active Directory domain, both functions will return the same result[md]computer name\login. Once you publish your dataset to Power BI service, both functions will return the email address of the user. These functions can only be used in measures or table filter DAX expressions; if you try to use either of them in a calculated column or a calculated table, you will get an error.

To review how dynamic row-level security works using the Wide World Importers data model we created earlier in the book, we can perform the next steps.

1. Right-click the **Date** table in the **Fields** pane and click **New Column**.
2. Specify the following formula:
   ```
   UserAccess = "test" & FORMAT ( ‘Date’[Date], "yyyy" ) & "@xxlbi.com"
   ```
3. Create a new security role by clicking **Modeling** > **Security** > **Manage Roles** > **Create**.
4. Rename the new role to **Single Role**.
5. In the Manage roles window, right-click on the **Date** table, then click **Add Filter** > **[UserAccess]**.
6. Modify the table filter DAX expression to the following one:
   ```
   [UserAccess] = USERPRINCIPALNAME()
   ```

7. Click **Save**.
8. Click **Modeling** > **Security** > **View As Roles**.
9. Click the **Other User** and **Single Role** checkboxes.
10. In the text field next to Other user, type **test2016@xxlbi.com**.
11. Click **OK**.

In the steps above, we created a calculated column that contained a different email address for each calendar year. In this way, we segregated access to data for each user based on calendar year. We have done so for review purposes; in real life, sales data may be secured based on region or department, or any other business logic, and these rules are often contained in a single security table that is maintained manually.

If you followed the steps above, you would see only 2016 data, as well as the bar shown in Figure 3.41; this bar would appear at the top of the Report or Data view.

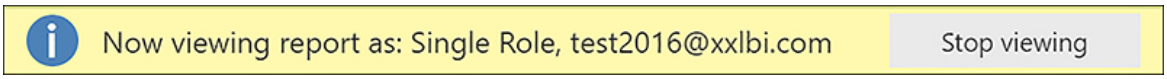


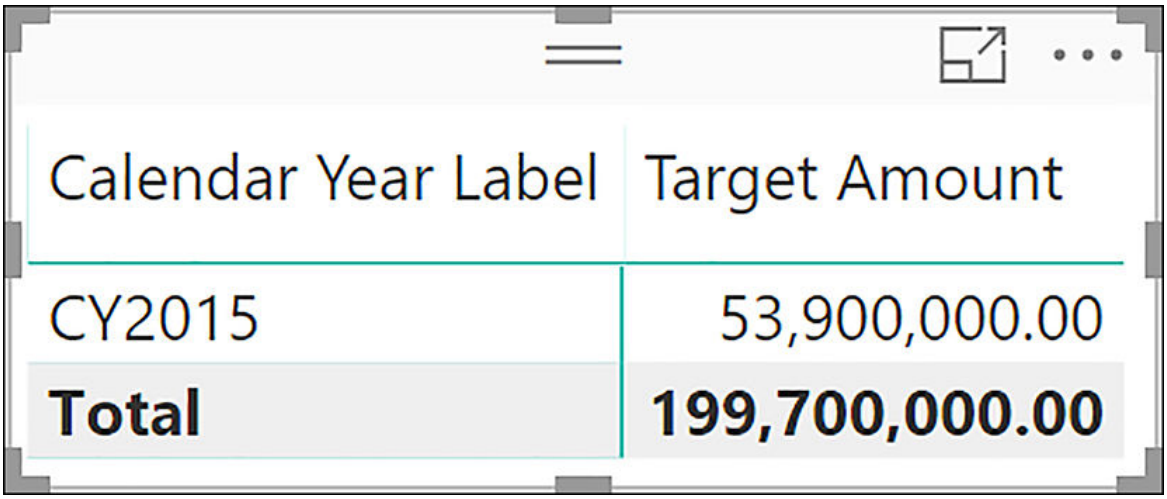
FIGURE 3.41 Viewing report as a specific user

At this stage, click **Modeling** > **Security** > **View As Roles**, then change the email address next to Other user to **test2015@xxlbi.com** and click **OK**. There are two things to note here:

- Only 2015 data is showing now.
- You are still using the same role as before **Single Role**.

Because you are using a single role, this approach is preferable in large-scale implementations of Power BI where there are many users that need to see different data.

If you are using bidirectional relationships, it is important to note that row-level security filters only go in one direction by default. This might yield unexpected results. For example, if we create a table that shows the Target Amount by Calendar Year Label, we will see the target for one year, but the total target amount for all years at the total level, like in Figure 3.42.



Calendar Year Label	Target Amount
CY2015	53,900,000.00
Total	**199,700,000.00**

Figure 3.42 Target Amount by Calendar Year Label before applying security filters in both directions

This behavior is due to the Date table security filters not reaching the Target table because there is a bidirectional relationship between them. To make the security filters go in both directions, you need to do the following:

1. Click **Home** > **Relationships** > **Manage Relationships**.
2. Select the relationship that should be affected by row-level security. In our Wide World Importers example, it is the relationship between the Date table and the Calendar Year table.
3. Click **Edit**, which opens the Edit relationship window.
4. Click the **Apply Security Filter In Both Directions** checkbox. Note that this option is active only if the cross filter direction is set to **Both**.
5. Click **OK** > **Close**.

After you follow the steps above, you might notice that the Target amounts are now also filtered to one year only, as shown in Figure 3.43.

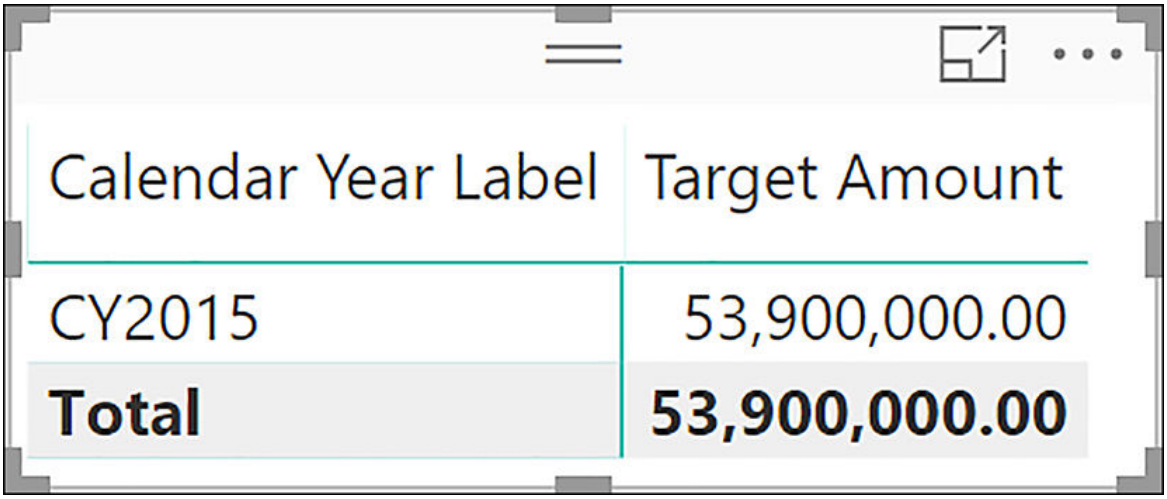


FIGURE 3.43 Target Amount by Calendar Year Label after applying security filters in both directions

MORE INFO **DYNAMIC ROW-LEVEL SECURITY**

For a detailed overview of dynamic row-level security, see an article by Tri Nguyen, "Dynamic RLS (row level security) with Power BI" at *https://tringuyenminh92.com/index.php/2017/08/11/dynamic-rls-row-level-security-with-power-bi/*.

Assigning roles in Power BI service

Once you have configured row-level security in Power BI Desktop, you will need to publish your report to Power BI service and add members to each role. To do this, you can right-click on the dataset for which you want to configure row-level security and then click **Security**. You will be presented with a page as shown in Figure 3.44.

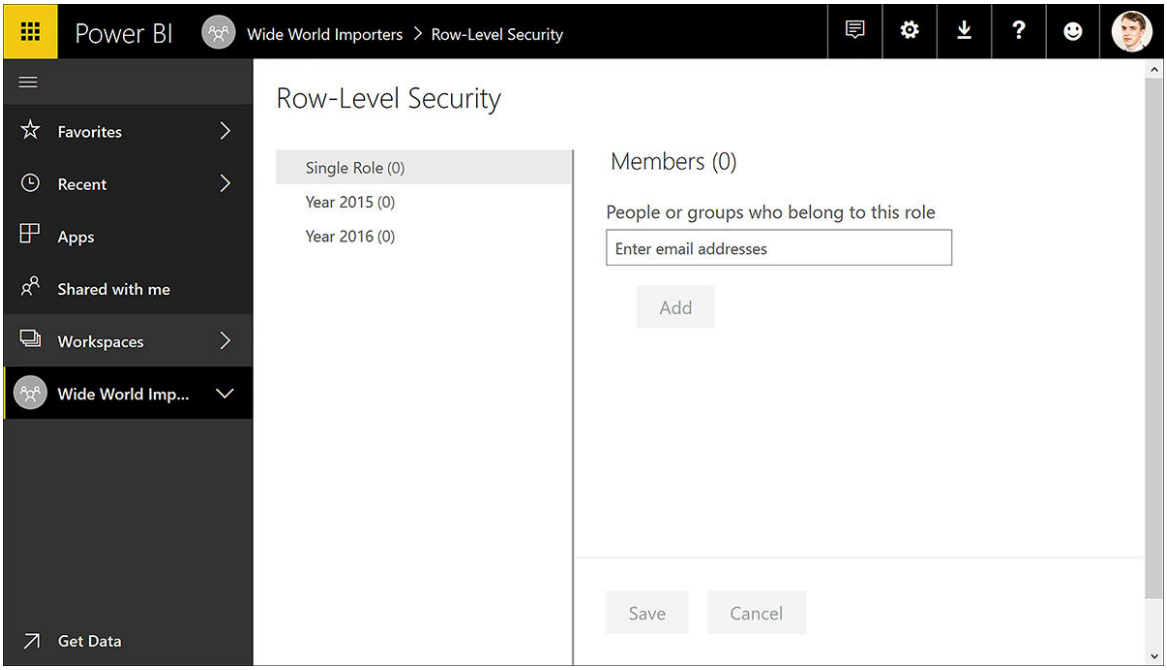


Figure 3.44 Row-Level Security page

On the left side of the Row-Level Security page, you can see a list of all roles in the dataset. The numbers in brackets show how many members each role has. On the right, you can view, add, or remove members for the selected role.

To add a member to a role, you need to select a role on the left first, then enter email addresses or security groups in the **People Or Groups Who Belong To This Role** field. As an example, we can add My Security Group to the Year 2015 role. After you enter the security group name, you need to click **Add** > **Save**. The changes will be applied immediately.

If you want to remove a member from a role, you can click on the cross next to the member and then **Save**.

Viewing as roles in Power BI service

Like the View As Roles feature in Power BI Desktop, you can test roles in Power BI service. For this, you need to hover over a role on the **Row-Level Security** page, click the ellipsis, and then click **Test As Role**. You will then see the way a report appears to members of the role, like in Figure 3.45.

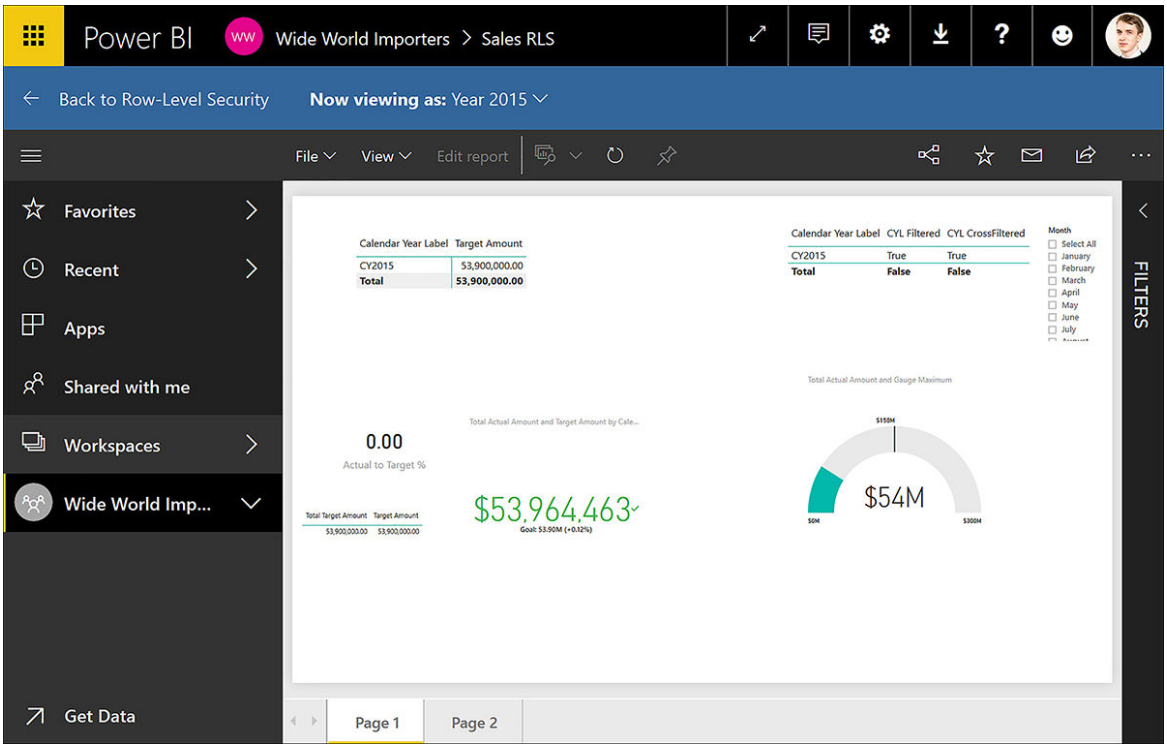


Figure 3.45 Testing a role in Power BI service

If needed, you can test a combination of roles or view as a specific person by clicking on the role in the blue bar at the top and selecting the desired parameters. Once you are satisfied with how the roles work, you can click **Back To Row-Level Security**.

> ***IMPORTANT ROW-LEVEL SECURITY AND WORKSPACE PRIVACY***
>
> Row-level security will only work if the workspace privacy is set to Members can only view Power BI content. Workspace admins and members with edit rights will always see the whole dataset regardless of the security settings, even though the Test as role feature may show a restricted dataset.

> ***MORE INFO* ROW-LEVEL SECURITY IN POWER BI SERVICE**
>
> For an overview of how you can configure row-level security in Power BI service, including a video tutorial, limitations and common issues, see “Row-level security (RLS) with Power BI” at *https://docs.microsoft.com/en-us/power-bi/service-admin-rls*.

Skill 3.5: Configure apps and apps workspaces

You can collaborate with others on creating Power BI content in app workspaces. Once you are done with creating datasets, reports, and dashboards, you can package your content as apps and distribute them in your organization. In this section, we are going to review the skills necessary to create and configure a Power BI app workspace, as well as create, publish, and update apps.

This section covers how to:

- Create and configure an app workspace
- Publish an app
- Update a published app
- Package dashboards and reports as apps

Create and configure an app workspace

To create an app workspace in Power BI service, you need to click **Workspaces** > **Create App Workspace**. You will then see the Create an app workspace menu shown in Figure 3.46.

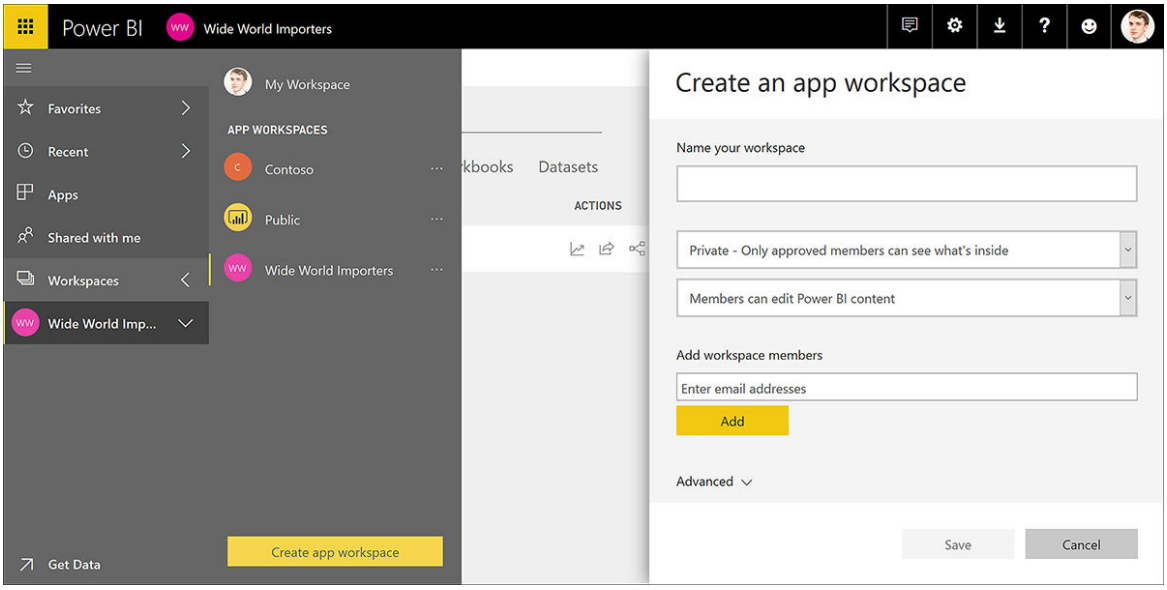


FIGURE 3.46 Create an app workspace

In the menu, you will need name your app workspace and choose its privacy level. The two available options are as follows:

- **Public** Anyone can see what's inside
- **Private** Only approved members can see what's inside

The default option is **Private**. Note that in case of public workspaces, "anyone" does not mean anyone on the Internet; instead, it means that users with an Exchange Online and a Power BI Pro license can make themselves members of the group through an Outlook app.

> ***MORE INFO*** **JOINING AN APP WORKSPACE**
>
> To become a member of a public app workspace, a user would need to join a group in Outlook, which can be done with a desktop app or Outlook on the web. For detailed instructions, see "Join a group in Outlook" at *https://support.office.com/en-us/article/join-a-group-in-outlook-2e59e19c-b872-44c8-ae84-0acc4b79c45d*.

You can also choose who can edit Power BI content by selecting either of the following two options in the next drop-down list:

- **Members Can Edit Power BI Content**
- **Members Can Only View Power BI Content**

By default, members are given **Members Can Edit Power BI Content** access. As discussed in Skill 3.4, row-level security only works for members with read-only access.

You can add members by typing their email addresses in the Add workspace members text field and clicking **Add**. For each member, you can specify if they are going to be an **Admin** or a **Member**; the default is **Member**.

In the advanced settings, you can allocate the workspace to a Premium capacity if it is available in your organization.

> ***MORE INFO*** **MANAGING APP WORKSPACES**
>
> For more information on how you can manage your app workspaces, including setting an app image and description, see "Manage your app workspace in Power BI and Office 365" at *https://docs.microsoft.com/en-us/power-bi/service-manage-app-workspace-in-power-bi-and-office-365*.

> ***MORE INFO*** **COLLABORATING IN APP WORKSPACES**

Because Power BI app workspaces are Office 365 groups, you can also collaborate in app workspaces outside of Power BI service, including having group conversations and scheduling events. For more details, see "Collaborate in your Power BI app workspace" at *https://docs.microsoft.com/en-us/power-bi/service-collaborate-power-bi-workspace*.

Publish an app

When you are ready to share your reports and dashboards with users in your organization, you can publish an app. An app is a collection of Power BI items, such as dashboard, reports, and workbooks, packaged together. Only members who have edit access to an app workspace can publish apps. Currently, there can only be one app per app workspace.

To publish an app in Power BI service, you need to go to the app workspace view first. For this, you need to click **Workspaces** in the navigation pane on the left and click the app workspace from which you want to publish an app (see Figure 3.47).

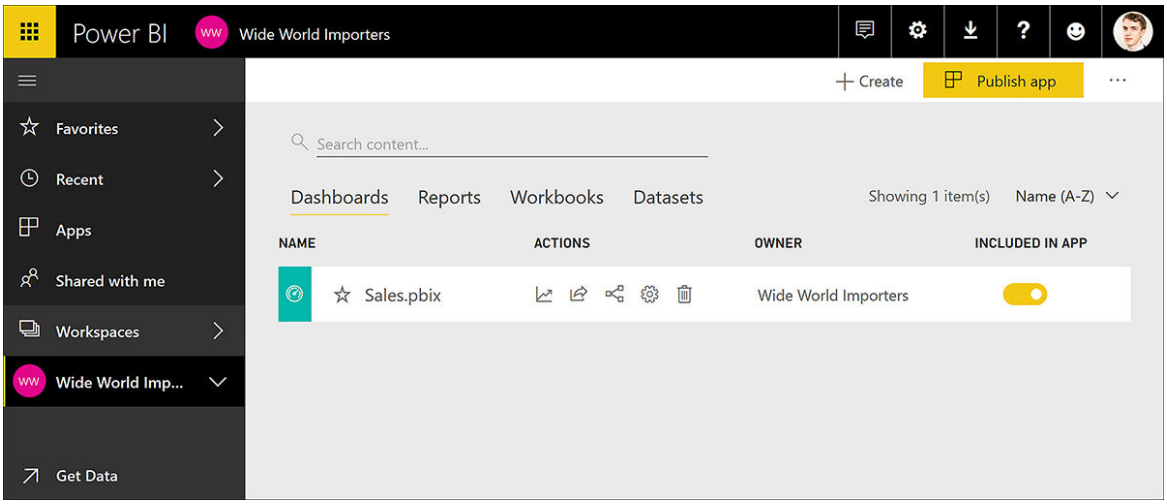


Figure 3.47 App workspace view

In our Wide World Importers example, we have one dashboard, called Sales.pbix, and two reports: Sales and Sales RLS, which also have datasets with the same names. Next to each item, there are a number of actions available, depending on the item type; the reader is encouraged to review the actions on their own.

To publish an app, you need to click the **Publish App** button in the top-right corner of the window. You will then be taken to the **Publish App** page, where

you will see three tabs: **Details**, **Content**, and **Access**. The Details tab is shown in Figure 3.48.

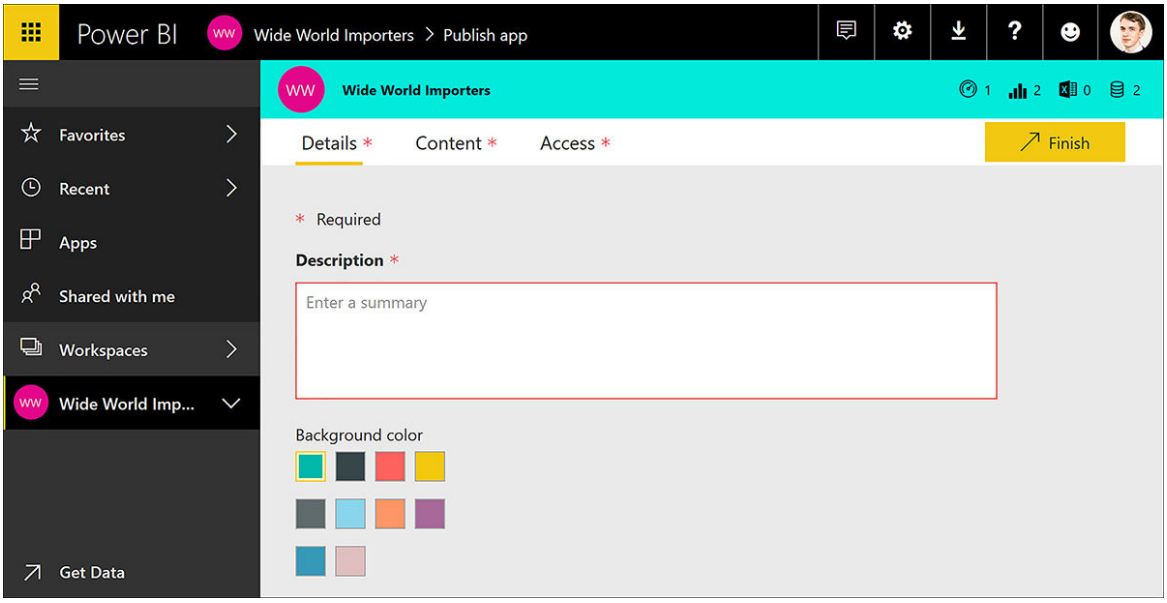


FIGURE 3.48 App details

In the Details tab, you need to enter an app description to help users understand its contents. You can also personalize your app by selecting a background color. For example we can enter **Exam Ref App** as description. Once you specify description, you may click the **Content** tab, shown in Figure 3.49.

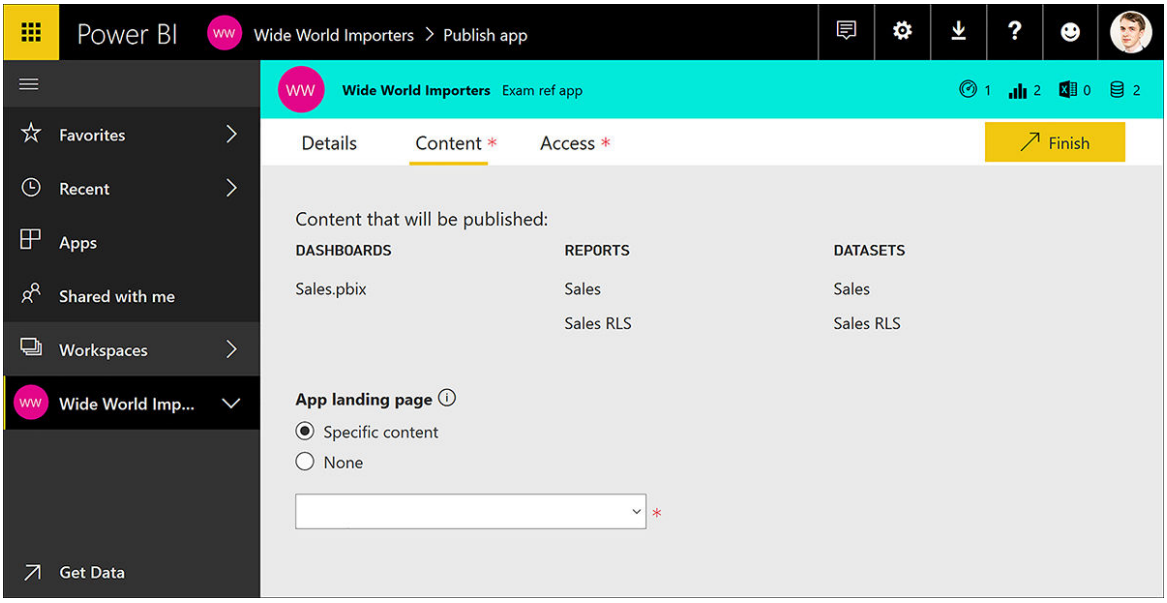


FIGURE 3.49 App content

In the Content tab, you can see the Power BI content that will be published across three categories: Dashboards, Reports, and Datasets. Workbooks are listed in the Reports section. At this stage, you cannot exclude any items; we are going to review the steps you need to take to select specific items for an app later in this chapter.

Below the app content, you can specify the app landing page, which is what users who go to your app will see first. You can choose either of the following two options:

- **Specific Content** This can be a dashboard or a report, which you can select from the drop-down list below.
- **None** Users will see the app contents page instead of a specific item.

By default, **Specific content** is selected, though you need to select the landing item. In our example, we are going to select **None**. We can now proceed to the Access tab, shown in Figure 3.50.

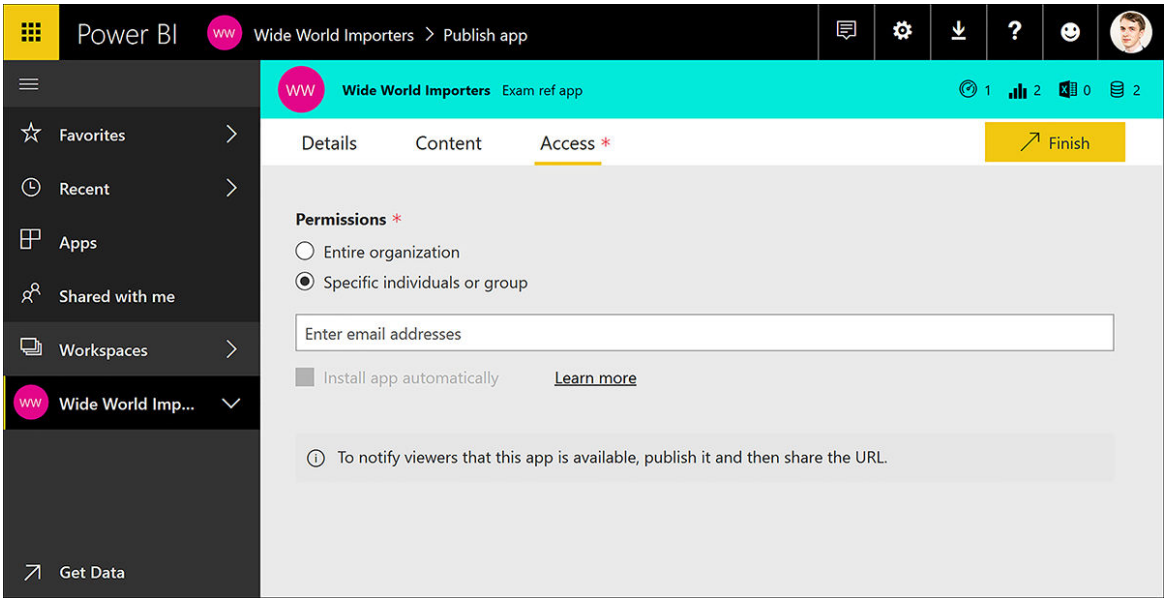


FIGURE 3.50 App access

For your app, you can choose to publish it to the entire organization or for specific individuals or groups. The default selection is **Specific Individuals Or Group**. If you choose to keep the setting, you will need to enter email addresses of individuals or groups below.

Below the email addresses input area, you can see a disabled checkbox, **Install App Automatically**. This option, if enabled, allows you to install apps automatically for specific individuals or groups, but not for the entire organization. To enable this option, you need to click **Settings** > **Admin Portal** >

Tenant Settings > **Content Pack And App Settings** > **Push Apps To End Users** > **Disabled** (this will change the setting to **Enabled**), **Apply**. As with other tenant settings, this feature can be enabled for the entire organization or a subset of it.

In our example, we can select **Entire Organization**, which does not allow you to install app automatically regardless of your tenant settings. Once we click **Finish** in the top-right corner, we will see the **Ready To Publish** window, where we need to click **Publish**. After this, we will see the **Successfully Published** window, shown in Figure 3.51.

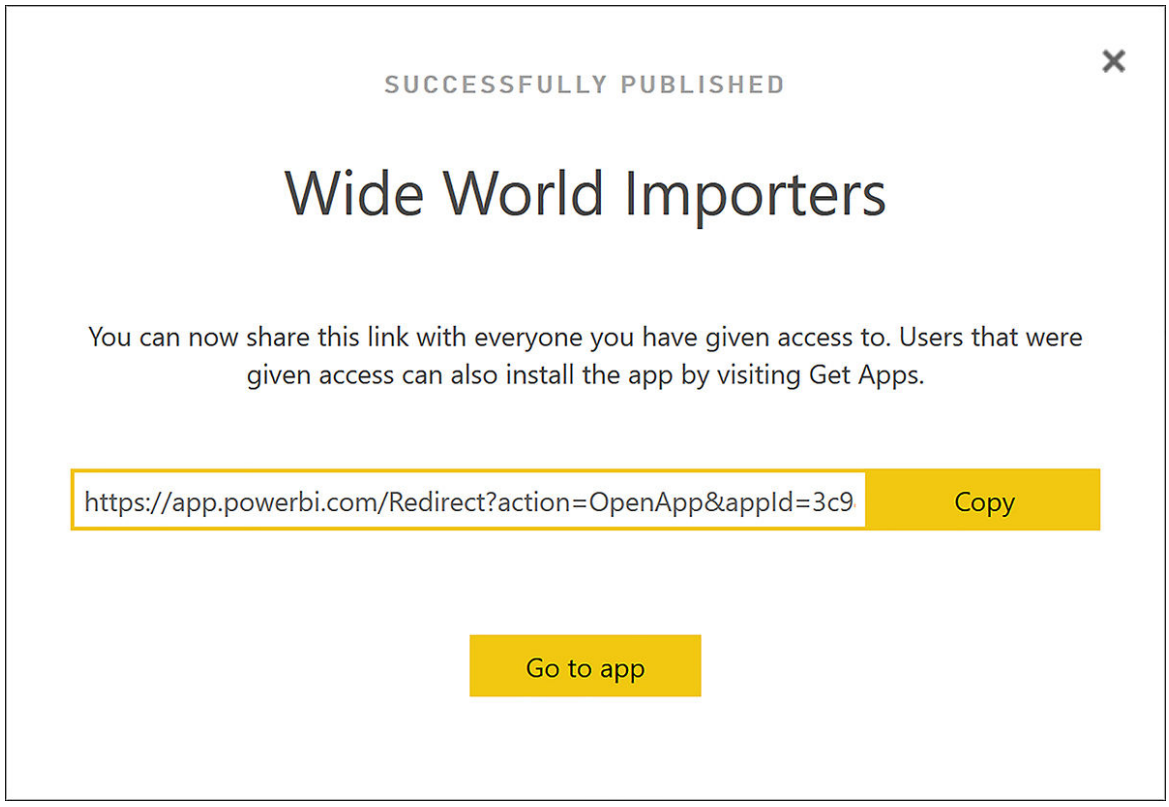


FIGURE 3.51 Successfully Published

In this window, you can copy the app link and share it with users to which you have given access. Alternatively, to get an app, a user can click **Apps** in the navigation pane in Power BI service, and then **Get Apps**, which will open the AppSource window, shown in Figure 3.52.

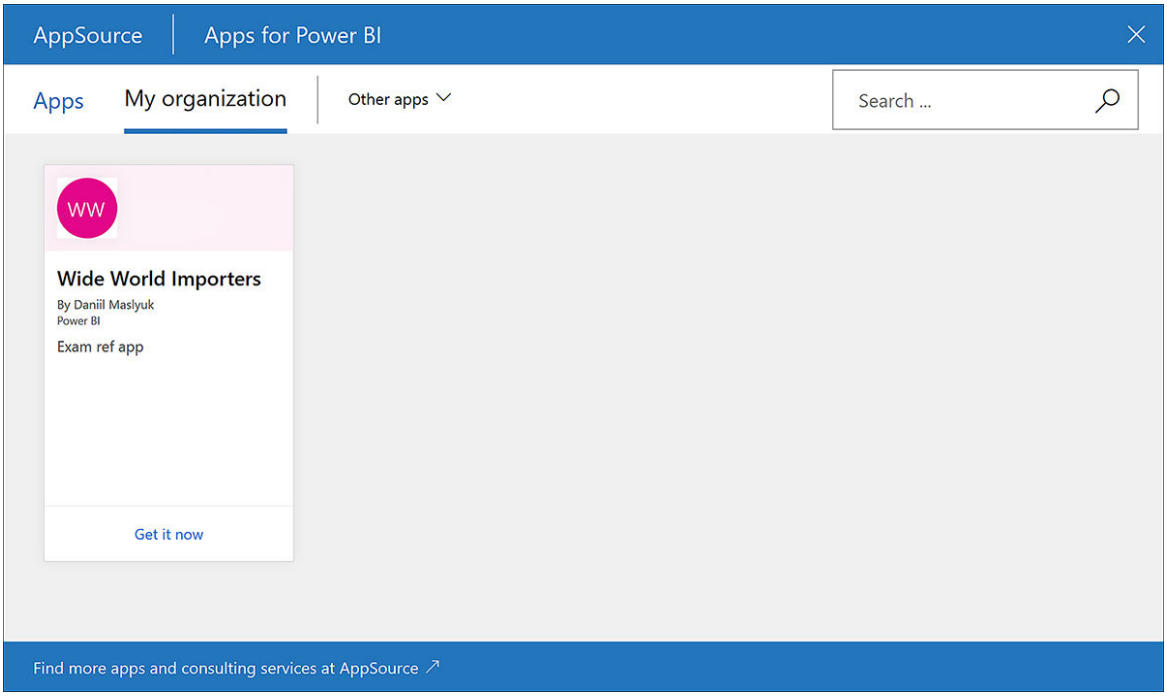


FIGURE 3.52 AppSource Apps for Power BI

In the AppSource window, they will need to click **Get It Now** next to the app they want to get. Note that a user does not need to be a member of the app workspace to get the app. Regardless of how the users get the app, it will then appear in Apps. If an app was installed automatically, users would not be able to delete it from their list of apps.

> ***MORE INFO* INSTALLING APPS**
>
> For more details on how users can install and use apps, see "Install and use apps with dashboards and reports in Power BI" at *https://docs.microsoft.com/en-us/power-bi/service-install-use-apps*.

Unpublishing an app

If you want to unpublish an app, you can do so by clicking the ellipsis in the top-right corner of the app workspace and clicking **Unpublish App**. You will need to confirm your action by clicking **Unpublish** in the Unpublishing an app window. Doing so will not delete the app workspace contents; instead, the app will be removed from Apps list of each user and become inaccessible.

Update a published app

After you publish your app, you can make changes to it if you are an app workspace admin or a member with edit rights. For this, you need to go to the app workspace and make the changes you want; once you have made the changes, you need to go back to the app workspace list of contents and click **Update App**. You can also update the Details, Content, and Access settings that you configured when you created the app. Clicking **Update App** will open the Ready to update dialog box, where you will need to click **Update** to propagate the app changes.

Note that in the Access tab, you will see the app link, as well as dashboard links. When you share any of those links, users will see all contents of the app, not just dashboards or reports.

Package dashboards and reports as apps

When creating or updating an app, you have an option to exclude some dashboards or reports from the app. In the app workspace view, shown in Figure 3.47, there is an Included in App switch next to each dashboard, report, and workbook; datasets are automatically included in apps. To exclude an item from the app, you need to click on the relevant **Included In App** switch. If you are excluding items from an app that has already been published, you will need to update the app.

When you exclude a report that was used to create a tile in a published dashboard, you will see a warning like the one shown in Figure 3.53.

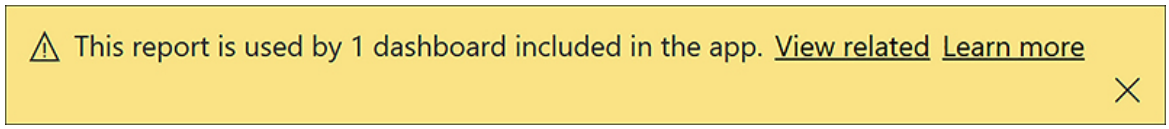


FIGURE 3.53 Report exclusion warning

This warning does not prevent you from publishing the app, but the dashboard that is using the report may display an error message at the top of it, and the affected tiles won't be displayed correctly. The error message is shown in Figure 3.54.

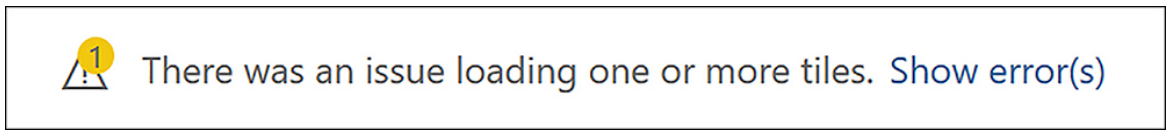


Figure 3.54 Dashboard tiles error

Clicking on the **Show Error(s)** hyperlink highlights the affected tiles.

***MORE INFO* CREATING APPS**

For a detailed overview on how to create apps, including a video introduction and frequently asked questions, see "Create and publish apps with dashboards and reports in Power BI" at *https://docs.microsoft.com/en-us/power-bi/service-create-distribute-apps*.

Thought experiment

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

You are the BI developer at Adventure Works responsible for enabling report creation and sharing by business users. Currently, there is an on-premises data warehouse built with SQL Server 2017, and there are some files stored on a shared drive that contain sales targets. The files are updated manually on a weekly basis. All reports need to be refreshed automatically at least once a day.

Everyone in the organization has a Power BI Pro license. Adventure Works has a hot desking policy, and all employees must lock their laptops in allocated cabinets.

Some business users would like to be able to create their own reports and share them internally. Only a select group of users must have the rights to share reports externally.

Adventure Works has a separate manager for each product category. One of the reports must be secured in such a way that each manager must be able to see the products of the category they manage only, while the CEO must be able to see all sales. An HR analyst maintains a table that maps the category name to the manager email address.

The management requested your assistance in making sure that business users can create and share their Power BI reports within the organization.

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

1. Business users ask your guidance on how they should configure automatic refresh. Which gateway installation mode is appropriate for Adventure Works?

   A. On-Premises Data Gateway.

B. On-Premises Data Gateway (Personal Mode).

2. How can you make sure that each category manager can see sales of their category only and allow the CEO to see all sales in a single report? Your solution must involve minimal effort.

   A. Create one report for each category manager, with a different category filter in each report, and a separate report for the CEO.

   B. Configure dynamic row-level security and a separate role for the CEO.

   C. Create a category slicer in the report.

3. The CEO wants to share the environmental report created in Power BI with external users. The report does not contain any sensitive data. How can this be achieved? Your solution must involve minimal effort.

   A. Invite the external users in Adventure Works Active Directory.

   B. Add the external users into an app workspace.

   C. Use the **Publish To Web** feature.

4. A business user reports that Q&A does not work on a certain dashboard. Which of the following is NOT a possible reason for this?

   A. Q&A is disabled for this dashboard.

   B. The only data source uses DirectQuery.

   C. There is a custom visual in one of the tiles.

   D. Row-level security is enabled for all datasets used in the dashboard.

5. A business user wants to distribute a set of reports with certain users. The reports were created in an app workspace. What is the best way to share the reports? Your solution must involve minimal effort for all parties involved.

   A. Share each report with each user.

   B. Create a security group containing all necessary users and share reports with the security group.

   C. Invite all users in the app workspace with the reports.

   D. Create an app and install it automatically.

Thought experiment answers

1. The answer is **A**. Because users need to lock their laptops away each night, a personal gateway is not a good choice. Furthermore, because the users need

to use the same data sources, installing gateways in personal mode is going to create extra work compared to installing a single data gateway.

2. The answer is **B**. Creating separate reports for each manager will duplicate the reports and make it difficult to maintain them. Creating a slicer in the report is not going to secure data at all. Dynamic row-level security is the most appropriate choice given that there is a table that can be used to filter categories based on the active username. The CEO can have a separate role to view everything.
3. The answer is **C**. Because the users with whom the report should be shared with might change over time, it is best to publish the report to web. Adding external users to Active Directory or an app workspace would involve extra costs.
4. The answer is **C**. Custom visuals do not interfere with the Q&A feature. On the other hand, Q&A box might disappear if it is disabled, the data source uses DirectQuery, or there is row-level security configured.
5. The answer is **D**. Having an app installed automatically adds it to the Apps section of Power BI service, which is an advantage compared to sharing the reports individually. Sharing reports with each user individually can take time and be error-prone because the user who shares will need to type all emails. A security group is preferred over typing email individually, because it can be maintained separately from Power BI service, but each report will still need to be shared individually. Inviting users to app workspaces should be done for collaboration purposes; it is not required when users need to only view reports.

Chapter summary

- To access on-premises data from Power BI service, also known as PowerBI.com, you need to install a data gateway. A data gateway is a piece of software that acts as a bridge between your on-premises data and Power BI service.
- After you install a gateway, you will need to add each data source in your gateway, and for each data source, you will need to add users so that they can use the data source in their reports published to Power BI service.
- When you have a gateway with data sources configured, you can schedule refresh for your datasets, which keeps your reports and dashboards up to date.

- You can publish your reports to Power BI service from Power BI Desktop; this will create a dataset and a report in Power BI service with the same name as your Power BI Desktop file. If you already have a dataset with the same name, you will be asked if you want to replace it.
- Reports published to Power BI service can be edited in Power BI service directly, though you can only edit the report layout in this way. If you want to change the data model by adding a measure, for example, you will need to download a .pbix copy of the report and edit in Power BI Desktop.
- In Power BI service, you can create dashboards by pinning report visuals; the dashboard visuals are called tiles. Dashboards differ from reports by having only one page and being able to combine tiles from multiple datasets. In addition to report visuals, you can also create tiles with static content, like text and images.
- Dashboard tiles can link to specific Power BI reports of your choice, or you can point them to a custom URL.
- You can perform natural language querying in Power BI service as well as Power BI Desktop. To improve the accuracy of queries, you can create featured questions in Power BI service and synonyms in Power BI Desktop.
- Reports can be made publicly available by using the Publish to web feature; this way, there is no security and anyone on the Internet can potentially access the report.
- One of the ways to securely publish a report is to publish it to Microsoft SharePoint. The users of the report will be authenticated with their Active Directory credentials and will only be able to see the reports if you explicitly grant them access rights.
- Power BI Report Server allows you to publish Power BI reports on-premises. There is significant difference between functionality of Power BI service and Power BI Report Server: for instance, it is not possible to edit reports in the browser in case of Power BI Report Server. On the other hand, Power BI Report Server allows you to comment on reports. Power BI Report Server uses a separate version of Power BI Desktop optimized for it.
- Instead of granting access to dashboards and reports to users one by one, you can create a security group in Office 365 admin portal and add the relevant users to that group; this way, you may specify just the security group when sharing Power BI content.

- You can share dashboards and reports by themselves, or you can create an app workspace and grant users access to it, where they will be able to access all content within the app workspace.
- Certain features, such as sharing and exporting data from reports and dashboards, can be controlled in Power BI tenant settings. Each feature can be enabled or disabled for the entire organization, or for a subset of the organization.
- Row-level security is a feature that allows you to secure your data; the feature filters rows based on the rules you specify in roles that you create in Power BI Desktop; you then allocate users to the roles in Power BI service. You can test your roles both in Power BI Desktop and Power BI service to see how your report appears to users. Row-level security only works for users who have read-only access to data.
- App workspaces, which are used for collaboration, are created in Power BI service; you can grant members edit or read-only rights; in addition to members, you can appoint admins who will always have edit rights and be able to add and remove members.
- You can publish your app workspace contents, such as dashboards, reports, and workbooks, as an app and push it to users; alternatively, users with access rights can install the app themselves. If you decide to make changes to your app, you can update it and users will then see the updated content. You can choose which items to include in the app before or after creating it.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Code Snippets

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

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

```
= 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"})
```

```
= 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"})
```

```
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"
```

// 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"
```

```
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]
```

```
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"
```

```
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"
```

```
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"
```

```
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"
```

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

```
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"
```

```
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')
```

```
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')
```

```
// 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')
```

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

```
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"
```

```
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'
```

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

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

```
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 [_]
```

```
= Table.AddColumn(Dimension_Date, "Year Month Number", each [Calendar Year] * 100
+ [Calendar Month Number], Int64.Type)
```

```
= Table.AddColumn(AddedYearMonthNumber, "Fiscal Year Month Number", each [Fiscal
Year] * 100 + [Fiscal Month Number], Int64.Type)
```

```
let
    Source = Sql.Databases("localhost"),
    WideWorldImportersDW = Source{[Name="WideWorldImportersDW"]}[Data],
    Dimension_Date = WideWorldImportersDW{[Schema="Dimension",Item="Date"]}[Data],
    AddedYearMonthNumber = Table.AddColumn(Dimension_Date, "Year Month Number",
each [Calendar Year] * 100 + [Calendar Month Number], Int64.Type)
in
    AddedYearMonthNumber
```

```
let
    Source = Sql.Databases("localhost"),
    WideWorldImportersDW = Source{[Name="WideWorldImportersDW"]}[Data],
    Dimension_Date = WideWorldImportersDW{[Schema="Dimension",Item="Date"]}[Data],
    AddedYearMonthNumber = Table.AddColumn(Dimension_Date, "Year Month Number",
    each [Calendar Year] * 100 + [Calendar Month Number], Int64.Type),
        AddedFiscalYearMonthNumber = Table.AddColumn(AddedYearMonthNumber, "Fiscal
    Year Month Number", each [Fiscal Year] * 100 + [Fiscal Month Number], Int64.Type)
    in
        AddedFiscalYearMonthNumber
```

```
let
    Source = Table.FromRows(
        Json.Document(Binary.Decompress(
            Binary.FromText("i45WMlSK1QGSBgYGcAaYHQsA ", BinaryEncoding.Base64),
            Compression.Deflate)
        ),
        let _t = ((type text) meta [Serialized.Text = true]) in type table [Scale = _t]
    ),
    #"Changed Type" = Table.TransformColumnTypes(Source,{{"Scale", Int64.Type}})
in
    #"Changed Type"
```

```
let
    Source = { 1, 1000, 1000000 },
    #"Converted to Table" = Table.FromList(Source, Splitter.SplitByNothing(), null,
null, ExtraValues.Error),
    #"Renamed Columns" = Table.RenameColumns(#"Converted to Table",{{"Column1",
"Scale"}}),
    #"Changed Type" = Table.TransformColumnTypes(#"Renamed Columns",{{"Scale",
Int64.Type}})
in
    #"Changed Type"
```

```
Table.FromRecords (
    {
        [Scale = 1],
        [Scale = 1000],
        [Scale = 1000000]
    },
    type table [Scale = Int64.Type]
)
```

// Untyped columns

```
#table (
    {"Quantity", "Unit Price"},
    {
        { 2, 3 },
        { 5, 7 }
    }
)
```

// Columns in data types defined

```
#table (
    type table [Quantity = Int64.Type, Unit Price = Currency.Type],
    {
        { 2, 3 },
        { 5, 7 }
    }
)
```

```
#table (
    type table [Scale = Int64.Type],
    {
        { 1 },
        { 1000 },
        { 1000000 }
    }
)
```

```
Unit Price Including Tax = Sale[Unit Price] * ( 1 + Sale[Tax Rate] / 100 )
```

```
// AM or PM, depending on time of the day
Upper = FORMAT ( NOW (), "AM/PM" )

// am or pm, depending on time of the day
Lower = FORMAT ( NOW (), "am/pm" )
```

```
DAX Formatted Length = LEN ( FORMAT ( 'Date'[Date], "dd-MM-yyyy" ) )
```

```
Buying Group First Three Characters = LEFT ( Customer[Buying Group], 3 )
```

```
Buying Group First Space Position = FIND ( " ", Customer[Buying Group] )
```

```
Buying Group First Space Position No Error = FIND ( " ", Customer[Buying Group], , 0 )
```

```
Buying Group First Space Position No Error = IFERROR ( FIND ( " ", Customer[Buying
Group] ), 0 )
```

```
Buying Group First Word = IFERROR ( LEFT ( Customer[Buying Group], FIND ( " ",
Customer[Buying Group] ) - 1 ), Customer[Buying Group] )
```

```
Buying Group First Word =
IFERROR (
    LEFT ( Customer[Buying Group], FIND ( " ", Customer[Buying Group] ) - 1
),
    Customer[Buying Group]
)
```

```
Number of T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( Customer[Buying Group], "T", "" ) )
```

// Using second SUBSTITUTE

```
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( SUBSTITUTE ( Customer[Buying Group], "t", "" ), "T", "" ) )
```

// Using LOWER

```
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( LOWER ( Customer[Buying Group] ), "t", "" ) )
```

// Using UPPER

```
Number of all T's =
LEN ( Customer[Buying Group] )
    - LEN ( SUBSTITUTE ( UPPER ( Customer[Buying Group] ), "T", "" ) )
```

```
Target Quantity =
LOOKUPVALUE (
    Target[Target Quantity],
    Target[Calendar Year], RELATED ( 'Date'[Calendar Year] ),
    Target[Bill To Customer], RELATED ( Customer[Bill To Customer] )
)
```

```
Target Quantity =
LOOKUPVALUE (
    Target[Target Quantity],
    Target[Calendar Year], RELATED ( 'Date'[Calendar Year] )
)
```

```
Price Category =
IF (
    'Stock Item'[Unit Price] < 100,
    "Low",
    IF (
        'Stock Item'[Unit Price] < 1000,
        "Medium",
        "High"
    )
)
```

```
Price Category Number =
IF (
    'Stock Item'[Price Category] = "Low",
    1,
    IF (
        'Stock Item'[Price Category] = "Medium",
        2,
        3
    )
)
```

```
Price Category Number = SWITCH ( 'Stock Item'[Price Category], "Low", 1, "Medium", 2, 3
)
```

// First, create Price Category Number

```
Price Category Number =
IF (
    'Stock Item'[Unit Price] < 100,
    1,
    IF (
        'Stock Item'[Unit Price] < 1000,
        2,
        3
    )
)
```

// Second, create Price Category

```
Price Category = SWITCH ( 'Stock Item'[Price Category Number], 1, "Low", 2, "Medium",
"High" )
```

// The order of creation does not matter in this case

```
Price Category =
IF (
    'Stock Item'[Unit Price] < 100,
    "Low",
    IF (
        'Stock Item'[Unit Price] < 1000,
        "Medium",
        "High"
    )
)

Price Category Number =
IF (
    'Stock Item'[Unit Price] < 100,
    1,
    IF (
        'Stock Item'[Unit Price] < 1000,
        2,
        3
    )
)
```

```
Five Price Categories =
IF (
    'Stock Item'[Unit Price] < 10,
    "Very Low",
    IF (
        'Stock Item'[Unit Price] < 100,
        "Low",
        IF (
            'Stock Item'[Unit Price] < 200,
            "Medium",
            IF (
                'Stock Item'[Unit Price] < 1000,
                "High",
                "Very High"
            )
        )
    )
)
```

```
Five Price Categories SWITCH =
SWITCH (
    TRUE (),
    'Stock Item'[Unit Price] < 10, "Very Low",
    'Stock Item'[Unit Price] < 100, "Low",
    'Stock Item'[Unit Price] < 200, "Medium",
    'Stock Item'[Unit Price] < 1000, "High",
    "Very High"
)
```

```
Unit Price (bins) =
IF (
    ISBLANK ( 'Stock Item'[Unit Price] ),
    BLANK (),
    IF (
        'Stock Item'[Unit Price] >= 0,
        ROUNDDOWN ( 'Stock Item'[Unit Price] / 105.5, 0 ) * 105.5,
        ROUNDUP ( 'Stock Item'[Unit Price] / 105.5, 0 ) * 105.5
    )
)
```

```
New Price =
IF (
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 )
        * 0.8
        < 20,
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 ),
    Sale[Unit Price]
        * ( 1 + Sale[Tax Rate] / 100 )
        * 0.8
)
```

```
Measure name =
VAR FirstVariable = … // DAX expression
VAR NthVariable = … // DAX expression
RETURN = … // DAX expression
```

```
New Price =
VAR TaxPct = Sale[Tax Rate] / 100
VAR PriceInclTax = Sale[Unit Price] * ( 1 + TaxPct )
VAR DiscountedPriceInclTax = PriceInclTax * 0.8
RETURN
    IF (
        DiscountedPriceInclTax < 20,
        PriceInclTax,
        DiscountedPriceInclTax
    )
```

```
TaxPct = Sale[Tax Rate] / 100

PriceInclTax = Sale[Unit Price] * ( 1 + Sale[TaxPct] )

DiscountedPriceInclTax = Sale[PriceInclTax] * 0.8

New Price =
IF (
    Sale[DiscountedPriceInclTax] < 20,
    Sale[PriceInclTax],
    Sale[DiscountedPriceInclTax]
)
```

```
// Returns different values for each row
Sales # = COUNTROWS ( RELATEDTABLE ( Sale ) )

// Returns different values only with bidirectional filtering enabled
Cities # = COUNTROWS ( RELATEDTABLE ( City ) )
```

```
Profit $ = 'Stock Item'[Recommended Retail Price] - 'Stock Item'[Unit Price]

Profit % = DIVIDE ( 'Stock Item'[Profit $], 'Stock Item'[Unit Price] )
```

```
Profit $ = 'Stock Item'[Profit %] * 'Stock Item'[Unit Price]
```

```
Sum of Scale Calculate = CALCULATE ( SUM ( Scale[Scale] ) )
```

```
#table (
    type table [ Scale = Int64.Type ],
    {}
)
```

```
Expensive Stock Items =
FILTER (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)
```

```
Expensive or Gray Stock Items =
FILTER (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] > 300,
        'Stock Item'[Color] = "Gray"
    )
)
```

```
// Different value for each row
Countrows Calculatetable = COUNTROWS ( CALCULATETABLE ( Sale ) )

// Same number for each row
Countrows Filter = COUNTROWS ( FILTER ( Sale, TRUE () ) )
```

```
Days in Month EARLIER =
COUNTROWS (
    FILTER (
        'Date',
        'Date'[Calendar Month Label]
            = EARLIER ( 'Date'[Calendar Month Label] )
    )
)
```

```
Days in Month VAR =
VAR CurrentMonth = 'Date'[Calendar Month Label]
RETURN
    COUNTROWS (
        FILTER (
            'Date',
            'Date'[Calendar Month Label] = CurrentMonth
        )
    )
```

```
Productive Salespeople =
FILTER (
    Employee,
    CALCULATE ( COUNTROWS ( Sale ) ) > 5000
)
```

```
All Stock Item = ALL ( 'Stock Item' )

All Color = ALL ( 'Stock Item'[Color] )

All Color, Buying Package = ALL ( 'Stock Item'[Color], 'Stock Item'[Buying Package] )
```

```
Customer Rows = COUNTROWS ( Customer )

Customer Rows Calculate = CALCULATE ( COUNTROWS ( Customer ) )

Customer Rows Calculate All = CALCULATE ( COUNTROWS ( ALL ( Customer ) ) )
```

```
Filter All Customer =
FILTER (
    ALL ( Customer ),
    NOT ISBLANK ( Customer[Customer Key] )
)
```

```
Days in Month ALLEXCEPT =
CALCULATE (
    COUNTROWS ( 'Date' ),
    ALLEXCEPT (
        'Date',
        'Date'[Calendar Month Label]
    )
)
```

```
Expensive Stock Items =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)
```

```
Expensive and Black Stock Items =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300,
    'Stock Item'[Color] = "Black"
)
```

```
Stock Items below $1 or above $1000 =
CALCULATETABLE (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] < 1,
        'Stock Item'[Unit Price] > 1000
    ),
    'Stock Item'[Unit Price] <> 0
)
```

```
Expensive or Gray Stock Item Wrong =
CALCULATETABLE (
    'Stock Item',
    OR (
        'Stock Item'[Unit Price] > 300,
        'Stock Item'[Color] = "Gray"
    )
)
```

```
// Boolean filter condition

Expensive Stock Item Boolean =
CALCULATETABLE (
    'Stock Item',
    'Stock Item'[Unit Price] > 300
)

// Table filter expression

Expensive Stock Item Filter All =
CALCULATETABLE (
    'Stock Item',
    FILTER (
        ALL ( 'Stock Item'[Unit Price] ),
        'Stock Item'[Unit Price] > 300
    )
)
```

```
Expensive or Gray Stock Items CalculateTable =
CALCULATETABLE (
    'Stock Item',
    FILTER (
        ALL (
            'Stock Item'[Unit Price],
            'Stock Item'[Color]
        ),
        OR (
            'Stock Item'[Unit Price] > 300,
            'Stock Item'[Color] = "Gray"
        )
    )
)
```

```
Sale Rows Calculate = CALCULATE ( COUNTROWS ( Sale ) )

Sale Rows Calculate VALUES = CALCULATE ( COUNTROWS ( VALUES ( Sale ) ) )
```

```
Employee Calculate = CALCULATE ( VALUES ( Employee[Employee] ) )
```

```
// 13 rows
Month Values = VALUES ( 'Date'[Month] )

// 12 rows
Month Distinct = DISTINCT ( 'Date'[Month] )
```

```
Month = SUMMARIZE ( 'Date', 'Date'[Month], 'Date'[Calendar Month Number] )
```

```
Month Sale =
SUMMARIZE (
    'Date',
    'Date'[Month],
    'Date'[Calendar Month Number],
    "Sale Rows", COUNTROWS ( Sale )
)
```

```
Month Sale =
SUMMARIZE (
    Sale,
    'Date'[Month],
    'Date'[Calendar Month Number],
    "Sale Rows", COUNTROWS ( Sale )
)
```

// Returns 198043439.450001

```
Summarize Sale Single =
SUMMARIZE (
    Sale,
    "Total Sales", SUM ( Sale[Total Including Tax] )
)
```

// Returns 197776428.010001

```
Summarize Date Single =
SUMMARIZE (
    'Date',
    "Total Sales", SUM ( Sale[Total Including Tax] )
)
```

```
// Returns 48 rows
Date Year Month =
SUMMARIZE (
    'Date',
    'Date'[Calendar Year],
    'Date'[Month],
    'Date'[Calendar Month Number]
)

// Returns 42 rows
Sale Year Month =
SUMMARIZE (
    Sale,
    'Date'[Calendar Year],
    'Date'[Month],
    'Date'[Calendar Month Number]
)
```

// 49 rows

```
Month Year = SUMMARIZECOLUMNS ( 'Date'[Calendar Year], 'Date'[Month] )
```

// 42 rows

```
Month Year =
SUMMARIZECOLUMNS (
    'Date'[Calendar Year],
    'Date'[Month],
    "Sale Rows", COUNTROWS ( Sale )
)
```

```
AddColumns Month Sale =
ADDCOLUMNS (
    ALL ( 'Date'[Month], 'Date'[Calendar Month Number] ),
    "Sale Rows", CALCULATE ( COUNTROWS ( Sale ) )
)
```

```
AddColumns YearMonthSequential =
ADDCOLUMNS (
    ALL (
        'Date'[Calendar Year],
        'Date'[Calendar Month Number]
    ),
    "Year Month Sequential",
    'Date'[Calendar Year] * 12 + 'Date'[Calendar Month Number]
)
```

```
SummarizeColumns YearMonthSequential =
SUMMARIZECOLUMNS (
    'Date'[Calendar Year],
    'Date'[Calendar Month Number],
    "Year Month Sequential",
    VALUES ( 'Date'[Calendar Year] ) * 12
        + VALUES ( 'Date'[Calendar Month Number] )
)
```

```
SelectColumns Calendar Year =
SELECTCOLUMNS (
    ALL ( 'Date'[Calendar Year], 'Date'[Month] ),
    "Year", 'Date'[Calendar Year]
)
```

```
Summarize SelectColumns Quarter =
SUMMARIZE (
    SELECTCOLUMNS (
        'Date',
        "Quarter", FORMAT ( 'Date'[Date], "\QQ" )
    ),
    [Quarter]
)
```

```
Top 3 Employees by Sales =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
)
```

```
Top 3 Employees by Name =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    Employee[Employee],
    ASC
)
```

```
Bottom 3 Employees by Sales =
TOPN (
    3,
    VALUES ( Employee[Employee] ),
    CALCULATE ( SUM ( Sale[Total Excluding Tax] ) ),
    ASC
)
```

```
Buying Package, Brand =
CROSSJOIN (
    VALUES ( 'Stock Item'[Buying Package] ),
    VALUES ( 'Stock Item'[Brand] )
)
```

```
Buying Package CrossJoin =
CROSSJOIN (
    VALUES ( 'Stock Item'[Buying Package] ),
    SELECTCOLUMNS (
        VALUES ( 'Stock Item'[Buying Package] ),
        "Buying Package 2", 'Stock Item'[Buying Package]
    )
)
```

```
Top 3 Employees per Calendar Year =
GENERATE (
    DISTINCT ( 'Date'[Calendar Year] ),
    TOPN (
        3,
        VALUES ( Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)
```

```
// Returns 123 rows
Top 3 Employees per Calendar Year Month =
GENERATE (
    ALLNOBLANKROW ( 'Date'[Calendar Year], 'Date'[Month] ),
    TOPN (
        3,
        SUMMARIZE ( RELATEDTABLE ( Sale ), Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)

// Returns 130 rows
Top 3 Employees per Calendar Year Month =
GENERATEALL (
    ALLNOBLANKROW ( 'Date'[Calendar Year], 'Date'[Month] ),
    TOPN (
        3,
        SUMMARIZE ( RELATEDTABLE ( Sale ), Employee[Employee] ),
        CALCULATE ( SUM ( Sale[Total Excluding Tax] ) )
    )
)
```

```
1 to 5 Fixed Decimal = GENERATESERIES ( CURRENCY ( 1 ), CURRENCY ( 5 ) )
```

```
1 to 3 May 2018 at 12-hour intervals =
GENERATESERIES (
    DATE ( 2018, 5, 1 ),
    DATE ( 2018, 5, 3 ),
    TIME ( 12, 0, 0 )
)
```

```
Uppercase Latin alphabet: =
SELECTCOLUMNS (
    GENERATESERIES ( 65, 90 ),
    "Letter", UNICHAR ( [Value] )
)
```

```
Calendar = CALENDAR ( MIN ( Sale[Invoice Date Key] ), MAX ( Sale[Invoice Date Key] ) )
```

```
Calendar =
CALENDAR (
    MIN (
        MIN ( Sale[Delivery Date Key] ),
        MIN ( Sale[Invoice Date Key] )
    ),
    MAX (
        MAX ( Sale[Delivery Date Key] ),
        MAX ( Sale[Invoice Date Key] )
    )
)
```

```
Calendar =
CALENDAR (
    DATE (
        YEAR (
            MIN (
                MIN ( Sale[Delivery Date Key] ),
                MIN ( Sale[Invoice Date Key] )
            )
        ),
        1,
        1
    ),
    DATE (
        YEAR (
            MAX (
                MAX ( Sale[Delivery Date Key] ),
                MAX ( Sale[Invoice Date Key] )
            )
        ),
        12,
        31
    )
)
```

```
One Row =
ROW (
    "Sale Rows", COUNTROWS ( Sale ),
    "Date Rows", COUNTROWS ( 'Date' )
)
```

```
Table1 =
UNION (
    ROW ( "Color", "Red", "Value 1", 1 ),
    ROW ( "Color", "Red", "Value 1", 2 ),
    ROW ( "Color", "Green", "Value 1", 3 ),
    ROW ( "Color", "Blue", "Value 1", CURRENCY ( 1 ) )
)
```

// Bridging table between Date and Target

```
Calendar Year =
DISTINCT (
    UNION (
        ALLNOBLANKROW ( Target[Calendar Year] ),
        ALLNOBLANKROW ( 'Date'[Calendar Year] )
    )
)
```

// Bridging table between Customer and Target

```
Bill To Customer =
DISTINCT (
    UNION (
        ALLNOBLANKROW ( Target[Bill To Customer] ),
        ALLNOBLANKROW ( Customer[Bill To Customer] )
    )
)
```

```
IntersectOneTwo =
INTERSECT (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

```
IntersectTwoOne =
INTERSECT (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

```
ExceptOneTwo =
EXCEPT (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

```
ExceptTwoOne =
EXCEPT (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),

    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

```
NaturalInnerJoinOneTwo =
NATURALINNERJOIN (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

```
NaturalInnerJoinTwoOne =
NATURALINNERJOIN (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

// Many side of a relationship

```
TableOne =
UNION (
    ROW ( "ColorOne", "Red", "Value 1", 1 ),
    ROW ( "ColorOne", "Red", "Value 1", 2 ),
    ROW ( "ColorOne", "Green", "Value 1", 3 ),
    ROW ( "ColorOne", "Blue", "Value 1", 1 )
)
```

// One side of a relationship

```
TableThree =
UNION (
    ROW ( "ColorThree", "Green", "Value 3", 3 ),
    ROW ( "ColorThree", "Blue", "Value 3", 1 ),
    ROW ( "ColorThree", "Yellow", "Value 3", 2 )
)
```

```
NaturalInnerJoin One Three = NATURALINNERJOIN ( TableOne, TableThree )
```

```
Aggregated Virtual Table =
ROW (
    "NumRows",
    COUNTROWS ( NATURALINNERJOIN ( TableOne, TableThree ) )
)
```

```
NaturalLeftOuterJoinOneTwo =
NATURALLEFTOUTERJOIN (
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    ),
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
)
```

```
NaturalLeftOuterJoinTwoOne =
NATURALLEFTOUTERJOIN (
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    ),
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
)
```

```
Scale DataTable =
DATATABLE (
        "Scale", INTEGER,
        {
                { 1 },
                { 1000 },
                { 1000000 }
        }
)
```

```
Enriched Scale =
DATATABLE (
    "Scale", INTEGER,
    "Description", STRING,
    {
        { 1, "Normal" },
        { 1000, "Thousands" },
        { 1000000, "Millions" }
    }
)
```

```
Scale Wrong =
DATATABLE (
        "Scale", INTEGER,
        {
                { 1 + 0 },
                { 1000 },
                { 1000000 }
        }
)
```

```
Anonymous table =
{
    ( 1, "a", DATE ( 2018, 5, 1 ) ),
    ( 2, "b", DATE ( 2018, 5, 23 ) )
}
```

```
Tables Var =
VAR Table1 =
    UNION (
        ROW ( "Color", "Red", "Value 1", 1 ),
        ROW ( "Color", "Red", "Value 1", 2 ),
        ROW ( "Color", "Green", "Value 1", 3 ),
        ROW ( "Color", "Blue", "Value 1", 1 )
    )
VAR Table2 =
    UNION (
        ROW ( "Color", "Green", "Value 2", 3 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Blue", "Value 2", 1 ),
        ROW ( "Color", "Yellow", "Value 2", 2 )
    )
RETURN
    NATURALINNERJOIN ( Table1, Table2 )
```

```
Buying Groups = VALUES ( Customer[Buying Group] )
```

```
Wingtip Toys =
CALCULATETABLE (
    VALUES ( Customer[Buying Group] ),
    Customer[Buying Group] = "Wingtip Toys"
)
```

```
Wingtip Toys VAR =
VAR BuyingGroups =
    VALUES ( Customer[Buying Group] )
RETURN
    CALCULATETABLE (
        BuyingGroups,
        Customer[Buying Group] = "Wingtip Toys"
    )
```

```
Calendar =
VAR Days =
    CALENDAR ( "2018-1-1", "2020-12-31" )
RETURN
    GENERATE (
        Days,
        VAR BaseDate = [Date]
        VAR MonthName = FORMAT ( BaseDate, "MMM" )
        VAR MonthNumber = MONTH ( BaseDate )
        VAR BaseYear = YEAR ( BaseDate )
        RETURN
            ROW (
                "Month", MonthName,
                "MonthNo", MonthNumber,
                "Year", BaseYear
            )
    )
```

```
Net Profit % = DIVIDE ( Example[Net Profit], Example[Gross Profit] )
```

```
Net Profit % = DIVIDE ( Example[Net Profit], Example[Gross Profit] )
```

```
Total Net Profit = SUM ( Example[Net Profit] )

Total Net Profit = SUMX ( Example, Example[Net Profit] )
```

```
Gross Sales = SUMX ( Sale, Sale[Quantity] * Sale[Unit Price] )
```

```
Full Price Sales =
SUMX (
    Sale,
    Sale[Quantity] * RELATED ( 'Stock Item'[Recommended Retail Price] )
)
```

```
Net Profit % =
DIVIDE (
    SUM ( Example[Net Profit] ),
    SUM ( Example[Gross Profit] )
)
```

```
// Create these two measures first

Total Net Profit = SUM ( Example[Net Profit] )
Total Gross Profit = SUM ( Example[Gross Profit] )

// Create this measure last

Net Profit % = DIVIDE ( [Total Net Profit], [Total Gross Profit] )
```

```
Years and Months with Sales =
COUNTROWS (
    SUMMARIZE (
        Sale,
        'Date'[Calendar Year],
        'Date'[Month]
    )
)
```

```
CountX Invoice Delivery =
COUNTX (
    Sale,
    Sale[Invoice Date Key] + Sale[Delivery Date Key]
)
```

```
CountBlank DeliveryDate = COUNTBLANK ( Sale[Delivery Date Key] )
```

```
Distinct StockItems = DISTINCTCOUNT( 'Stock Item'[Stock Item Key] )

Distinct StockItems = COUNTROWS ( 'Stock Item' )
```

```
// Calculated column

Sum Column = SUM ( Scale[Scale] )

// Measure

Sum Measure = SUM ( Scale[Scale] )
```

```
// Create the Total Profit measure first
Total Profit = SUM ( Sale[Profit] )

// Create the All-time Profit measure that references Total Profit
All-time Profit = CALCULATE ( [Total Profit], ALL ( 'Date' ) )
```

```
January Profit Wrong = CALCULATE ( [Total Profit], 'Date'[Month] = "January" )
```

// Including conditions for both columns

```
January Profit =
CALCULATE (
    [Total Profit],
    'Date'[Month] = "January",
    'Date'[Calendar Month Number] = 1
)
```

// Boolean expressions internally converted to table filters

```
January Profit =
CALCULATE (
    [Total Profit],
    FILTER (
        ALL ( 'Date'[Month] ),
        'Date'[Month] = "January"
    ),
    FILTER (
        ALL ( 'Date'[Calendar Month Number] ),
        'Date'[Calendar Month Number] = 1
    )
)
```

// Alternative approach: iterating over a table that includes both columns

```
January Profit =
CALCULATE (
    [Total Profit],
    FILTER (
        ALL ( 'Date'[Month], 'Date'[Calendar Month Number] ),
        'Date'[Month] = "January"
    )
)
```

```
Profit AllSelected = CALCULATE ( [Total Profit], ALLSELECTED ( 'Date' ) )
```

```
Profit AllSelected = CALCULATE ( [Total Profit], ALLSELECTED () )
```

```
Profit YTD = CALCULATE ( [Total Profit], DATESYTD ( 'Date'[Date] ) )
```

```
Profit FYTD = CALCULATE ( [Total Profit], DATESYTD ( 'Date'[Date], "30-6" ) )
```

```
Profit FYTD = TOTALYTD ( [Total Profit], 'Date'[Date], "30-6" )
```

```
Opening Profit = OPENINGBALANCEMONTH ( [Total Profit], 'Date'[Date] )

Closing Profit = CLOSINGBALANCEMONTH ( [Total Profit], 'Date'[Date] )
```

```
Closing Profit = CALCULATE ( [Total Profit], ENDOFMONTH ( 'Date'[Date] ) )
```

```
Profit Last Month DateAdd =
CALCULATE (
    [Total Profit],
    DATEADD ( 'Date'[Date], -1, MONTH )
)

Profit Last Month ParallelPeriod =
CALCULATE (
    [Total Profit],
    PARALLELPERIOD ( 'Date'[Date], -1, MONTH )
)
```

```
Opening Profit DateAdd =
CALCULATE (
    [Total Profit],
    DATEADD ( STARTOFMONTH ( 'Date'[Date] ), -1, DAY )
)
```

```
Opening Profit DateAdd Wrong =
CALCULATE (
    [Total Profit],
    STARTOFMONTH ( DATEADD ( 'Date'[Date], -1, DAY ) )
)
```

```
// Same as FIRSTDATE
FILTER (
    VALUES ( 'Date'[Date] ),
    'Date'[Date] = MIN ( 'Date'[Date] )
)
// Same as LASTDATE
FILTER (
    VALUES ( 'Date'[Date] ),
    'Date'[Date] = MAX ( 'Date'[Date] )
)
```

```
Profit Custom Period =
CALCULATE (
    [Total Profit],
    DATESBETWEEN (
        'Date'[Date],
        DATE ( 2013, 10, 6 ),
        DATE ( 2014, 4, 5 )
    )
)
```

```
Rolling Monthly Profit =
CALCULATE (
    [Total Profit],
    DATESINPERIOD (
        'Date'[Date],
        MAX ( 'Date'[Date] ),
        -1,
        MONTH
    )
)
```

```
Total Profit by Invoice Date =
CALCULATE (
    [Total Profit],
    USERELATIONSHIP ( 'Date'[Date], Sale[Invoice Date Key] )
)
```

```
MyParameter Value =
IF (
    HASONEVALUE ( MyParameter[MyParameter] ),
    VALUES ( MyParameter[MyParameter] ),
    5
)
```

```
Profit Scaled = [Total Profit] / SELECTEDVALUE ( Scale[Scale], 1 )
```

```
First Profit =
CALCULATE (
    [Total Profit],
    FIRSTNONBLANK (
        'Date'[Date],
        [Total Profit]
    )
)
```

```
Total Target Quantity = SUM ( Target[Target Quantity] )
Total Target Amount = SUM ( Target[Target Amount Excluding Tax] )
Target Quantity Intersect =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( Target[Calendar Year] ),
        VALUES ( 'Date'[Calendar Year] )
    )
)
```

```
Target Quantity Intersect Wrong =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( 'Date'[Calendar Year] ),
        VALUES ( Target[Calendar Year] )
    )
)
```

```
Target Quantity Intersect Values =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        VALUES ( Target[Calendar Year] ),
        VALUES ( 'Date'[Calendar Year] )
    ),
    INTERSECT (
        VALUES ( Target[Bill To Customer] ),
        VALUES ( Customer[Bill To Customer] )
    )
)

Target Quantity Intersect Summarize =
CALCULATE (
    [Total Target Quantity],
    INTERSECT (
        ALL ( Target[Calendar Year], Target[Bill To Customer] ),
        SUMMARIZE ( Sale, 'Date'[Calendar Year], Customer[Bill To Customer] )
    )
)
```

```
Target Quantity TreatAs =
CALCULATE (
    [Total Target Quantity],
    TREATAS (
        SUMMARIZE ( Sale, 'Date'[Calendar Year], Customer[Bill To Customer] ),
        Target[Calendar Year],
        Target[Bill To Customer]
    )
)
```

```
Total Profit YoY% =
IF(
        ISFILTERED('Date'[Date]),
        ERROR("Time intelligence quick measures can only be grouped or filtered by the
Power BI-provided date hierarchy or primary date column."),
        VAR __PREV_YEAR = CALCULATE([Total Profit], DATEADD('Date'[Date].[Date], -1,
YEAR))
        RETURN
                DIVIDE([Total Profit] - __PREV_YEAR, __PREV_YEAR)
)
```

```
MyParameter Value = SELECTEDVALUE(MyParameter[MyParameter], 5)
```

```
Discounted Profit = [Total Profit] * ( 1 - [MyParameter Value] / 100 )
```

```
Target Is Valid =
NOT (
    -- Check the Date table granularity
    ISFILTERED ( 'Date'[Calendar Month Label] )
        || ISFILTERED ( 'Date'[Date] )
        || ISFILTERED ( 'Date'[Day] )
        || ISFILTERED ( 'Date'[Fiscal Month Label] )
        || ISFILTERED ( 'Date'[Fiscal Year Label] )
        || ISFILTERED ( 'Date'[Month] )
        || ISFILTERED ( 'Date'[Short Month] )
    -- Check the Customer table granularity
        || ISFILTERED ( Customer[Customer] )
        || ISFILTERED ( Customer[Postal Code] )
        || ISFILTERED ( Customer[Primary Contact] )
    -- Check other tables
        || ISCROSSFILTERED ( City )
        || ISCROSSFILTERED ( Employee )
        || ISCROSSFILTERED ( 'Stock Item' )
)
```

```
Target Amount = IF ( [Target Is Valid], [Total Target Amount] )
```

```
Actual to Target = [Total Actual Amount] - [Target Amount]
```

```
Actual to Target % = DIVIDE ( [Actual to Target], [Target Amount] )

Actual to Target % = DIVIDE ( [Total Actual Amount] - [Target Amount], [Target Amount] )
```

```
Level 1 = PATHITEM ( PC[Path], 1 )

Level 2 = PATHITEM ( PC[Path], 2 )

Level 3 = PATHITEM ( PC[Path], 3 )
```

```
Level 1 = PATHITEMREVERSE ( PC[Path], 1 )

Level 2 = PATHITEMREVERSE ( PC[Path], 2 )

Level 3 = PATHITEMREVERSE ( PC[Path], 3 )
```

```
Colors <- dataset$Color
Amounts <- dataset$'Total Actual Amount' / 1000000
par ( mar = c ( 2, 6, 2, 2 ) + 0.1 )
barplot ( Amounts, horiz = TRUE, names.arg = Colors, las = 1 )
```

```
{
    "name": "ER778",
    "dataColors": ["#717171", "#F1595E", "#79C369", "#589AD3", "#F9A659", "#D77FB2",
"#CE7058", "#9E67AB"],
    "visualStyles": {
        "*": {
            "*": {
                "*": [{
                    "fontSize": 10,
                    "color": {
                        "solid": {
                            "color": "#000000"
                        }
                    }
                }]
            }
        }
    }
}
```

```
Total Quantity =
CALCULATE (
    SUM ( Sale[Quantity] ),
    TREATAS ( VALUES ( 'Stock Item'[Stock Item Key] ), Sale[Stock Item Key] )
)
```

```
Total Quantity =
CALCULATE (
    SUM ( Sale[Quantity] ),
    INTERSECT( Sale, 'Stock Item' )
)
```

```
Quantity by Invoice Date =
CALCULATE (
    SUM ( Sale[Quantity] ),
    USERELATIONSHIP ( 'Date'[Date], Sale[Invoice Date Key] )
)
```

```
// Column of type Date
[Date] < DATE(2018,05,23)

// Column of type Text
[Calendar Year Label] = "Value"

// Column of type True/False
[Is Latest]
```

```
Number of years column = DISTINCTCOUNT ( 'Date'[Calendar Year] )
```

```
Number of years measure = DISTINCTCOUNT ( 'Date'[Calendar Year] )
```