

Learn

Power Query

A low-code approach to connect and transform data from multiple sources for Power BI and Excel



Packt>

www.packt.com

Linda Foulkes and Warren Sparrow

Learn Power Query

A low-code approach to connect and transform data from multiple sources for Power BI and Excel

Linda Foulkes

Warren Sparrow



BIRMINGHAM—MUMBAI

Learn Power Query

Copyright © 2020 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Kunal Chaudhari

Acquisition Editor: Karan Gupta

Senior Editor: Nitee Shetty

Content Development Editor: Tiksha Lad

Technical Editor: Pradeep Sahu

Copy Editor: Safis Editing

Project Coordinator: Deeksha Thakkar

Proofreader: Safis Editing

Indexer: Manju Arasan

Production Designer: Nilesh Mohite

First published: July 2020

Production reference: 1160720

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-83921-971-9

www.packt.com

To my special person:

It has certainly been a challenging, but extremely rewarding, phase of my life authoring this book. I would like to thank my special person, Mick, for his inspiration, understanding, and pushing me to meet deadlines. You have contributed greatly to my life, and I appreciate you immensely. I love you.

To my co-author:

It is not often you find a co-author who will check all the boxes and turn up for a Zoom meeting at 6 a.m. in his tie. Warren, thank you for working alongside me, motivating me, and pouring your knowledge and expertise into this book, and also, for giving me a breather near the end to complete Learn Microsoft Office 2019 (which I wrote alongside Power Query).

*I have enjoyed our long chats, your humor, as well as having your support throughout the process. Congratulations on your first publication
– we did it!*

-Linda Foulkes

*This book is dedicated to Michelle, my wife, my best friend, and my love,
and to my two beautiful daughters, Danica and Jordan.*

-Warren Sparrow



Packt . com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [packt . com](http://packt.com) and, as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub . com](mailto:customercare@packtpub.com) for more details.

At [www . packt . com](http://www.packt.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Linda Foulkes is a Microsoft Office Master Trainer, Certified Educator, and Microsoft Innovative Expert Educator and Trainer with an educational and corporate background spanning over 25 years. As well as being certified as an IT trainer, Linda represented South Africa at the Microsoft Global Forum in Redmond in 2015. She has certified and coached students to compete at the Microsoft Office Specialist Championships in Texas. She has also published *Learn Microsoft Office 2019* through Packt. Linda has also presented at conferences and conducted webinars for SchoolNet SA, and hosted TeachMeets and MicrosoftMeets. She has a keen interest in e-learning and has developed e-learning paths and content for the Microsoft Office suite of programs.

Warren Sparrow is a Microsoft, Adobe, and National Geographic Certified Educator. He has been a Microsoft Innovative Expert Educator for the last 6 years, and a fellow in 2015. Warren is regularly invited to be a guest speaker at different global educational events, including the annual Microsoft conference on Office 365 in 2015. His core focus is long-term strategy, training, and technology implementation. He has provided training and development in both the education and corporate sectors. He also played an advisory role to the Western Cape Education Department (South Africa), giving assistance on the roll-out of hardware and software, as well as the training of educators and the implementation of technology in school classrooms.

Acknowledgements

Both Warren and I have enjoyed the process of writing this book immensely and would like to thank the Packt team for their dedication to this process. Without the support of a strong team toward the end of chapter writing, the review process, meeting deadlines, and the submission of final proofs would have been a near-impossible task to achieve. We would like to thank all those involved, but extend a sincere, special gratitude to Karan Gupta, Acquisition Editor, who devoted 100% of his time to the initial stages of the book and was always available for support throughout, to the Content Development Editor, Tiksha Lad, for her encouragement and professional contribution, and to Prajakta Naik, for checking that we were on course to meet deadlines and for offering support where needed. Last, but by no means least, we would like to thank our reviewer, Vishwanath Muzumdar, for being our critic and advising us accordingly in order to enable us to finish this great resource.

About the reviewer

Vishwanath Muzumdar has more than 8 years' experience in information technology consulting, business analysis, business development, and business process management in the business intelligence space.

He is an MS Power BI developer (champion) in the creation of powerful visual reporting for clients. His goal is to utilize his strong prioritization skills, analytical ability, team management skills, and expertise in the Microsoft Power BI reporting tool in order to achieve organizational objectives.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors . packtpub . com](https://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

Section 1: Overview of Power Pivot and Power Query

1

Installation and Setup

Technical requirements	4	Power Query Office versions and differences	12
Introducing Power Pivot	4	Installing Power Query in Office 2013 and 2010	15
Power Pivot Office versions and differences	5	Activating my Power Query/Pivot again	16
Introduction to Power BI	6	Launching Power Query, Power Pivot, and Power BI	18
Power BI versions and differences	6	Launching Power Query within Excel	18
Mac/Apple	10	Accessing Power Query from Power BI	18
Introduction to Power Query	11	Summary	20
Features of Power Query	11		

2

Power Pivot Basics, Inadequacies, and Data Management

Technical requirements	22	Creating relationships between tables	37
Creating a pivot table	23	Power Query to the rescue	39
Creating a Power Pivot	30	Creating a calculated column	39
Creating a table in Excel	30	Creating a calculated field	44
Adding tables to the data model	34	Creating a Power Pivot table	46

Shortcomings of Power Pivot	47	Problem 3 – calculated fields	48
Problem 1 – selecting multiple items	47	Problem 4 – Microsoft Office versions	48
Problem 2 – Power Pivot preview	48	Summary	49

3

Introduction to the Power Query Interface

Technical requirements	52	Discovering the Load To... options	79
The Power Query window and its elements	52	Changing the default custom load settings	80
The main ribbon and tabs	53	Loading queries to the worksheet manually	83
The navigation pane or the Queries list	54	Data profiling tips	89
Data table preview	57	Column profile	92
The Query Settings pane	59	Column quality	92
Working with APPLIED STEPS	60	Column distribution	94
Investigating the View settings	62	Summary	94
Using Advanced Editor	67		
Creating a basic power query	76		

4

Connecting to Various Data Sources Using Get & Transform

Technical requirements	96	Understanding custom connections	117
A brief introduction to databases	96	Connecting from Workbook	120
Connecting from a table or range	99	Connecting from a folder	124
Connecting data to the web	103	Exploring data source settings	133
Connecting from a relational database	111	From Excel	133
Connecting through Excel's Get & Transform tool	112	Summary	135
Connecting through Power BI	114		

Section 2: Power Query Data Transformations

5

Transforming Power Query Data

Technical requirements	140	Replacing null values	162
Turning data with the unpivot and pivot tools	140	Working with the header row	163
Refreshing data	144	Splitting columns	164
Basic column and row tools	147	Merging and appending tools	167
Removing columns	147	Merging columns using combine	167
Removing top or bottom rows	148	Merging text and values into one column	170
Using the index column	149	Appending (combining) tables	172
Creating a conditional column with the if...then...else statement	150	Grouping data	175
Filtering data using the And/Or conditions	153	Working with extraction tools	177
Creating single-criteria filters	156	Extracting an age from a date	177
Creating dynamic multiple-criterion filters	158	Extracting columns	178
Removing duplicate rows	161	Using the extract column features	179
		Summary	182

6

Advanced Power Queries and Functions

Technical requirements	184	Understanding the Index and Modulo functions	201
Writing an IF function in Power Query	184	Beginning with the modulo function	201
Creating a parameter table for queries	191	Understanding index functions	204
Changing the monthly data source	196	Appending multiple files	214
		Appending multiple tabs	220
		Summary	223

7

Automating Reports in Power Query

Technical requirements	226	Understanding the Power BI refresh types	235
Understanding the storage modes and dataset types	226	Learning how to refresh a OneDrive connection	236
Viewing the Power BI Desktop Storage mode setting	227	Viewing and performing a OneDrive refresh	241
Choosing the Import storage mode setting	231	Setting a scheduled refresh	243
Looking at where Power BI stores data	234	Incremental refresh	244
Investigating whether Microsoft SQL Server Analysis Services is running	234	Automatic page refresh	245
		Dataflow refresh	246
		Summary	246

8

Creating Dashboards with Power Query

Technical requirements	248	Selecting data visualization, a dataset, and an appropriate chart	272
Creating a basic power pivot and PivotChart	248	Saving, publishing, and sharing a dashboard	279
Using Power BI to collect and connect data	252	Sharing a dashboard	282
Combining files	258	Best practices	292
Using Power BI to add data to a data model	267	Summary	293

Section 3: Learning M

9

Working with M

Technical requirements	298	Using #shared to return library functions	301
The beginnings of M	298	Text data types	301
Understanding the M syntax and learning how to write M	299	Number data types	302

Lists	304	Searching for relevant data	307
Records	305		
Table data types	306	Importing a CSV file using M	308
		Summary	310

10

Examples of M Usage

Technical requirements	314	Management Studio	329
Merging using the concatenate formula	314	Using parameters	335
Data type conversions	321	Parameterizing a data source	335
Setting up a SQL server	322	Using parameters in the Data view	348
Installing SQL Server		Summary	349

11

Creating a Basic Custom Function

Technical requirements	352	Creating the function manually	355
Creating a function manually using M	352	Testing the parameter function	358
Changing the file path of the query to a local path	354	Creating a date/time column using three M functions	364
		Summary	373

12

Differences Between DAX and M

Technical requirements	376	and storage engine	383
Learning about the DAX and M functionality	376	Creating a calculated column	384
Constructing DAX syntax	377	Creating calculated measures	385
Constructing DAX formulas in Excel	380	Using quick measures	386
Using IntelliSense	380	Formulating a DAX measure from scratch	389
Creating a DAX formula	382	Organizing measures	395
Understanding the DAX formula		Summary	396

Other Books You May Enjoy

Preface

Power Query is a data connection technology that allows you to connect, combine, and refine data sources to meet all of your analysis requirements. This book will take you on a journey through Power Query, starting with the shortcomings of other tools regarding data analysis and management. Then, we will delve into the Power Query interface, looking at how to connect, combine, and refine data with really powerful query tools and learning how to use the Power Query M formula language, which opens up a whole new world of data mashup. We will complete our journey by creating dashboards and multi-dimensional reports in Power Query.

Who this book is for

This book would suit professional business analysts, data analysts, BI professionals, and Excel users who wish to take their skills to the next level by learning how to collect, combine, and transform data into insights using Power Query.

What this book covers

Chapter 1, Installation and Setup. Power Query is now integrated into all the data analysis or business intelligence tools from Microsoft such as Excel, Analysis Services, and Power BI. It allows users to discover, combine, and refine their data from various sources. In this chapter, you will learn how to install and access these tools across multiple office versions.

Chapter 2, Power Pivot Basics, Inadequacies, and Data Management, introduces you to the shortcomings of Power Pivot in handling complex data and offers Power Query as a solution to retrieve, extract, and reshape data. You will be taken through an example to demonstrate the differences between Power Query and Power Pivot and to learn how to convert worksheet data into a table.

Chapter 3, Introduction to the Power Query Interface, introduces you to the Power Query interface and takes you on a journey through its tabs, creating a basic Power Query query and visiting the **View** tab in Power BI to set data profiling options, as well as discovering how to send data back to an Excel workbook.

Chapter 4, Connecting to Various Data Sources Using Get & Transform, explains how to connect to numerous data sources using the **Get & Transform** tool, known as Power Query, and investigate data source settings.

Chapter 5, Transforming Power Query Data, covers how to reshape tabular data, including altering rows, columns, and tables using a multitude of Power Query tools.

Chapter 6, Advanced Power Queries and Functions, concentrates on the more advanced queries and functions in Power Query, such as the `IF`, `INDEX`, and `MODULO` functions. You will learn to create parameters to alter query paths and append multiple files and sheet tabs.

Chapter 7, Automating Reports in Power Query, goes through the options provided by Power Query to streamline and automate reports from multiple sources. In this chapter, we will look at creating a report from multiple files in a folder to a single dataset, which will update when new data is added to the Power Query data folder.

Chapter 8, Creating Dashboards with Power Query, looks at dashboards, which are a business-intelligent, single-canvas page that allows the user to tell a story through various visualizations created from table data to highlight important data points for an organization. In this chapter, you will learn how to create a dashboard from connected data, select a visualization type, and publish and customize the dashboard. We will also cover multi-dimensional reporting.

Chapter 9, Working with M introduces the Power Query M language and explains how to use and write the syntax, including steps to reveal a list of functions and definitions. This chapter will start by explaining how M got its name and how Microsoft tried to change it. We will also look at the structure and syntax of M. All programming languages have a specific structure and once you master the structure of M, it becomes much easier to understand and use. We will look at the main data types and functions and provide a walkthrough demonstration of how to use each of these data types, before looking at how to import a CSV file using M.

Chapter 10, Examples of M Usage, concentrates on a few examples of using M, including the concatenate function, which first compares the difference between formulas in Excel and Power BI before looking at the ampersand operator (&) and how it can be used. This chapter examines how `Text.From` and `Text.Combine` can be used to join and concatenate different strings, dates, and columns. There is an extensive section on how to set up your own free and legal SQL server for you to use for non-commercial purposes. It has full functionality, and we will also cover how to import the AdventureWorks databases into SQL for us to be able to use them as a resource. Lastly, this chapter concentrates on **parameters** and how they can be used effectively in filtering data sources, adding parameters to control statements that allow us to filter according to different dates. We will continue by adding parameters to order objects and columns in ascending and descending order, before looking at how we can make these changes in Power BI's **Data** view.

Chapter 11, Creating a Basic Custom Function, will take you through the steps to create functions manually using M in Power Query, as well as how create a date and time column using functions.

Chapter 12, Differences Between DAX and M, looks at the differences between M, which is the mashup functional language of Power Query and is used to query numerous data sources, and DAX, which allows functions to work on data stored in tables, much like Excel. In this chapter, you will learn about the differences between the two languages by working through examples and learning how to create calculated measures.

To get the most out of this book

We assume that you have a solid working knowledge of Excel up to an advanced level and that you can construct and troubleshoot formulas and functions. You should be familiar with all that Excel has to offer and be at a stage to advance your skills and want to learn more about data analysis and data management as an effective business solution. Refer to the following table for the software used in the book:

Software covered in this book	OS requirements
Office 2016 or above Office 2010 is supported but needs additional add-in software	Windows
Power Query, Power Pivot, Power BI Desktop/ Power BI Online (if you have a subscription)	Windows
SQL Server	Windows SQL Server
Microsoft SQL Server Management Studio	Windows
Microsoft SQL Server Administration Manager	Windows
SQL – AdventureWorks	https://github.com/Microsoft/sql-server-samples/blob/master/license.txt

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to copy/pasting of code.

Download the example code files

You can download the example code files for this book from your account at www.packt.com. If you purchased this book elsewhere, you can visit www.packtpub.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packt.com.
2. Select the **Support** tab.
3. Click on **Code Downloads**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Learn-Power-Query/>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Code in Action

Code in Action videos for this book can be viewed at <https://bit.ly/2ZeYxfb>.

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here:

https://static.packt-cdn.com/downloads/9781839219719_ColorImages.pdf

Conventions used

There are a number of text conventions used throughout this book.

Code in text: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The basic M syntax is relatively straightforward, provided that we follow the correct structure with `let` and `in`."

A block of code is set as follows:

```
let
    Source = ""
in
    Source
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "The variable name is also the name of **APPLIED STEPS** found on the right side of the screen."

Tips or important notes
Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packt.com.

Section 1: Overview of Power Pivot and Power Query

This is the first part of this *Learn Power Query* book. It will take you through the steps to access Power Query across different versions of Microsoft Excel and to install the Power BI engine. We will look at Power Pivot and its shortcomings and how Excel users find it difficult to clean their data in Power Pivot. We will solve this problem by introducing Power Query through an example. We will discuss the use of Power Query, Power Pivot, and Power BI, and then delve into the basics of Power Query and Power Pivot.

This section comprises the following chapters:

- *Chapter 1, Installation and Setup*
- *Chapter 2, Power Pivot Basics, Inadequacies, and Data Management*
- *Chapter 3, Introduction to the Power Query Interface*
- *Chapter 4, Connecting to Various Data Sources Using Get & Transform*

1 Installation and Setup

There are many reasons why you would need to use Power Query, but here is a short summary of the reasons why Power Query is a very necessary step if you wish to prepare data for analysis, manipulation, or visualization:

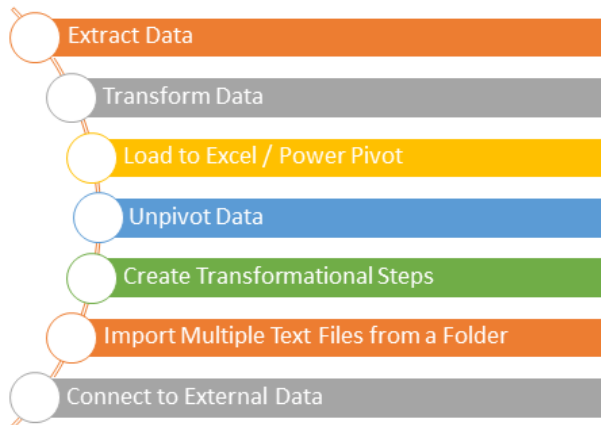


Figure 1.1 – Features of Power Query

Power Query is now integrated into all the data analysis and business intelligence tools from Microsoft, such as Excel, Analysis Services, and Power BI. It allows users to discover, combine, and refine their data from various sources.

In this chapter, we will take you through the steps you will need to follow to access Power Query across different versions of Microsoft Excel and install the Power BI engine. We will look at Power Pivot and its shortcomings and how Excel users find it difficult to clean their data in Power Pivot. We will solve this problem by introducing Power Query through an example. After that, we will discuss the use of Power Query, Power Pivot, and Power BI, and then delve into Power Query and Power Pivot basics.

In this chapter, we're going to cover the following main topics:

- Introduction to Power Pivot, Power BI, and Power Query
- The respective Office versions and differences between the Power software
- Installation guide across Excel versions
- Launching Power Query, Power Pivot, and Power BI

Technical requirements

It is imperative that you are able to locate an application from an operating system.

You will require intermediate knowledge of Microsoft Excel, including working with rows, columns, worksheets, and workbooks, and have a basic understanding of formulae and functions. It is also important that you know how to create a chart and modify its elements.

The GitHub URL for this chapter is <https://github.com/PacktPublishing/Learn-Power-Query/>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=cxKvtorqPlQ&list=PLeLcwrwLe186O_GJEZs47WaZXwZjwTN83&index=2&t=4s.

Introducing Power Pivot

It is important that you understand the term *data model* (see https://en.wikipedia.org/wiki/Data_modeling) before delving into what Power Pivot is about. A data model is where two or more tables are linked together by a common field or column. If you have already worked with Microsoft Access databases, then you will have an understanding of table relationships. Linking tables from one or more sources to a single data source is known as a data model:

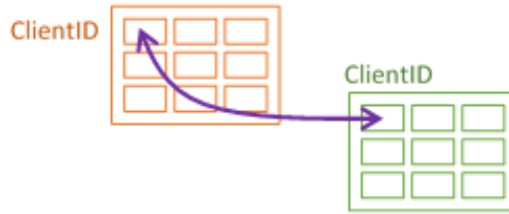


Figure 1.2 – Representation of linked tables

Power Pivot is a part of the Power BI family; it is considered the brain of the family as its purpose is to model, crunch, create calculations, and analyze. Using the analogy of a vehicle, it is the engine in the data model that hosts all of the data. It can digest large sets of data that reside in the multi-table data model it creates and can then be used as a data source, for example, to create pivot tables. When working with data on a worksheet in Excel, you can use Power Pivot to create a data model and subsequently create a link between tables to create extensive relationships and build simple or complex calculations all within the Excel environment. Power Pivot models data, sets relationships, calculates additional columns, creates measures, sets KPIs, and can also produce a cube.

Power Pivot Office versions and differences

We will first list the compatible Office versions and then discuss the differences between them.

Power Pivot Office 2019 (Office 365), 2016, and 2013

A bit of good news is that previously, Power Pivot was only available with Office Pro Plus 2013/2016 versions, but recently Microsoft have made Power Pivot available in Office 365 Home, Office 365 Personal, Office 365 Business Essentials, Office 365 Business, Office 365 Business Premium, and Office 365 Enterprise E1.

Power Pivot is already installed on these systems by default, but sometimes, it needs to be *activated* so that you can see it. For these versions, you can follow the steps provided in the *Activating my Power Query/Pivot again* section.

Power Pivot Office 2010

The Power Pivot add-in was not bundled with Office 2010, but you can download this for free from <https://www.microsoft.com/en-us/download/details.aspx?id=43348>. Power Pivot for Office 2010 had two different versions, aptly named version 1 and version 2. They were written as plugins for the development of SQL Server 2012. When you go to the download site, it will state that you are downloading **Microsoft® SQL Server® 2012 SP2 PowerPivot for Microsoft Excel® 2010**. Do not worry if you do not have SQL Server; this is the version that you need for Excel 2010. You will notice that the file size is relatively large when you download the add-in.

Introduction to Power BI

Power BI offers a cloud platform (Software as a Service – SaaS) experience that empowers businesses to service themselves with all their intelligence needs. The huge benefit of this is being able to handle millions of rows of data with ease – you can model and analyze data by defining relationships. Its most wonderful feature is that it allows you to define a formula once and then manipulate data using the same formula!

Power BI consists of Power Query, Power Pivot, and Power View and allows you to present large, complex data in meaningful and interesting ways by creating reports and dashboards. It is available in the cloud, which is where users upload and share with other users, allowing natural language queries to be performed on data models. Its standalone version is free of charge and is named Power BI for Desktop Applications. It includes all three apps (Power Query, Power View, and Power Pivot). Power View allows you to create interactive visualizations and consists of an interface where you drag-and-drop items to create custom outputs from data.

Power BI versions and differences

We will first list the compatible versions for Windows and macOS and then discuss the differences between the available versions.

Windows

There are three different versions of Power BI that you can download. Let's go over them now.

Power BI Desktop/Free

Power BI Desktop, which is sometimes referred to as Power BI Free, is intended for small-to medium-sized businesses. It allows you to connect to just over 70 data sources, publish to the web, and export your data to Excel.

You can download the file from <https://www.microsoft.com/en-us/download/details.aspx?id=58494> or from the Microsoft Store: <https://aka.ms/pbidesktopstore>.

Note

There are a couple of minimum system requirements that most computers will already have, such as Windows 7 or newer, Internet Explorer 10 as a minimum, 1 GB RAM, .Net 4.5, and a CPU speed of at least 1 GHZ. But the one surprise that you might have is that your standard screen size of 1,024 x 768 or 1,280 x 800 is actually too small for Power BI. It is recommended that you have a resolution of at least 1,440 x 900 or 1,600 x 900 as certain controls display beyond these resolutions.

Once again, you will be given the option to download the software for either a 32- or 64-bit system.

Please note that this is an application that you will need to install and that it might be different to the version of Office that you have.

Download the relevant 32- or 64-bit file that you require for the installation of Power BI Desktop. Run and install the downloaded file. When you get to the window that says that you are about to install Power BI, have a look at the privacy statement to see what data they collect while you are using Power BI. You have the option to opt out of this. Click on the appropriate place in the window to learn how to do this.

Select the destination folder where you would like to install Power BI. Personally, I would leave it as the default. One of the main reasons for this is that if you get stuck and need help, another person is always going to give advice about the default location. If you have chosen to install it in a different directory, this could become more difficult:

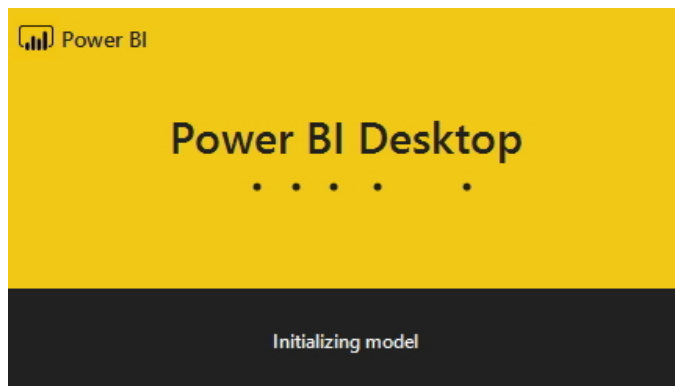
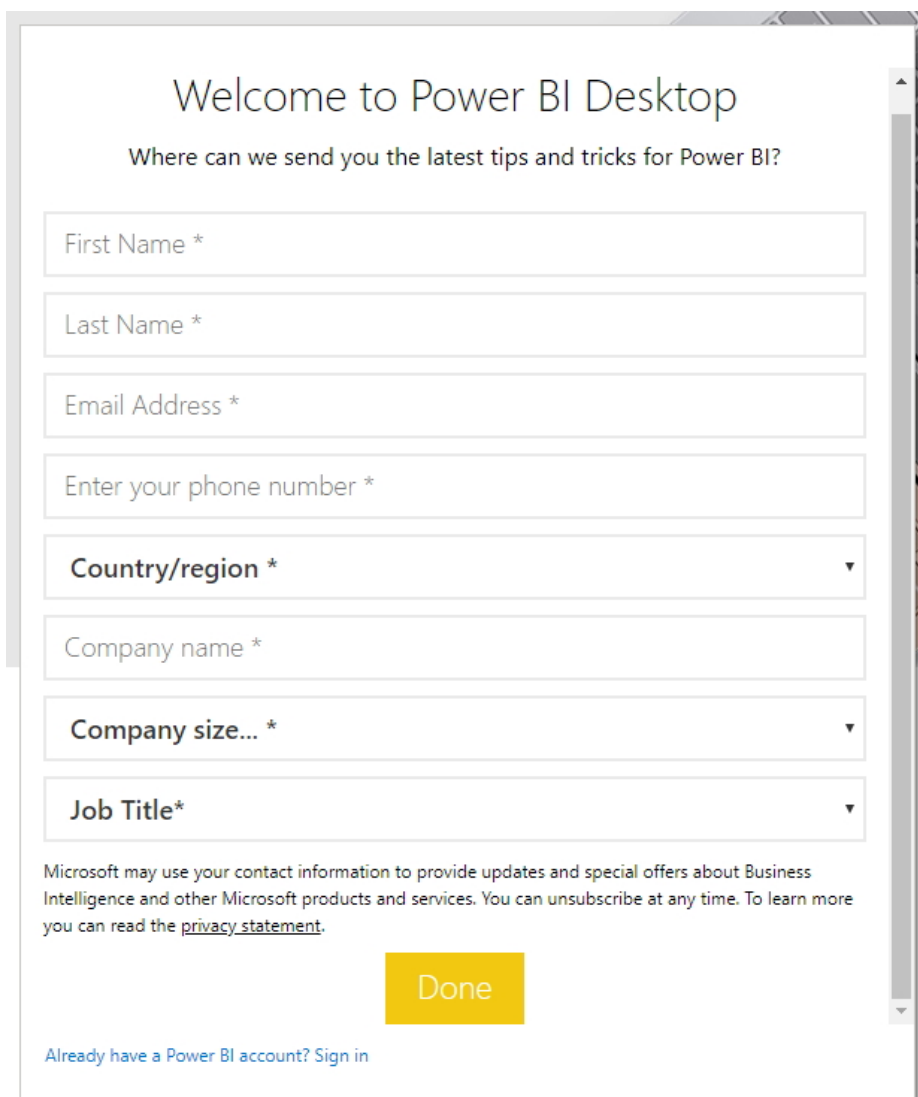


Figure 1.3 – Power BI Desktop application loading screen

Once Power BI has loaded for the first time, you have the option of registering for their newsletter, which provides you with tips and tricks on how to use it:



The image shows a registration window titled "Welcome to Power BI Desktop". Below the title is a question: "Where can we send you the latest tips and tricks for Power BI?". The form contains several input fields, each with an asterisk indicating it is required: "First Name", "Last Name", "Email Address", "Enter your phone number", "Country/region" (a dropdown menu), "Company name", "Company size..." (a dropdown menu), and "Job Title" (a dropdown menu). Below these fields is a paragraph of text: "Microsoft may use your contact information to provide updates and special offers about Business Intelligence and other Microsoft products and services. You can unsubscribe at any time. To learn more you can read the [privacy statement](#)." At the bottom of the form is a large yellow button labeled "Done". Below the button is a link: "Already have a Power BI account? Sign in".

Welcome to Power BI Desktop

Where can we send you the latest tips and tricks for Power BI?

First Name *

Last Name *

Email Address *

Enter your phone number *

Country/region *

Company name *

Company size... *

Job Title*

Microsoft may use your contact information to provide updates and special offers about Business Intelligence and other Microsoft products and services. You can unsubscribe at any time. To learn more you can read the [privacy statement](#).

Done

[Already have a Power BI account? Sign in](#)

Figure 1.4 – Power BI Desktop application registration screen

You also have the option of signing in (use this link to sign up to Power BI for a free trial: <https://powerbi.microsoft.com/en-us/>) or registering your Power BI Desktop account:

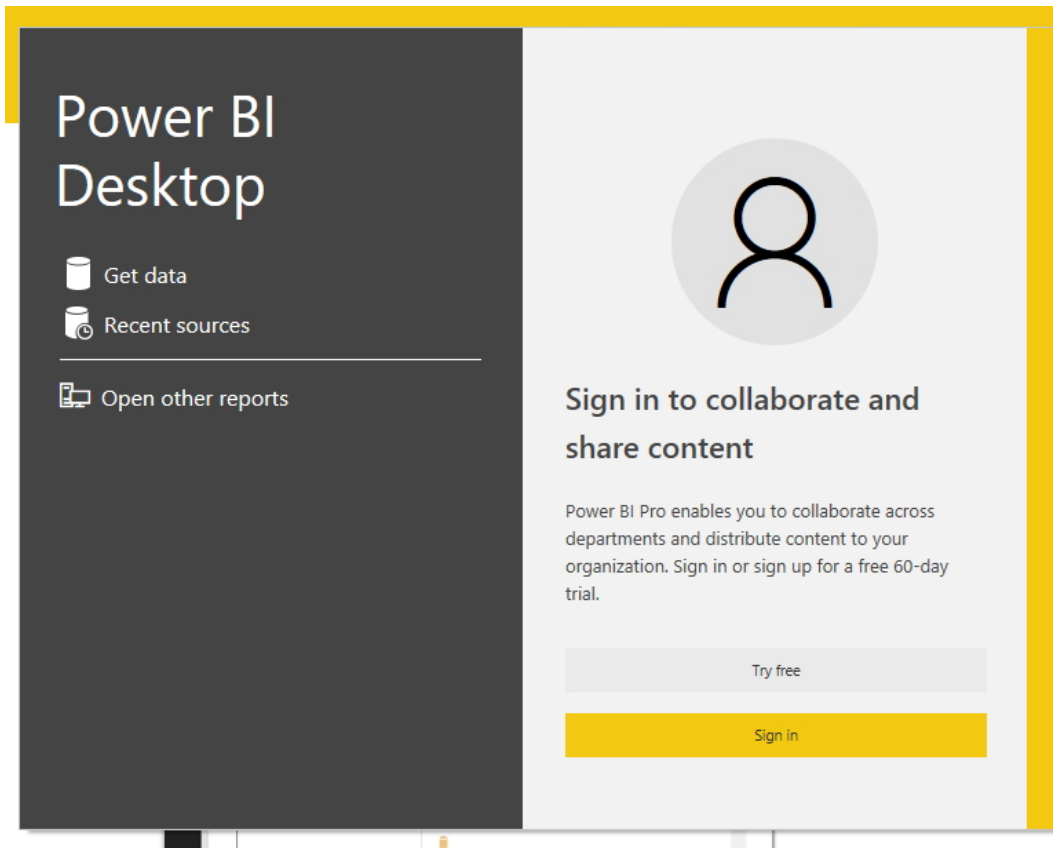


Figure 1.5 – Power BI Desktop application sign-in screen

Once you have completed this and agreed to everything, you will see the splash screen for the first time, which you can choose not to see again by unticking the **Show this screen on startup** option:

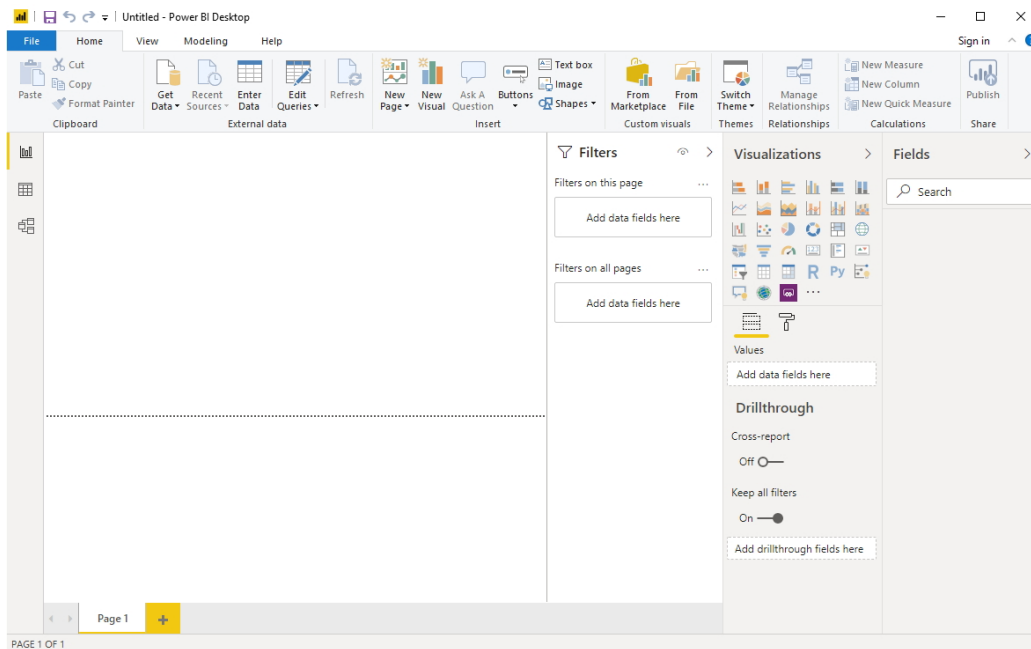


Figure 1.6 – Power BI Desktop application home screen

We personally found that the easier way to install Power BI Desktop is to download it from the Windows Store (available from Windows 8 and above). The only problem with this is that you have to use all the default settings. So, if you want to customize anything, I would recommend the previous way of installing it.

Mac/Apple

Unfortunately, Power BI will only run on a Windows operating system. If you have a Mac, you will not be able to install Power BI, but there are other options, such as Dual Boot Camp, creating a local virtual machine (VM), or using a third-party service. You could also try and run it with an application such as Turbo .net or Parallels.

However, you can download the Power BI App on your iPhone or iPad from the App Store: <https://apps.apple.com/us/app/microsoft-power-bi/id929738808>.

Introduction to Power Query

Power Query is also part of the Power BI family and allows you to extract and load data from a huge range of sources, which includes data from outside a file or database and from web pages. It is considered a data grabber (or data shaper) as you feed data into Power Pivot from Power Query. When working with Excel data, you can create a data model by adding tables using data imported from various sources through Power Query, then use the **Add to Data Model** checkbox on import. Then, you use Power Pivot to create links (relationships) between tables. Both Power Query and Power Pivot allow you to import data sources, but Power Query wins this battle since it provides more choices and has a smaller file size.

You can pull data into existing Power Pivot or Excel tables and then shape the data before you use the data again in Excel. Power Query is available in Excel and Power BI, and the output of any transformed or shaped data can be sent to Excel or Power BI. With Power Query, you can connect to and fetch data, set up conditions, merge and combine, add and change data, and reshape and publish. Power Query uses its own language called M, which we will discuss in detail in the *Learning M* section.

Features of Power Query

Before you can create dashboards in Excel, you will need to involve Power Query. An Excel worksheet consists of 1,048,576 rows, and if you try to import more than this maximum from a data source, Excel will probably crash or end up only importing the limit, which would seriously affect the performance of the workbook. So, to solve this, you use Power Query. This is because it allows you to connect straight to the data source. This means that raw data is not stored directly in the Excel file; it still belongs to the source file.

Power Query stops tasks from having to be repeated manually over and over; for instance, repetitive tasks or macros in workbooks or worksheets. Power Query allows the creation of named queries in which you add a list of steps to perform on a set of data, without requiring coding language experts. This greatly reduces company costs, time, and the need for expert knowledge or outsourced programming experts.

Another cool feature of Power Query is the ability to unpivot data, which allows you to create several Pivot table reports by reversing data into table format. It also allows you to consolidate data from different tables into a single pivot table report. This is especially useful if you have data in separate Excel tables from different divisions of your business.

Another reason why Power Query is so useful is because it allows you to import text files residing in the same folder. Power Query, without effort, quickly imports them into one file in Excel. Alternatively, it allows you to just connect to them, making the creation of Pivot table reports simple and time-efficient!

Power Query is a free add-in for earlier versions of Excel and is included in later versions. Throughout the remaining topics in this chapter, you will learn about which versions of Office support Power Query and where to find Power Query for different versions or releases of Microsoft Excel.

Power Query Office versions and differences

We will first list the compatible versions for Windows and macOS and then discuss the differences between the available versions.

Windows/Android

The following Office versions for Windows are supported by Power Query:

- Microsoft Office 2019, 2016, and 2013: All versions.
- Microsoft Office 2010 Professional Plus with Software Assurance.
- Power Query Premium: All Power Query features available for Professional Plus, Office 365 ProPlus, and Excel 2013 Standalone.
- Power Query Public: Available for all other Office 2013 Desktop SKUs. Includes all Power Query features, except Corporate Power BI Data Catalog, Azure-based data sources, Active Directory, HDFS, SharePoint Lists, Oracle, DB2, MySQL, PostgreSQL, Sybase, Teradata, Exchange, Dynamics CRM, SAP BusinessObjects, and Salesforce.
- **Get & Transform** (Power Query) is currently not supported by Android, iOS, and Online.

Mac

The supported versions for macOS are as follows:

- **Excel 2011 and Excel 2016 for Mac:** Get & Transform (Power Query) is not supported.
- **Excel for Office 365 for Mac:** If you are an Office 365 subscriber and you have signed up for the Windows Insider program, you can refresh existing Power Query queries on your Mac.

Differences between Office 2019 (Office 365) and Office 2016 versions

If you have Office 2019 or Office 2016, you already have Power Query and Power Pivot installed. It has been renamed and is now on the **Data** tab of the ribbon in the **Get & Transform** section:

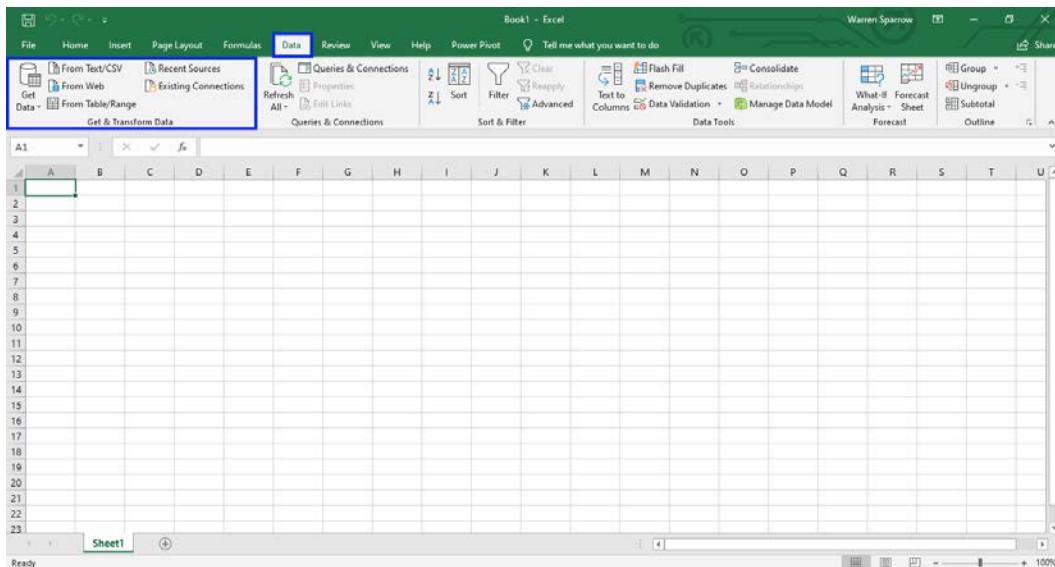


Figure 1.7 – Office 2019 window

Here is the Office 2016 window:

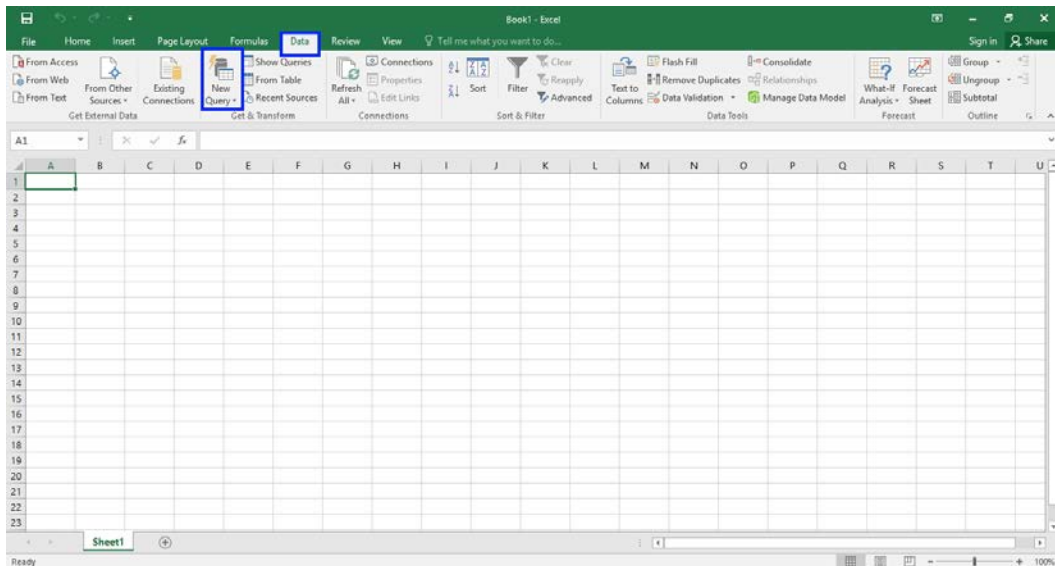


Figure 1.8 – Office 2016 window

As you can see, there are slight differences between the Office 2016 and 2019 versions, but they both work in a similar manner. If you have Office 2016, however, it is possible your ribbon might look different to the one shown previously; it might look more similar to the one shown in the following screenshot:

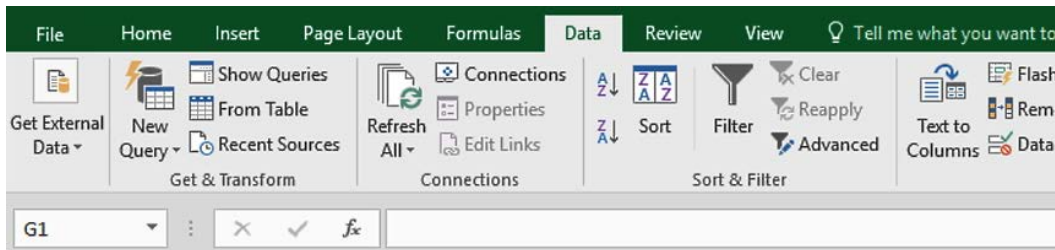


Figure 1.9 – Alternative Office 2016 window

This is because there are different builds of Excel 2016. The different builds include the MSI (Windows Installer) version, the Office 365 subscription, and the ProPlus version. Each build is slightly different and as a result, you might have different icons.

Installing Power Query in Office 2013 and 2010

You will need to download the Power Query add-in from <https://www.microsoft.com/en-us/download/details.aspx?id=39379>.

When you click on the **Download** button on that page, you will be asked if you would like to download the 32-bit or 64-bit system for Office. If you know which version of Office you have, you can download the relevant file.

Note:

The main difference between the two different versions is that the 32-bit version works well for up to about 2 million rows of data, give or take. The 64-bit system can work with more than 20 times that amount without any problem. Something else to remember is that if you have created a Power Pivot workbook for 5 million rows and you then give access to it to another person that has a 32-bit system, the workbook will open but they will not be able to interact with the Pivot table.

Once you have determined which system you have, download the relevant file from Microsoft. You can use these instructions to determine the system you have: <https://support.microsoft.com/en-gb/help/13443/windows-which-version-am-i-running>.

To install the Power Query add-in, follow these steps:

1. First, make sure that Excel is closed before you click on the downloaded file to run the installation. Agree to the terms and conditions and then click on **Next** until you have completed the setup.

Tip:

You must have at least Internet Explorer 9 to install the add-in. If you have a previous version of Internet Explorer, you will have to upgrade your Internet Explorer and then install the Power Query add-in.

2. Once installation is complete, open Excel 2010 or 2013. You will now see the new Power Query ribbon, which looks as follows:

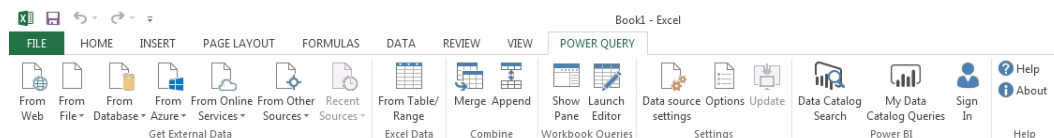


Figure 1.10 – Power Query ribbon

Activating my Power Query/Pivot again

Every now and again, you might notice that the Power Query or Power Pivot tab disappears. This is due to the software COM add-in failing to load. If this happens, you need to reselect the add-in from the **COMM Add-ins** menu. Although there are different ways to get to this menu, this is the process we find the easiest:

Note:
This process is applicable to both Power Query and Power Pivot. For illustration purposes, we are only showing the steps using Power Query.

1. Launch Excel, click on **File**, and then click on **Options**:

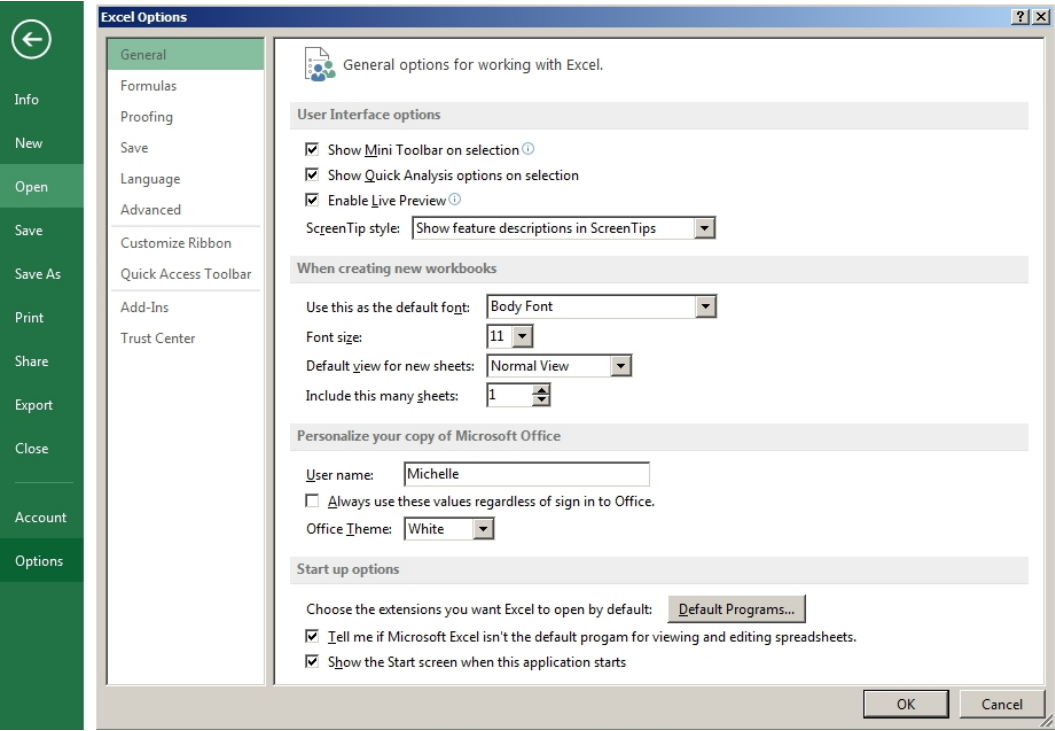


Figure 1.11 – Excel options

2. Click on **Add-ins** on the left-hand side menu and select **COM Add-ins** from the **Manage** drop-down list box:

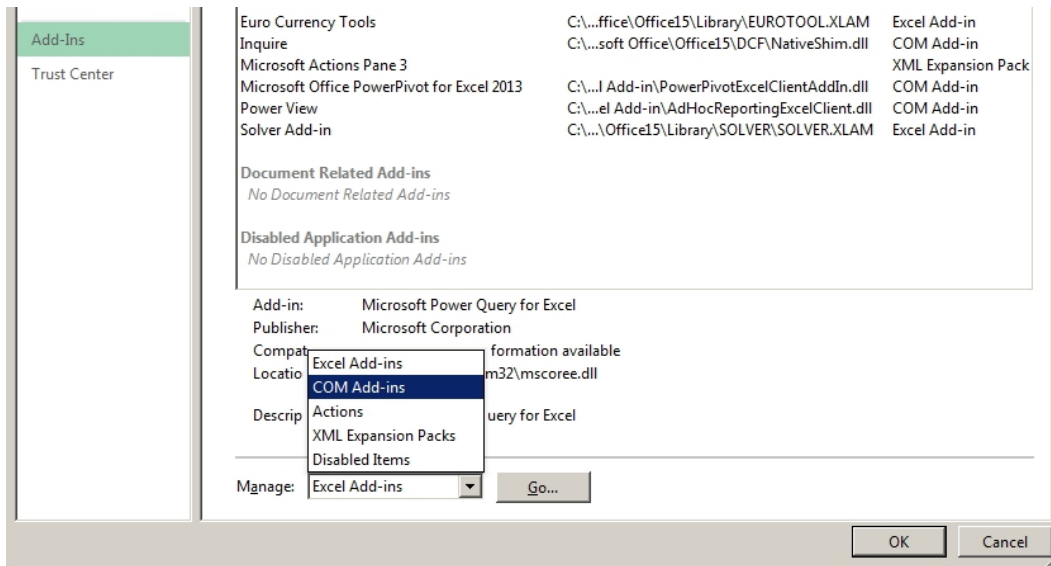


Figure 1.12 – Excel Add-ins

3. This will open the **COM Add-ins** window, where you can select the Power Query checkbox if it is not selected:

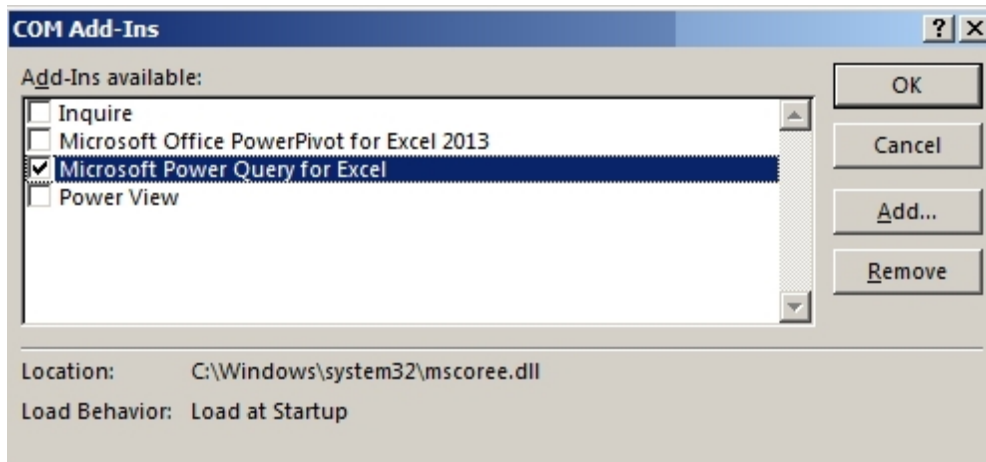


Figure 1.13 – Excel COM Add-Ins options

Launching Power Query, Power Pivot, and Power BI

In this section, you will successfully locate and launch the Power Query Editor from within Microsoft Excel and Power BI. We will be using Microsoft Office 2019 for this example.

Launching Power Query within Excel

Follow these steps to learn how to launch Power Query within Excel:

1. Open Microsoft Excel 2019.
2. Click on the **Data** tab along the ribbon.
3. Click on **Get Data** (this action accesses Power Query):

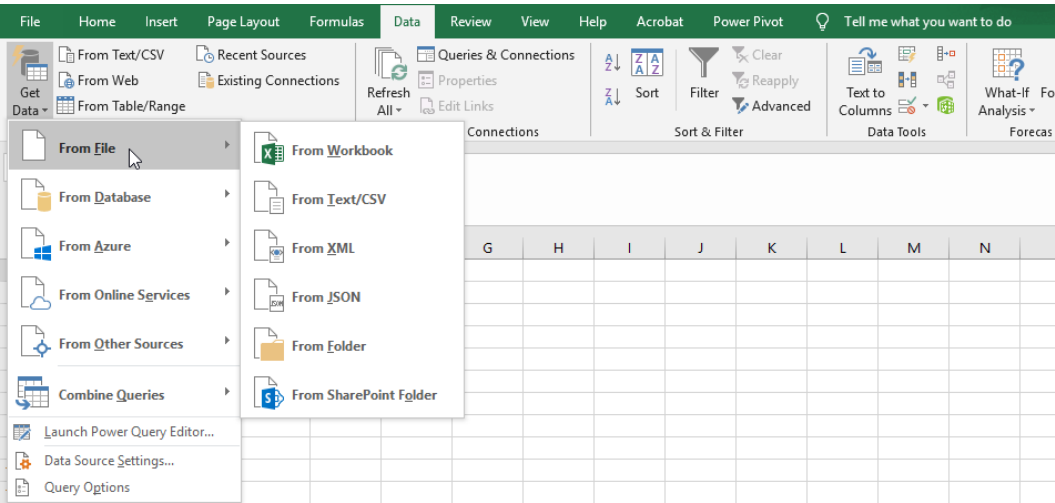


Figure 1.14 – Excel Data tab

4. Choose what data to pull into Power Query from the list of data sources.
5. The **Launch Power Query Editor...** option shown in the drop-down list is used to enable and show data transformations.

Accessing Power Query from Power BI

Follow these steps to learn how to access Power Query from Power BI:

1. Locate Power Query BI desktop on your computer. If you cannot see an icon on your desktop, use the search facility to locate the application and launch it:

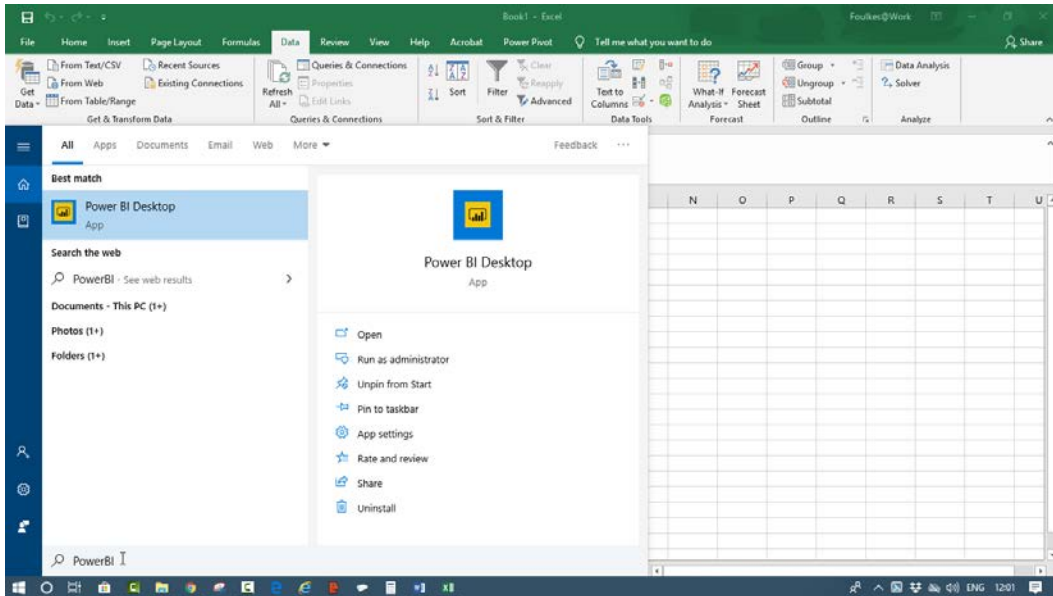


Figure 1.15 – Power BI Desktop

2. Power Query will populate and show the launch screen:

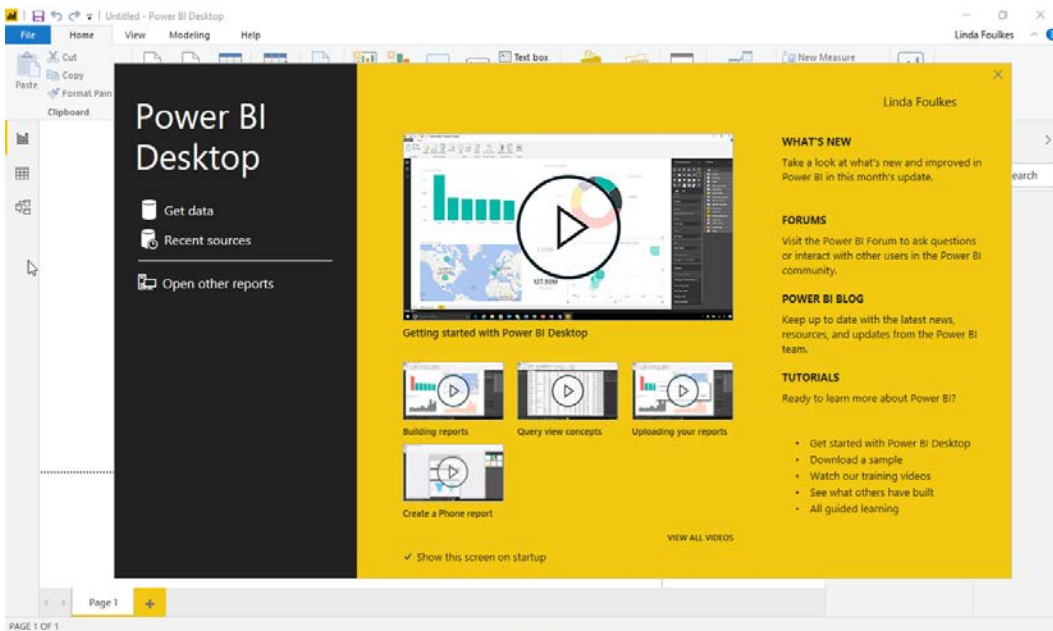


Figure 1.16 – Power BI Desktop startup

3. Use the launch screen to access the **Get Data** option so that you can open Power Query or close the launch screen using the **X** icon at the top right of the launch screen:

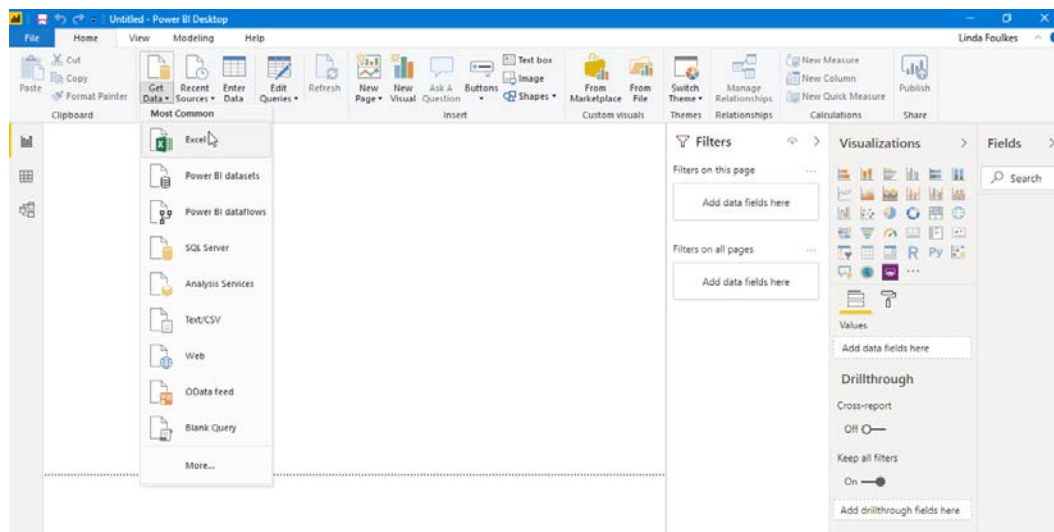


Figure 1.17 – Power BI Desktop

4. Click on **Get Data**, which is located on the **Home** tab (this activates Power Query from within Power BI).
5. Choose a data source to start the Power Query experience.

Summary

In this chapter, you were introduced to three very powerful tools that allow us to create data models, transform and shape data, and create stunning dashboard visualizations. You now have knowledge of what Power BI, Power Query, and Power Pivot can offer and have learned how to set up and install these apps within the Microsoft Office Excel platform, as well as for different desktop versions and releases. You have also learned about their differences and functionality across updates. To end this chapter, you learned how to locate and launch Power Query using the **Get Data** method.

Chapter 2, Power Pivot Basics, Inadequacies, and Data Management, focuses on the shortcomings of Power Pivot and why Power Query is the better tool to use when handling complex data in order to retrieve, extract, and reshape data. By completing this chapter, you will gain an understanding of the importance of data management and data analysis in business. We will look at examples in both Power Pivot and Power Query. You will also learn how to convert worksheet data into tables, ready for query input through Power Query.

2

Power Pivot Basics, Inadequacies, and Data Management

Power Query is now integrated into all the data analysis and business intelligence tools available from Microsoft, including Excel, Analysis Services, and Power BI. This allows users to discover, combine, and refine their data from various sources. In this chapter, you will learn how to install and access these tools.

In this chapter, we're going to cover the following main topics:

- Creating a pivot table
- Creating a Power Pivot
- Creating data tables in Excel
- Creating relationships between tables
- Power Query to the rescue
- Shortcomings of Power Pivot

Technical requirements

You will need an internet connection to download the relevant files from GitHub. Also, the code files for this chapter can be found at:

<https://github.com/PacktPublishing/Learn-Power-Query>

This chapter assumes that you already know how to write basic formulae in Excel.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=5TuXxIuMehY&list=PLcLcvrwLe186O_GJEZs47WazXwZjwTN83&index=3&t=0s.

Businesses have always used data for analysis and then, based on the results, make decisions about the future of their business. Many decisions today are almost instant decisions. People do not want to wait for the information to be analyzed; they want quick responses and turnaround times. For years, businesses used analytics to uncover the basic trends and insights, even before big data was a phrase. Of course, today, businesses need to be able to analyze huge amounts of data, which allows them to be more agile and work faster.

Without going into too much detail about why analytics plays a part in business and why businesses need data analytics, we will cover a few points that convey the importance of data analytics and how Power Pivot, Query, and more play a role in all of this.

One of the reasons data analysis has become so important in business is because it improves efficiency. Data is normally collected and analyzed internally and may cover things from employees to the business' performance. Through the use of data mining (huge datasets that are collected and analyzed) and collecting data from a huge number of people, this often becomes invaluable for marketing and advertising businesses. With regard to Power Query, being able to load and transform the data almost instantly results in a cost reduction, which, in turn, leads to faster and better decision-making, allowing the business to meet deliverables efficiently.

Now, let's start with the first topic of this chapter, which will be the basis for the rest of the topics.

Creating a pivot table

Most people have used a pivot table at some point in their work lives to summarize large amounts of data quickly and easily. As a quick example, I have launched Microsoft Excel and opened the `PowerQuery.xlsx` file, which can be downloaded from [GitHub](#):

Month	Salesman	Region	Product	No. Customers	Net Sales	Profit / Loss
Jan-17	John	North	Dyno1	8	1,592	563
Jan-17	John	North	Dyno2	8	1,088	397
Jan-17	John	West	Dyno3	8	1,680	753
Jan-17	John	West	Dyno1	9	2,133	923
Jan-17	John	West	Dyno2	10	1,610	579
Jan-17	John	Midlands	Dyno3	10	1,240	570
Jan-17	John	Midlands	Dyno1	7	1,316	428
Jan-17	John	Midlands	Dyno2	7	1,799	709
Jan-17	Lucie	North	Dyno3	8	1,624	621
Jan-17	Lucie	North	Dyno1	6	726	236
Jan-17	Lucie	North	Dyno2	9	2,277	966
Jan-17	Lucie	West	Dyno3	6	714	221
Jan-17	Lucie	West	Dyno1	9	2,682	1,023
Jan-17	Lucie	West	Dyno2	6	1,500	634
Jan-17	Lucie	Midlands	Dyno3	7	917	403
Jan-17	Lucie	Midlands	Dyno1	7	1,939	760
Jan-17	Lucie	Midlands	Dyno2	6	984	314
Jan-17	Jordan	North	Dyno3	9	981	372
Jan-17	Jordan	North	Dyno1	10	1,520	476
Jan-17	Jordan	North	Dyno2	6	966	330
Jan-17	Jordan	West	Dyno3	10	2,800	903
Jan-17	Jordan	West	Dyno1	6	1,536	572
Jan-17	Jordan	West	Dyno2	8	816	291
Jan-17	Jordan	Midlands	Dyno3	9	2,547	781
Jan-17	Jordan	Midlands	Dyno1	10	1,810	664
Jan-17	Jordan	Midlands	Dyno2	9	2,223	771
Jan-17	Olive	North	Dyno3	9	1,377	415
Jan-17	Olive	North	Dyno1	7	903	315
Jan-17	Olive	North	Dyno2	9	2,732	828
Jan-17	Olive	West	Dyno3	10	2,070	903

Figure 2.1 – Data for a pivot table

As you can see, we have a table with some sales data that includes salespeople from different regions and the products that they have sold. Let's use this data to make a pivot table:

1. Select/highlight the entire table. After highlighting the table, click on the **Insert** tab on the ribbon, and then click on **Pivot Table**.

Tip

You can also highlight the table after you have clicked on **Pivot Table**. Just click on the arrow button at the end of the **Table/Range** field and click on it again after your selection is complete.

- The range is automatically filled in for you. Here, you can choose to have this on a new worksheet or use the existing worksheet. I have opted for the existing worksheet so that I can see the data at the same time:

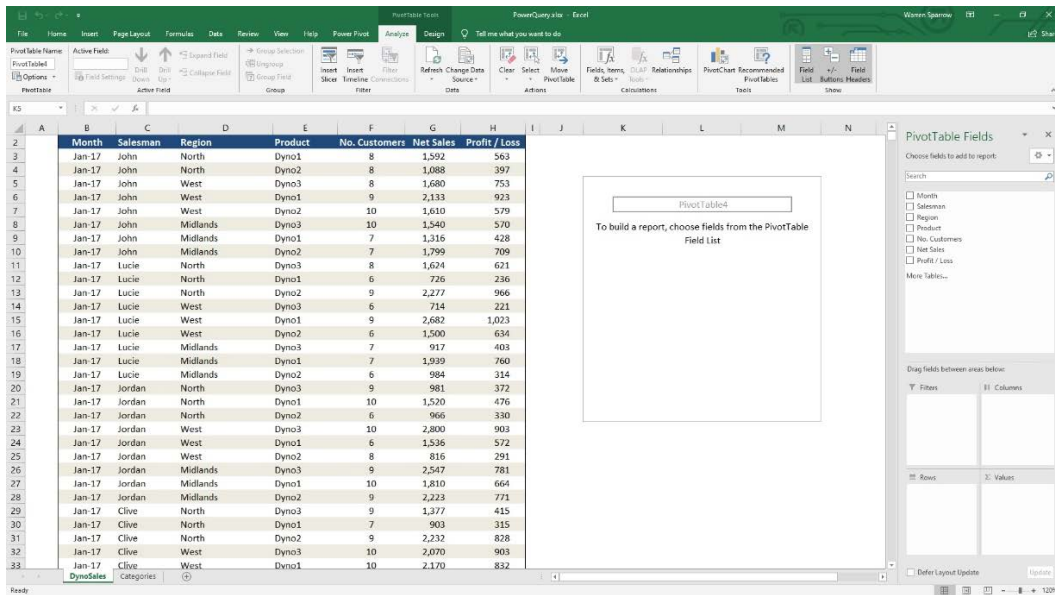


Figure 2.2 – Pivot table fields

Once you have selected where you would like the pivot table to be placed, click on OK.

- You can then select which information you would like to include in your pivot table. I have used a monthly filter, with the region in the columns and the name of the salesmen in the rows. I have also decided to use the sum of the net sales as my value, as shown here:

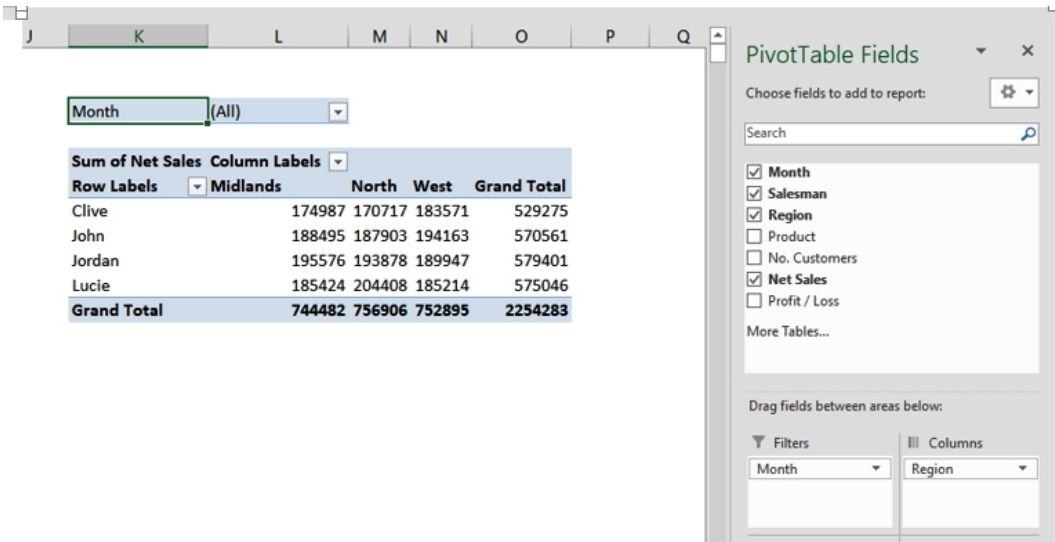


Figure 2.3 – Data selection in a pivot table

4. If we want to know who the top salesman was for a particular time frame or region, we can right-click on **Grand Total** and select **Sort Largest to Smallest**, as follows:

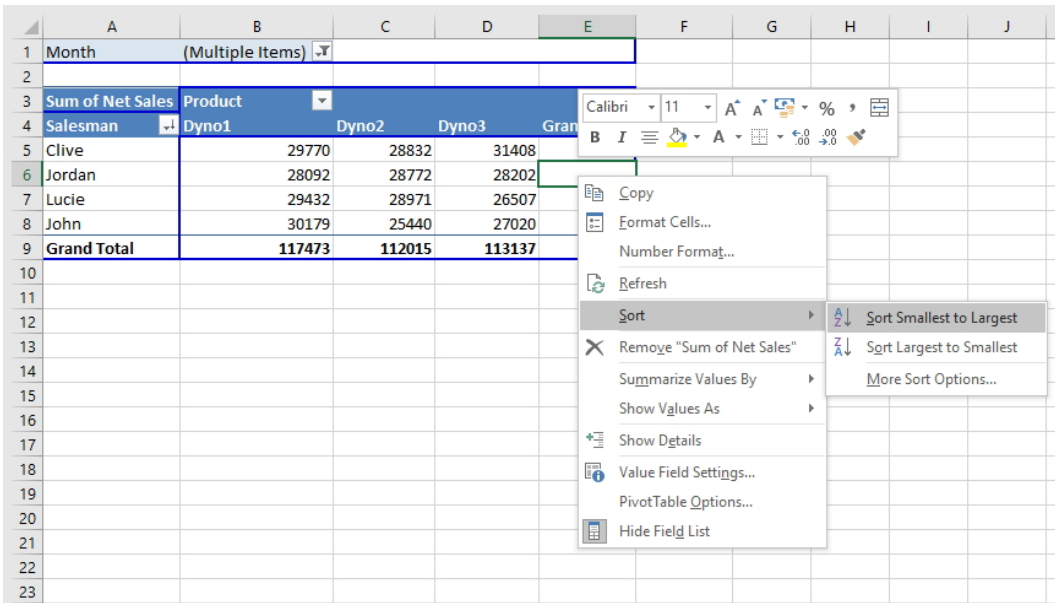


Figure 2.4 – Sorting the data in the pivot table

In the space of 30 seconds, we have created a basic pivot table that has summarized 1,400 rows of data with an interactive view that is easy to modify. One of the great things about this is that you do not need to learn any formulae when you want to change one of the criteria; for example, we won't have to do anything else if we want to know who the top salesman was for **North**, as it automatically refreshes and keeps the same order, from largest to smallest:

	A	B	C
1	Month	(Multiple Items)	
2			
3	Sum of Net Sales	Region	
4	Salesman	North	Grand Total
5	Lucie	30108	30108
6	Clive	29616	29616
7	Jordan	29220	29220
8	John	27744	27744
9	Grand Total	116688	116688

Figure 2.5 – Auto-sorting the data by making a change to the filter in the pivot table

Although pivot tables are easy to create and you can always change what is displayed, I personally like to add *slicers* to them. This does exactly the same thing as selecting the pivot table's fields, but to me, it adds an element that is also great to look at.

One of the reasons I am showing this to you is because this feature is automated in Power Query and I would like you to see how it is otherwise done manually. Of course, the slicers are highly interactive and allow anyone to manipulate the data very quickly:

1. First, you must have the pivot table selected. Then, click on the **Analyze** tab on the ribbon and select **Insert Slicer**, as shown here:

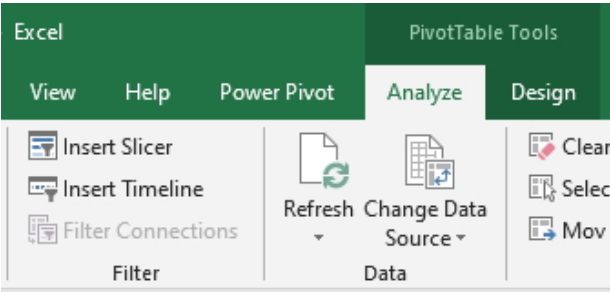


Figure 2.6 – Pivot table Insert Slicer menu

2. You then select the slicers that you would like to include, and then click on **OK**:

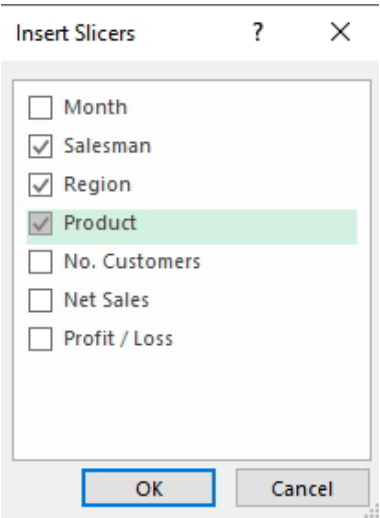


Figure 2.7 – Pivot table slicer options

3. By clicking on the top of the slicer window, you can drag the slicers into any position on your sheet, as shown here:

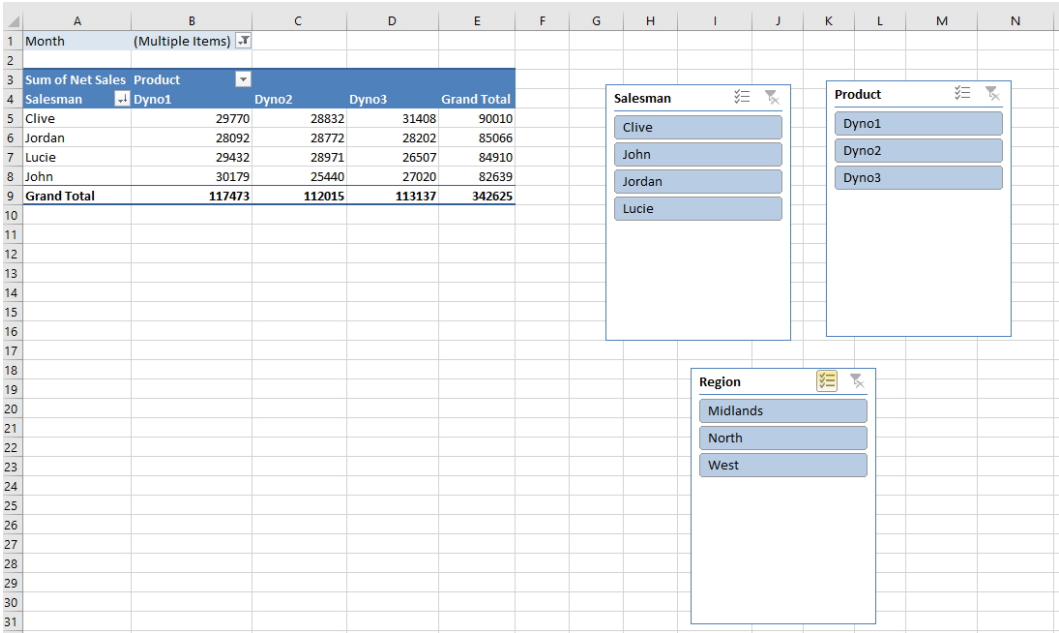


Figure 2.8 – Pivot table slicer position

The slicers allow you to filter data more interactively than just using simple filters and sorting.

Now that we have completed our pivot table, let's say we realize that we have forgotten to add categories that are located on another sheet for each of the products. Let's add these categories so that we can see that data too in our analysis:

Product	Category
Dyno1	Construction toy
Dyno2	Creative toy
Dyno3	Education toy
Dyno4	Electronic toy
Dyno5	Action Figure

Figure 2.9 – Additional Category data

To do so, we would need to insert a column in our original table and create the following VLOOKUP to pull the data:

```
=VLOOKUP(E3, Categories!B$3:C$7,2,FALSE)
```

This results in the following output:

	A	B	C	D	E	F	G	H	I
1									
2									
3		Jan-17	John	North	Dyno1	Construction toy	8	1,592	563
4		Jan-17	John	North	Dyno2	Creative toy	8	1,088	397
5		Jan-17	John	West	Dyno3	Education toy	8	1,680	753
6		Jan-17	John	West	Dyno1	Construction toy	9	2,133	923
7		Jan-17	John	West	Dyno2	Creative toy	10	1,610	579
8		Jan-17	John	Midlands	Dyno3	Education toy	10	1,540	570
9		Jan-17	John	Midlands	Dyno1	Construction toy	7	1,316	428
10		Jan-17	John	Midlands	Dyno2	Creative toy	7	1,799	709
11		Jan-17	Lucie	North	Dyno3	Education toy	8	1,624	621
12		Jan-17	Lucie	North	Dyno1	Construction toy	6	726	236
13		Jan-17	Lucie	North	Dyno2	Creative toy	9	2,277	966

Figure 2.10 – Adding data using VLOOKUP

We will need to extend the table range by selecting **Change Data Source** from the **Analyze** tab and then selecting the new table or range:

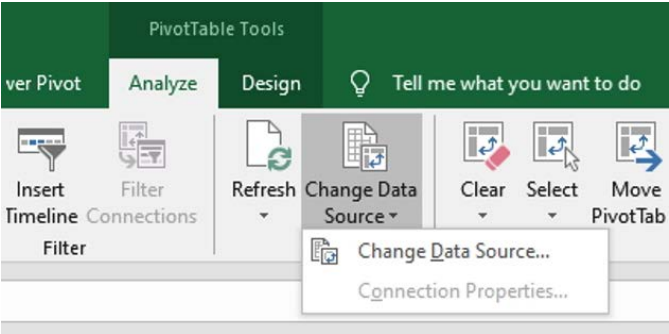


Figure 2.11 – Change Data Source option

Once we refresh the pivot table, it will display the new data. Although this works well with our example, if there are a considerable number of changes being made at once, it might be easier to create a new pivot table:

Month (All)

Sum of Net Sales	Column Labels					
Row Labels	Dyno1	Dyno2	Dyno3	Dyno4	Dyno5	Grand Total
Clive	162953	161134	170195	17729	17264	529275
Action Figure					17264	17264
Construction toy	162953					162953
Creative toy		161134				161134
Education toy			170195			170195
Electronic toy				17729		17729
John	177588	172834	177797	24065	18277	570561
Action Figure					18277	18277
Construction toy	177588					177588
Creative toy		172834				172834
Education toy			177797			177797
Electronic toy				24065		24065
Jordan	181414	163857	183305	27896	22929	579401
Action Figure					22929	22929
Construction toy	181414					181414
Creative toy		163857				163857
Education toy			183305			183305
Electronic toy				27896		27896
Lucie	174044	173706	178300	31953	17043	575046
Action Figure					17043	17043
Construction toy	174044					174044
Creative toy		173706				173706
Education toy			178300			178300
Electronic toy				31953		31953
Grand Total	695999	671531	709597	101643	75513	2254283

Product

Dyno1

Dyno2

Dyno3

Dyno4

Dyno5

Region

Midlands

North

West

Salesman

Clive

John

Jordan

Lucie

Figure 2.12 – Pivot table VLOOKUP

Once completed, this looks very impressive. However, this can be time-consuming, especially as you have to do the same thing every month with new data with different file extensions and columns.

Note that there are additional problems with pivot tables. The biggest is that Excel only handles just over 1 million rows that you can use, and although you might be thinking, *who is ever going to use more than that?*, when it comes to large datasets, it is common practice to have more than 5 million rows of data that you would like to analyze.

Creating a Power Pivot

Let's create a basic Power Pivot to compare how this is different from a pivot table. While doing this, we will look at some of the advantages and limitations of Power Pivot.

Note

Before we get started, I would like to say that when we look at the whole range of Power products (Power View, Power BI Desktop, Power Map, Power BI, and Power Query), Power Pivot is the DAX engine and is, in essence, the *brain* that drives all of the reporting, visualization, and analysis.

There are two parts to creating a Power Pivot - first, we will have to create a table in Excel and then add it to the data model.

Creating a table in Excel

I have opened an Excel document (shown in the following screenshot) that contains the same information as we used previously. However, instead of using 1,400 rows of data, I am going to extend that to 1,048,576 rows, which just happens to be the last row on an Excel spreadsheet:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Date	SalesRep	Product	Units	Discount		Product	Category	Price		SalesRep	Region
2	02/04/2014	Nancy	Game1	1	0.007		Dyno1	Construct	14.76		Clive	South
3	24/07/2011	Peter	Dyno1	12	0.012		Dyno2	Creative	29.76		John	West
4	02/03/2011	Harry	Board2	7	0.002		Dyno3	Education	45.2		Jordan	East
5	24/04/2010	Clive	Dyno4	4	0.007		Dyno4	Electronic	12.5		Lucie	North
6	23/05/2014	John	Game8	2	0.006		Dyno5	Action	11.87		Bevan	South
7	17/12/2015	Charlie	Dyno5	3	0.017		Board1	Education	8.55		Claire	South
8	27/09/2011	Peter	Board3	2	0.006		Board2	Electronic	14.34		Olivia	North
9	22/10/2014	Nancy	Game3	5	0.006		Board3	Education	11.23		James	West
10	29/03/2016	Mary	Game4	10	0.010		Board4	Action	9.67		Peter	West
11	27/11/2014	Thelma	Game7	2	0.020		Board5	Creative	8.34		Lars	East
12	23/12/2014	Jordan	Dyno5	9	0.016		Game1	Construct	4.56		Nancy	West
13	09/06/2010	Lucie	Game4	3	0.001		Game2	Creative	8.35		Alfie	North
14	07/09/2010	Bevan	Dyno3	3	0.009		Game3	Education	9.23		Thelma	East
15	06/10/2010	Susan	Game4	12	0.011		Game4	Education	12.51		Sean	North
16	20/01/2015	Lucie	Board5	9	0.002		Game5	Electronic	5.87		Mary	North
17	06/02/2011	Dennis	Dyno2	11	0.015		Game6	Construct	11.34		Harry	East
18	18/05/2015	Xavier	Game7	4	0.001		Game7	Action	5.88		Victoria	West
19	04/03/2011	Xavier	Dyno4	10	0.013		Game8	Creative	5.99		Robert	North
20	21/08/2013	Robert	Game7	5	0.002		Game9	Creative	4.65		Dennis	East
21	26/02/2011	Florence	Board3	13	0.020		Game10	Education	12.67		Susan	South
22	04/10/2012	Sean	Game8	8	0.019						Dale	North
23	31/08/2014	Susan	Dyno3	3	0.018						Helen	East
24	24/04/2011	Victoria	Dyno1	2	0.008						Florence	West
25	24/08/2011	Edgar	Game9	12	0.001						Charlie	North
26	29/03/2013	Clive	Game6	11	0.011						Michael	East
27	07/09/2010	Susan	Game2	2	0.002						Xavier	South
28	24/11/2013	Nancy	Game8	14	0.013						Edgar	North

Figure 2.13 – Data for Power Pivot – start of the sheet

The following screenshot shows the last row in this table, which is 1,048,576:

1048567	23/06/2010	Edgar	Dyno4	2	0.006	
1048568	27/02/2015	Peter	Dyno2	12	0.001	
1048569	04/01/2013	Robert	Game7	5	0.006	
1048570	19/01/2011	James	Dyno4	14	0.018	
1048571	05/10/2013	Dennis	Board4	12	0.009	
1048572	27/10/2010	Harry	Game10	6	0.004	
1048573	23/11/2014	Clive	Board2	10	0.015	
1048574	30/03/2013	Robert	Game8	7	0.012	
1048575	30/08/2015	Peter	Game10	8	0.013	
1048576	01/03/2015	Susan	Game10	3	0.001	

Figure 2.14 – Data for Power Pivot – end of the sheet

Here are the steps to convert the data:

1. First, we need to convert the preceding data into a table.

Note

Data and *table* are database terminology, both of which refer to a container for information. Each table contains information about a single thing, for example, a product or the sales representative selling the product, which is organized into rows and columns. Basically, a table is an object that allows us to create relationships to other objects in other tables. In a *database*, we view the contents of a table in a form called a datasheet. Although it looks like rows and columns, in a database, they are called records and fields.

There are two different ways in which you can do this. The first is to click anywhere inside the data range, click on the **Insert** tab on the ribbon, and then click on **Table**. The second option is to use the *Ctrl + T* keyboard shortcut:

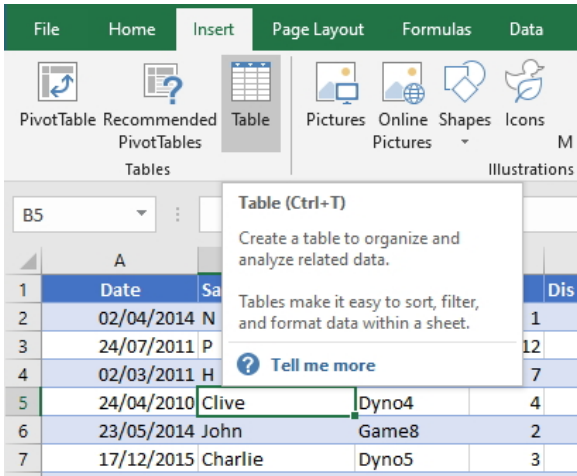


Figure 2.15 – Creating a table

2. You can either click on the **OK** button or press the *Enter* key to confirm this:

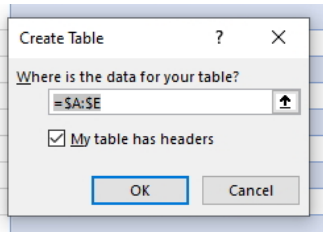


Figure 2.16 – Creating a table

Note

It is always a good idea to name your tables, especially when you have a few of them; this will make them easier for you to identify. Note that there are a few rules to adhere to when naming a table. The table name has to start with a letter or an underscore (_). You also cannot have any spaces in the name, and you obviously cannot give the table a name that already exists.

3. I am going to keep to database modeling and start my table name with a fact table, before my fact or transaction table (please read the beginning of *Chapter 4, Connecting to Various Data Sources Using Get & Transform*, for a full explanation of fact and dimension tables):

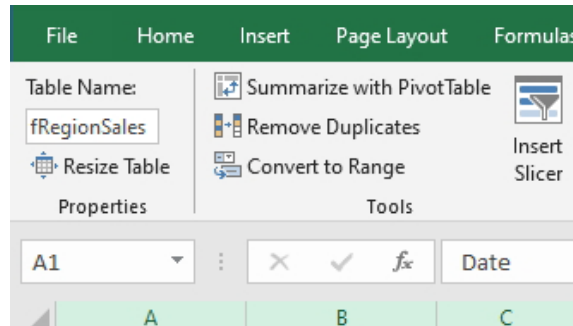


Figure 2.17 – Creating a table

4. Select the next range, which is the SalesRep data, and using either of the methods mentioned previously, convert this into a table. If your table has headings, do not forget to tick the **My table has Headings** box. I am going to rename this table with a prefix of D for dimension and call it SalesRep.
5. Repeat this procedure with the last lot of product data to convert it into a table. In keeping with the current format, I'll call it dProduct.

We have now converted all the pertinent data into tables. We can now add the tables to the data model.

Adding tables to the data model

Data modeling is a framework that shows how the data is connected within a database and how it is stored, processed, and updated inside the system. Although the relational model is the most commonly used, there are others available, such as the hierarchical and network models. This step should not be overlooked or taken lightly as this component provides the structure that will support the analytical needs of the decision-makers. Let's get started:

1. Click on the **Power Pivot** tab on the ribbon and select **Add to Data Model**:

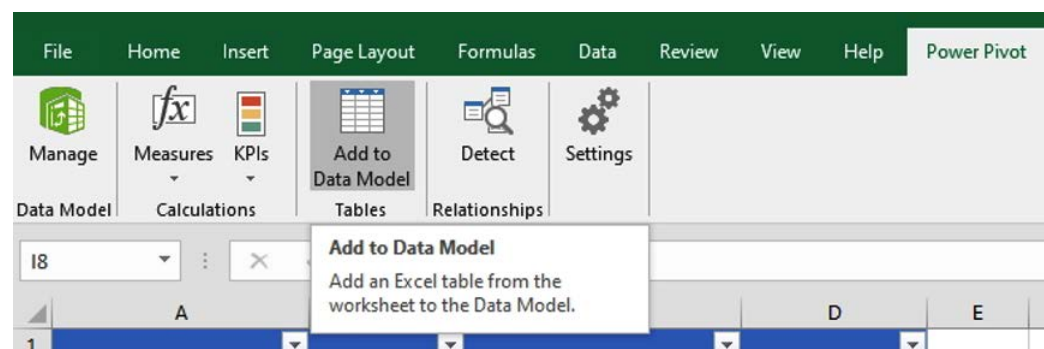


Figure 2.18 – Add to Data Model option

This may take a little while if the dataset is large. It is important to realize that the data model is linked to the data in the table we created. This means that if we change any information in our Excel table, it will automatically update our data model.

2. After clicking on **Add to Data Model**, the **Manage Data Model** window will open. Here, there are three new tabs at the top, namely **Home**, **Design**, and **Advanced**:

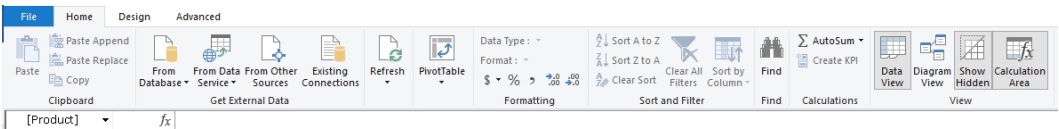


Figure 2.19 – Manage Data Model window

3. Go back to Excel, either by using *Ctrl + Tab* or clicking on the **Switch to Workbook** icon located on the top-left of the window. Add the two other tables to the data model in the same way as we added the first:

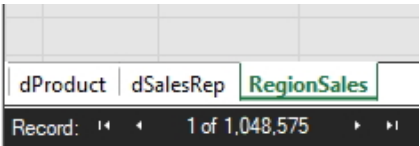


Figure 2.20 – All three tables in the data model window

- 4. When we look at **RegionSales**, you might notice that the date does not fit into the column. So, we will resize this in the same way we would resize a column in Excel.
- 5. Once the column has been resized, we will change the format of the column as we do not want the time and date format in the same column. Click on **Format** from the **Home** tab, and then select the fourth format type (DD/MM/YYYY) from the top of the drop-down list, which will then display the date format without the time:

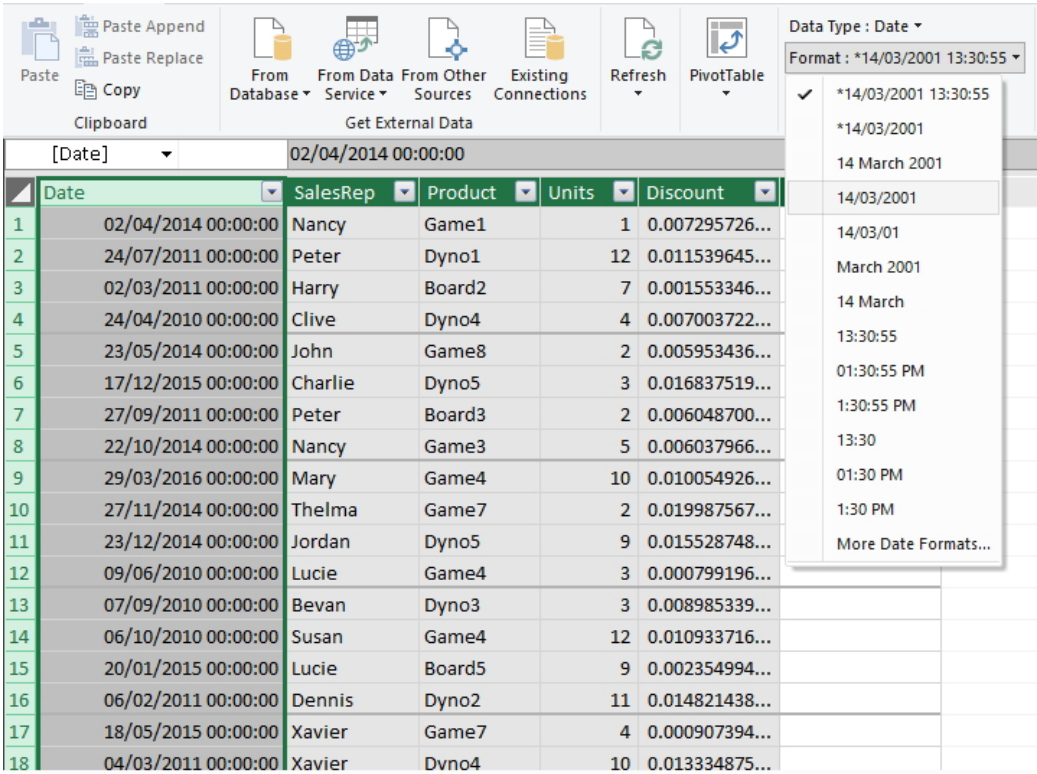
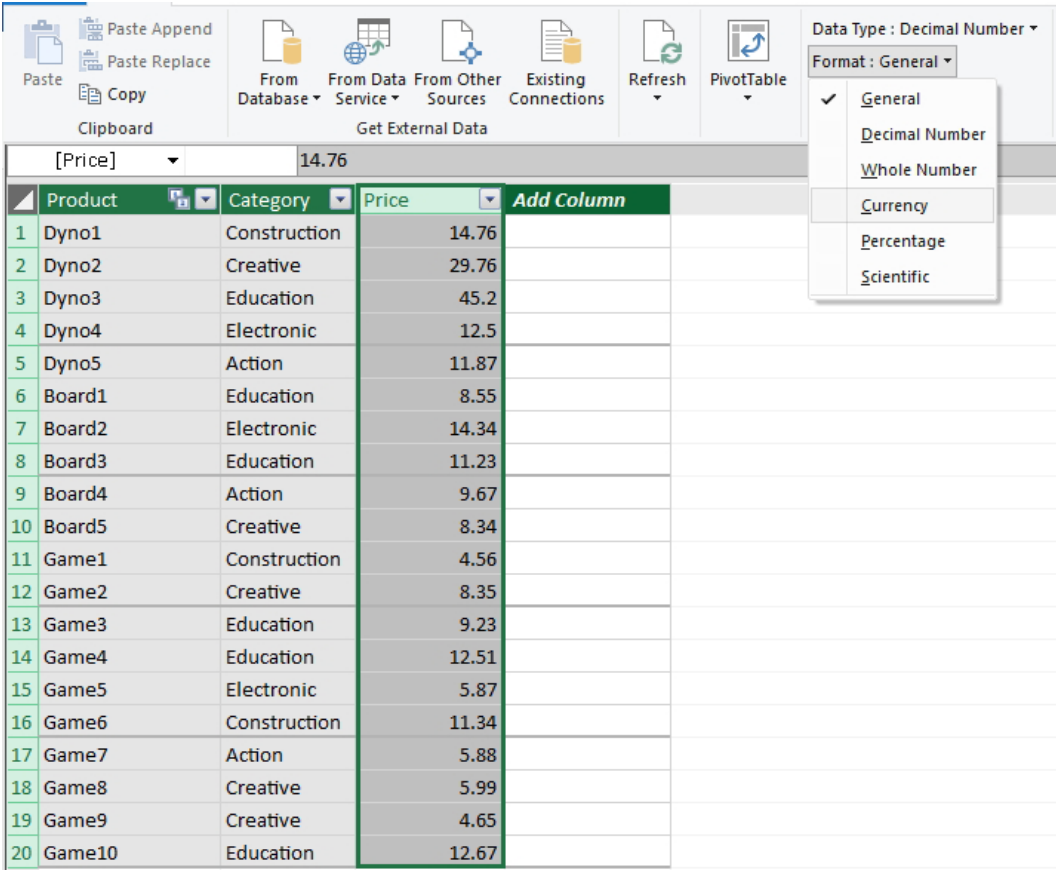


Figure 2.21 – Changing the date format in the data model

6. Now, let's change the format of **Price** by clicking on the **dProduct** tab, selecting the **Price** column, and then changing it to **Currency**:



The screenshot shows the Microsoft Excel ribbon with the 'dProduct' tab selected. The 'Price' column is highlighted, and the 'Format' dropdown menu is open, showing options like General, Decimal Number, Whole Number, Currency, Percentage, and Scientific. The 'Currency' option is highlighted. The data table below shows the following values:

	Product	Category	Price	Add Column
1	Dyno1	Construction	14.76	
2	Dyno2	Creative	29.76	
3	Dyno3	Education	45.2	
4	Dyno4	Electronic	12.5	
5	Dyno5	Action	11.87	
6	Board1	Education	8.55	
7	Board2	Electronic	14.34	
8	Board3	Education	11.23	
9	Board4	Action	9.67	
10	Board5	Creative	8.34	
11	Game1	Construction	4.56	
12	Game2	Creative	8.35	
13	Game3	Education	9.23	
14	Game4	Education	12.51	
15	Game5	Electronic	5.87	
16	Game6	Construction	11.34	
17	Game7	Action	5.88	
18	Game8	Creative	5.99	
19	Game9	Creative	4.65	
20	Game10	Education	12.67	

Figure 2.22 – Changing the format of the Price column to Currency

7. We can also sort any of the columns in the table by selecting the **Sort Oldest to Newest** icon, which will make the data easier to understand:

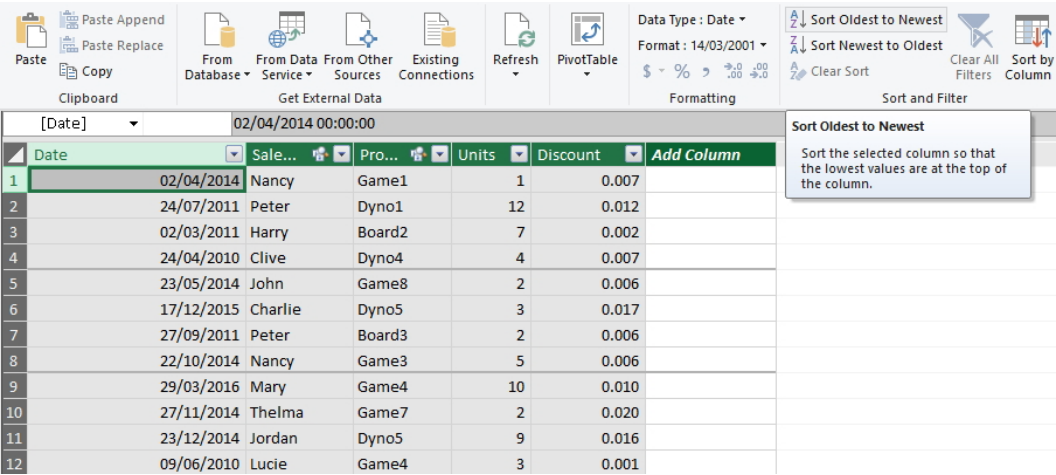


Figure 2.23 – Sorting the table in ascending or descending order

We can also change the **Discount** field to a percentage in the **fRegionSales** tab, but I have left mine as is.

Creating relationships between tables

Next, we will need to create our relationships. There are a number of reasons why we use relationships between tables. Two reasons for doing this are to establish a connection between tables that are logically related to each other and to minimize redundant data. Let's get started:

- 1. On the **Home** tab, click on the **Diagram View** icon:

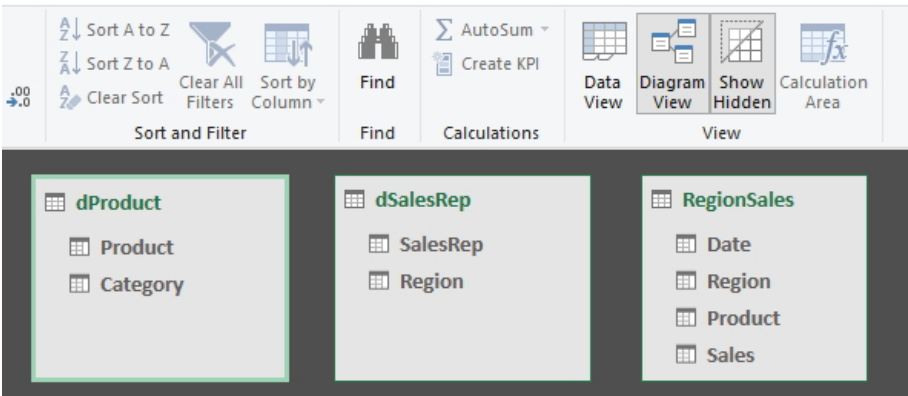


Figure 2.24 – Diagram View in the data model

2. To create a relationship, you will need to drag and drop **Region** in the **dSalesRep** table to **Region** in the **RegionSales** table. Doing so will link the two tables through a matching field.

There are different relationships that you can create in Power Pivot. The one we have created is a one-to-many relationship. In the **dSalesRep** table, the **Region** column has a unique list in the lookup table for each **SalesRep** in each region. The **RegionSales** table has many regions in this table since there are many sales. You will see a link from the one table to the other table.

3. From the **RegionSales** table, create a relationship with **Product** with the **dProduct** table in the same way:

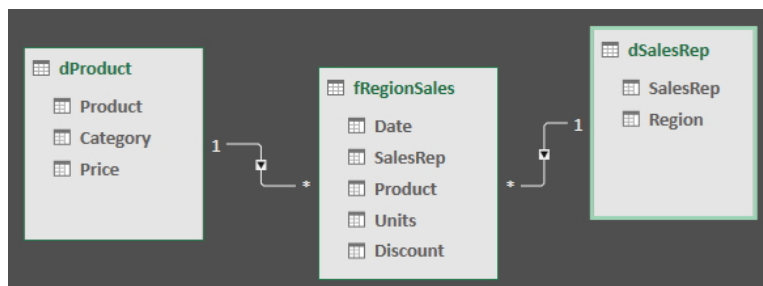


Figure 2.25 – Creating relationships between tables

It is sometimes difficult to see which columns are linked together when you are looking at the **Diagram View** screen. Looking at the preceding table, it looks like a category in the **dProduct** table is linked to the **Region** column in the **RegionSales** table. If you click on the actual link, this becomes much clearer.

If you would like to change or view the relationship, you can right-click on the relationship and then click **Edit Relationship....** This will bring up a window that displays the tables and columns and how they relate to each other:

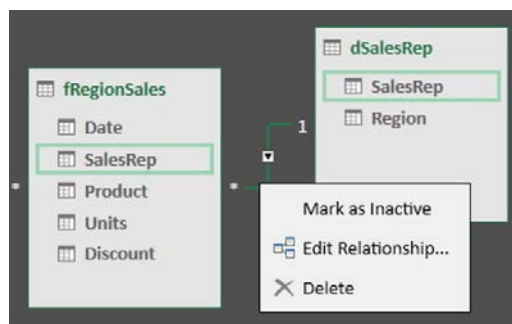


Figure 2.26 – Edit Relationship window

If you want to change the relationship, it is as easy as clicking on another column in the relevant table:

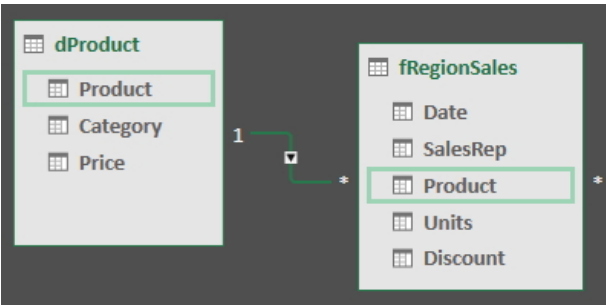


Figure 2.27 – Relationships between tables

Before going back to the **Data View** screen, we will need to save the relationships that you have created; otherwise, it will not link the tables together. There are a number of different ways in which you can save these relationships. You can either click on the **Save** icon; click on the **File** tab, then **Save**; or use the *Ctrl + S* keyboard shortcut.

Power Query to the rescue

Now that we have created the relationships between our different tables, we will link these tables together. We will need to create a new field called **NetIncome** in our **fRegionSales** table. To keep the integrity of the data model, we cannot use Power Pivot to edit the existing data. However, we can create a new calculated column based on our existing data in the table.

Creating a calculated column

Click to select the **fRegionSales** tab. We are now going to add a column and create a calculated column. Double-click on the next available column on the right that states **Add Column** and add **Net Income**. Then, press *Enter*:

	Date	SalesRep	Product	Units	Discount	Net Income	Add Column
1	15/06/2016	John	Game7	10	0.003		
2	15/06/2016	Sean	Game8	3	0.014		
3	15/06/2016	Michael	Game5	10	0.020		
4	15/06/2016	Michael	Game2	12	0.016		
5	15/06/2016	John	Game8	9	0.012		
6	15/06/2016	Mary	Board1	2	0.007		

Figure 2.28 – Creating a new column name

Looking at the preceding table, we may want to multiply the units and the discount, along with the unit price, which is in a different table. Normally, we would do a VLOOKUP in Excel to fetch this data. However, VLOOKUP does not exist in Power Pivot. Fortunately, there is another function that we can use called related. Let's get started:

1. Click in the first cell in the **Net Income** column and type an equals (=) sign, in the same way that you would for create a formula in Excel, and then type **related**. When you open the bracket, Power Pivot automatically populates the different linked tables (this is why we created the relationships between the tables):

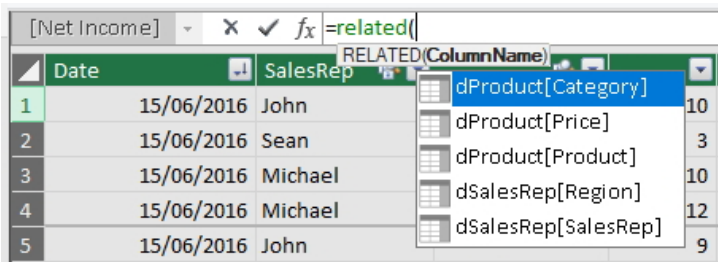


Figure 2.29 – Creating a related formula

2. At this point, there are two ways in which we can complete the formula. The easiest way is to click on **dProduct[Price]**, which was offered to us automatically. The second way is to click on the **dProduct** tab, and then select the **Price** column. I prefer this method for no other reason than to see that I have selected the correct column:

The screenshot shows the completed formula `=related(dProduct[Price])` in the formula bar. The table now has three columns: **Product**, **Category**, and **Price**. The **Price** column contains values calculated using the RELATED function.

	Product	Category	Price
1	Dyno1	Construction	£14.76
2	Dyno2	Creative	£29.76
3	Dyno3	Education	£45.20
4	Dyno4	Electronic	£12.50
5	Dyno5	Action	£11.87
6	Board1	Education	£8.55
7	Board2	Electronic	£14.34

Figure 2.30 – Creating the related formula with manual selection

- Now, close the brackets and press *Enter*. Notice that all the unit prices are now displayed in the column:

[Net Income]		fx		=related(dProduct[Price])			
	Date	SalesRep	Product	Units	Discount	Net Income	Ad
1	15/06/2016	John	Game7	10	0.003	5.88	
2	15/06/2016	Sean	Game8	3	0.014	5.99	
3	15/06/2016	Michael	Game5	10	0.020	5.87	
4	15/06/2016	Michael	Game2	12	0.016	8.35	
5	15/06/2016	John	Game8	9	0.012	5.99	

Figure 2.31 – Net Income column is updated with the product's price

Of course, if you knew all of this, you could have simply typed in `=related(dProduct[Price])`.

Note

There are a number of things I would like to mention before continuing. In Power Pivot, this is the convention when we write formulas. We will always have our table name and our field in square brackets.

In Excel, the cells and tables have a structured reference, for example F6, while Power Pivot is a bit more like an **Access Database**, which uses table names and field names.

We will discuss this in more depth later in this book, but you will need to understand row context. When we created a relationship between the different tables, PowerPivot knew that there was a relationship in the product column. In our first row, we have **Game7**. **PowerPivot** then looks in the **dProduct** table and finds the price for **Game7** and inserts it in that row. It then goes down to the second row and do exactly the same thing. Row context is something that we will use in calculated columns.

Lastly, when we look at the formula for each of the different rows in our table, you will notice that every row has the same formula. In Excel, you would normally use a relative cell reference or something else in your formula, but because of the row context, every formula is the same.

Now that we have obtained the product's price from the **dProduct** table, we can complete the rest of the formula to get the net income.

- Click in the formula bar to edit the formula and type $\ast (1 -$. Then, click on **Discount** and close the bracket. Now, we only need to multiply the number of units: type in $\ast ($, select **Units**, and close your brackets.

The completed formula will look as follows:

```
=related(dProduct[Price])*(1-fRegionSales[Discount])*(fRegionSales[Units])
```

This results in the following output:

[Net Income] X ✓ fx =related(dProduct[Price])*(1-fRegionSales[Discount])*(fRegionSales[Units])							
	Date	SalesRep	Product	Units	Discount	Net Income	Add Colu
1	15/06/2016	John	Game7	10	0.003	58.615706463...	
2	15/06/2016	Sean	Game8	3	0.014	17.723319193...	
3	15/06/2016	Michael	Game5	10	0.020	57.530667481...	
4	15/06/2016	Michael	Game2	12	0.016	98.590910377...	
5	15/06/2016	John	Game8	9	0.012	53.277641840...	
6	15/06/2016	Mary	Board1	2	0.007	16.983677940...	
7	15/06/2016	Charlie	Board2	3	0.010	42.575205935...	
8	15/06/2016	Xavier	Dyno5	13	0.015	152.00610909...	
9	15/06/2016	Helen	Dyno2	1	0.017	29.268880248...	
10	15/06/2016	Sean	Board1	11	0.001	93.991070872...	
11	15/06/2016	Peter	Dyno3	1	0.007	44.899078972...	

Figure 2.32 – The completed Product Price formula

- When looking at the final number, we may wish to convert this into currency. Click on the **Currency** icon and select the appropriate currency.

Now that we have finished creating a calculated column, I would like to look at creating a calculated field, which will allow us to use a formula to add up the total region sales:

Note

A calculated field is sometimes known as measures or explicit formulas.

Creating a calculated field

There are two ways to create a calculated field. First, in Power Pivot, you can click in the measured grid below your table. Let's say we want to work out the total net income. Although this is possible to do in a pivot table, it would be more beneficial to create the explicit measure in Power Pivot. One of the most important reasons for doing this is that once you create the formula, you can use it over again and you can also use it in other DAX formulae. Let's take a look:

1. We start off by creating the name of the calculated field. I am going to type in `Total Net Income:`. Although you start typing in the measured grid, it will get typed into the formula bar. We will now add `=sum(fRegionSales[Net Income])`.

Note

If you have used and coded in **Access**, you will know that you start by adding your calculated field name followed by a colon before starting your formula.

2. After pressing *Enter* and expanding the column, you will notice that the **Total** field is not formatted correctly:

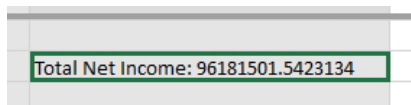


Figure 2.34 – Incorrectly formatted calculated field

3. Simply click on the **Currency** symbol from the **Home** tab ribbon to correct this.
- The second way to create a calculated field is to go back to Excel and click on **Measures** in the **PowerPivot** ribbon:

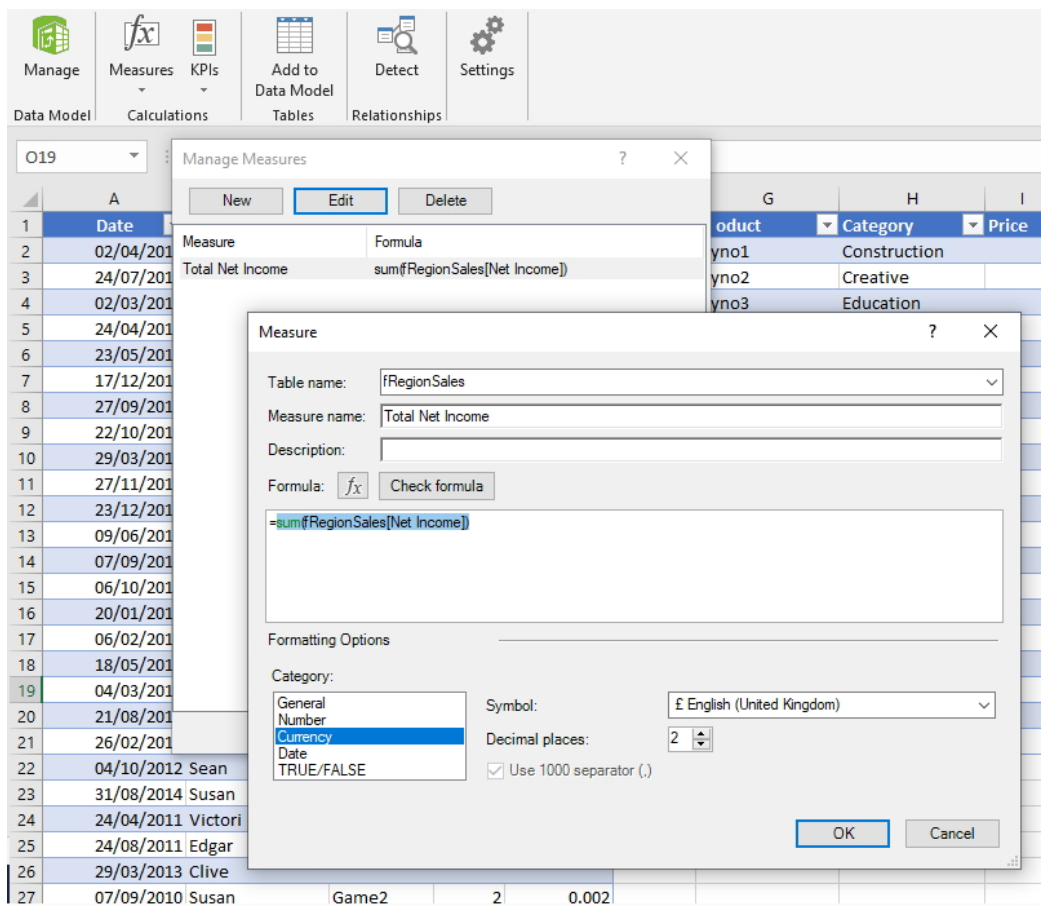


Figure 2.35 – Second way to create a calculated field

From here, you can create a new calculated field, but you can also edit existing ones like the one we have created. In the preceding screenshot, you will see that it has saved our number as **Currency**. Personally, I think the first way is easier as you can click and see the relevant information, as opposed to typing everything in a straight formula.

We have looked at a number of different things separately, such as creating relationships, creating calculated columns and fields, and writing some M formulae. In the next section, we will look at how all of these features can be used together to create a Power PivotTable and some of the shortcomings that Power Pivot has.

Creating a Power Pivot table

All the sections in this chapter have been necessary for us to get the data into a format that we could use to create our Power Pivot table. From the initial Excel data, which contains 1,048,576 rows that we imported as a table and added to our **Data Model**, we changed the data types and created the necessary relationships. We then created calculated columns that worked out the product price and our net income and calculated fields that worked out the region sales. We were exposed to the M language and its structure when writing the `=related(dProduct[Price])` formula.

Now that we have all set all these things up, we can now use this information to create a Power PivotTable. If you have closed the workbook that you created previously in this chapter, you will need to open it before continuing.

To create a PivotTable from Power Pivot, click on **PivotTable** on the **Home** tab and then select **New Worksheet**. The first thing that you should notice is that all the linked tables that we created earlier have been imported, so you are able to use information from any of these tables:

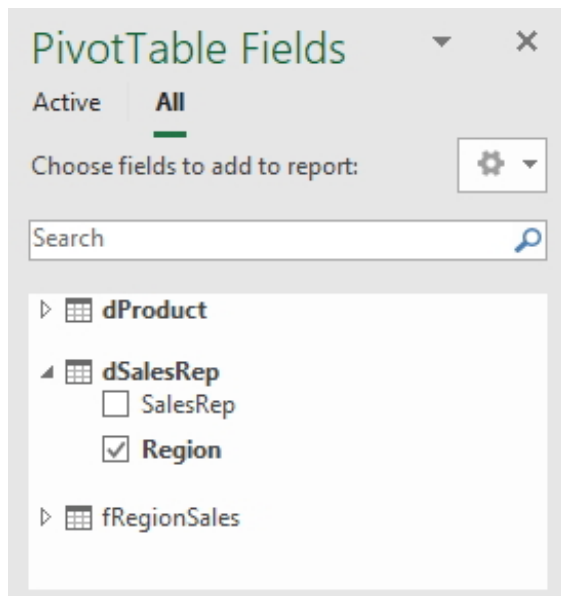


Figure 2.36 – PivotTable Fields with linked tables

I have created the following **PivotTable** with specific people from **South Region**, a few products, and **Total Net Income**:

Total Net Income												Column Labels	Grand Total
Row Labels	Dyno1		Dyno2		Dyno3		Dyno4		Dyno5				
	Bevan	Clive	Bevan	Clive	Bevan	Clive	Bevan	Clive	Bevan	Clive			
	South												
Action													
Construction	£220,097.49	£211,266.93									£165,200.76	£174,505.01	£339,705.76
Creative			£431,993.42	£440,613.85									£431,364.43
Education						£655,867.04	£627,322.12						£872,607.26
Electronic									£180,695.12	£179,362.33			£1,283,189.16
													£360,057.45
Grand Total	£220,097.49	£211,266.93	£431,993.42	£440,613.85	£655,867.04	£627,322.12	£180,695.12	£179,362.33	£165,200.76	£174,505.01	£3,286,924.06		

Figure 2.37 – Completed PivotTable

Note

Filter context is used when something or, more specifically, a measure is calculated. In our example, **Sum of Total Net Income** is filtered by the region that we have selected.

After all the work we have put in, creating a Power PivotTable was simple.

Shortcomings of Power Pivot

There are a couple of limitations of Power Pivot-driven pivot tables that you need to recognize so that you can work out if you can use them or the internal data model.

Problem 1 – selecting multiple items

The first issue is with trying to group dates into quarters or years. You could **Select Multiple Items** and then select the relevant dates that you wanted, but this is very time-consuming. Also, it is likely that human error could occur, as you might miss-click on one of them:

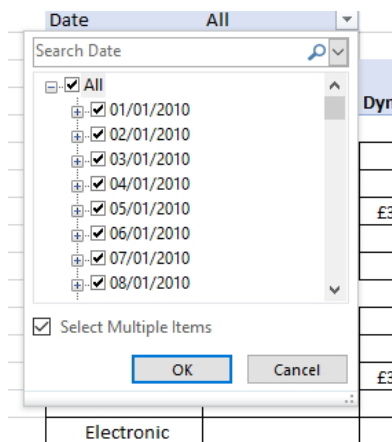


Figure 2.38 – Date problem with Power Pivot

Problem 2 – Power Pivot preview

The second problem is that in a standard pivot table, you can double-click on a cell and then see all the rows that make up that amount in that cell. In Power Pivot tables, you can only see the first 1,000 rows:

	A	B	C	D
1	Data returned for Total Net Income, East - Construction, Dyno1 (First 1000 rows).			
2				
3	fRegionSales[Date]	fRegionSales[SalesRep]	fRegionSales[Product]	fRegionSales[Unit]
4	27/02/2015	Dennis	Dyno1	
5	01/04/2014	Michael	Dyno1	
6	29/11/2014	Thelma	Dyno1	
7	25/06/2014	Jordan	Dyno1	
8	29/08/2015	Thelma	Dyno1	

Figure 2.39 – First 1,000 rows problem with Power Pivot

Problem 3 – calculated fields

The third problem that we saw in our example is that you cannot create calculated fields and items, which you can do in a regular pivot table.

Problem 4 – Microsoft Office versions

Lastly, Power Pivot-driven tables can only be refreshed or configured in Excel 2013 or newer. If the pivot table that you would like to create contains less than 1 million rows and you need to do one of the things that are mentioned here, then you are better off using the standard pivot table than trying to use the internal data model.

At the beginning of this chapter, you look at a comparison between using a VLOOKUP and slicers in terms of how Power Pivot and Query do the same thing. Hopefully, you should see how Power Pivot and Power Query work together in moving data from Excel to a more appropriate database data format.

Summary

Although we can create a pivot table relatively quickly in Excel, there are a few drawbacks that Power Query can solve. One of the biggest problems with Excel is that it can only have just over a million rows of data. The reality of this is that once you start going over a few hundred thousand, Excel becomes a bit sluggish.

Power Pivot allows us to work with huge datasets, which creates smaller and faster workbooks than standard pivot tables. One of the ways Power Pivot makes this more effective is by loading the data into the data model of Excel and not into a worksheet. We are then able to create relationships between the different tables and we do not have to worry about using VLOOKUP to create one table with everything in it.

By creating pivot tables based on this model, we can analyze multiple tables of data more easily and efficiently.

In the next chapter, we will introduce the Power Query interface, create a basic Power Query, and discover how to send data back to Excel from within Power Query.

3

Introduction to the Power Query Interface

In this chapter, you will be introduced to the Power Query interface and take a journey through its tabs, ribbons, and interface layout. During this tour, you will learn some nifty tips and tricks and best practices for transforming data. You will also gain the skills to create a basic power query and learn how to use the **View** tab to set data profiling options, as well as discover different methods on how to send reshaped data back to an Excel workbook.

The following topics are covered in this chapter:

- The Power Query window and its elements
- Creating a basic Power Query
- Discovering the **Load To...** options
- Data profiling tips

Technical requirements

Before working through this topic, you should have a solid understanding of the use of Power Query, be proficient at locating and launching Power Query through the Excel 2019 environment and saving a presentation, have a good knowledge of file management (files, folders, and file types), be familiar with the different ribbon options within Excel 2019, be able to navigate with ease around the environment, and be skilled at inserting and formatting graphic elements. It is also preferred for you do have a little database knowledge as some introductory terminology is used throughout this chapter.

The examples used in this chapter can be accessed from <https://github.com/PacktPublishing/Learn-Power-Query/>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=yXC8-jqgqXY&list=PLcLcwrwLe186O_GJEZs47WaZXWZjwTN83&index=4&t=1s.

The Power Query window and its elements

This section will introduce you to the Power Query window. You will become familiar with the elements in it, learning how to understand how its different parts function.

As mentioned in previous chapters, Power Query is launched via the **Get & Transform** group, located in the **Data** tab of the Excel ribbon. Once you have clicked on **Get Data**, select the **Launch Power Query Editor** option from the drop-down list. The Power Query editor will load as a separate window over the Excel 2019 interface:

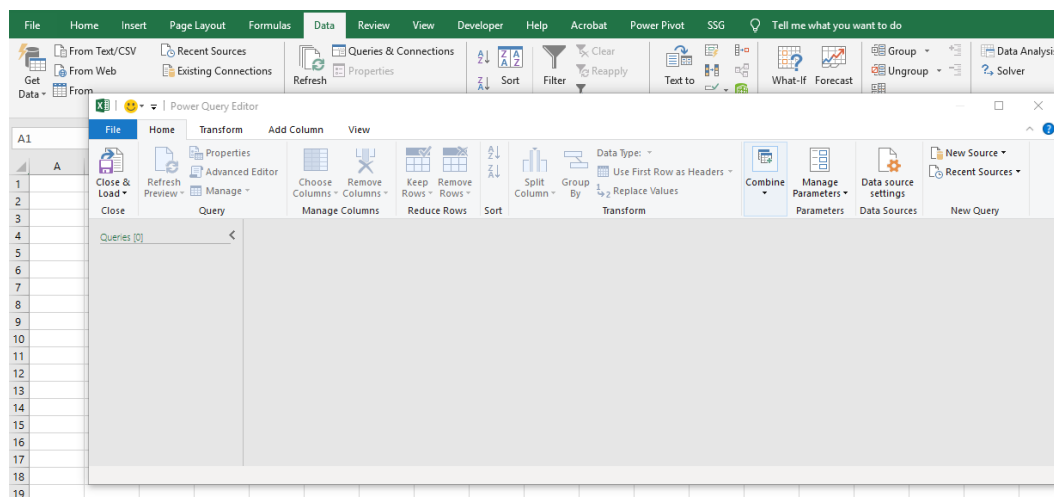


Figure 3.1 – The Power Query window without any data loaded

Note

The various methods of accessing Power Query have already been described in previous chapters.

We will cover the following Power Query elements in the upcoming sections:

- The main ribbon and tabs
- The navigation pane (the **Queries** list)
- Data table preview
- The **Query Settings** pane

Along with these elements, we will cover the following feature elements in Power Query:

- Working with **APPLIED STEPS**
- Investigating the **View** settings
- Using **Advanced Editor**

After covering these topics, we will be ready to create our first basic Power Query. So, let's get started!

The main ribbon and tabs

The main ribbon contains access to all the features within Power Query that allow you to mechanize processes to clean, transform, and shape data from different sources.

Any time an icon is used from the main ribbon, the step that is carried out on the data loaded within Power Query is stored and made available for editing or deleting at any point. This is especially useful when preparing data in Power Query to be ready for analysis or summary using other tools such as PivotTables or dashboard reporting.

Moving on, there are only five tabs in Power Query that guide you in accessing the program functions. They are **File**, **Home**, **Transform**, **Add Column**, and **View**:



Figure 3.2 – Tabs

Familiarize yourself with the content of each of the tabs, ensuring you know exactly where to go to perform a certain action in the program.

The navigation pane or the Queries list

This pane displays the queries that you have created. Any new data sources or queries that are added to be created in Power Query will be displayed in this pane:

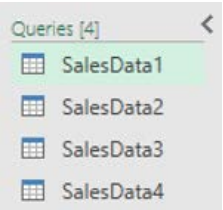


Figure 3.3 – The Queries list pane

Let's see how we can add/create a new query here from the Power Query window.

Adding a new query using the navigation pane

Perform the following steps to add a new query using the navigation pane:

1. Right-click with your mouse pointer on the background of the **Queries** list (from the navigation pane).
2. Click on the **New Query** option from the shortcut menu provided.
3. Choose a data source from the list provided, and if relevant, choose a further option to create the new query:

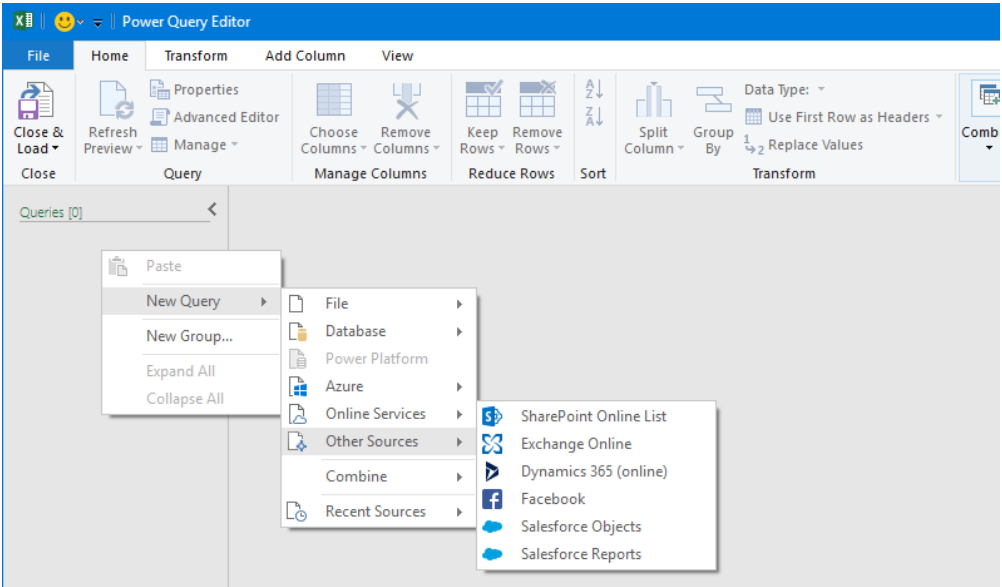


Figure 3.4 – Adding a new query using the navigation pane

You may want to edit or modify an existing query. The next section covers that.

Editing with the query options

There are also options to edit queries by right-clicking on a query name in the navigation pane. A shortcut menu will appear, where you can find options for renaming, deleting, duplicating, sorting, or even creating a function, as well as accessing **Advanced Editor**:

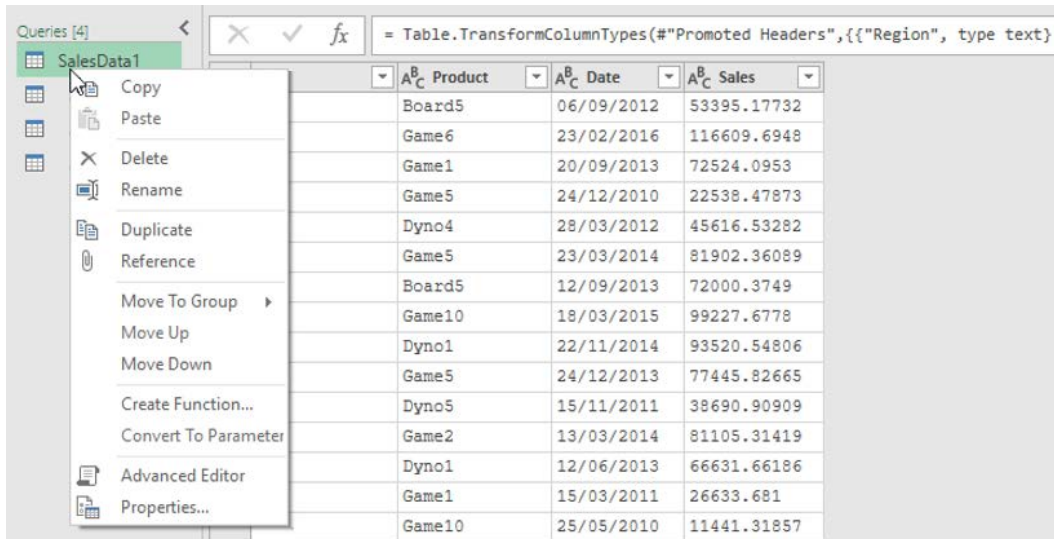


Figure 3.5 – The query right-click options

If you are working with a number of queries and some relate to the same topic, department, or region, you may want to group them together. Let's see how.

Grouping queries

This is simply a way of organizing your queries in the navigation pane. Creating a new group can be achieved either by right-clicking on the empty navigation pane background and selecting **New Group** from the shortcut menu, or by selecting the queries first, as follows:

1. Select the queries in the navigation pane that you wish to group by pressing *Ctrl* and clicking on each one.

2. Right-click on the selected queries:

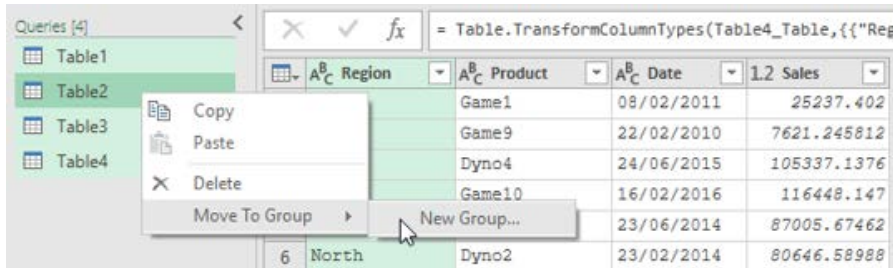


Figure 3.6 – Right-click to create a new group

3. Select **New Group**.
4. Name the group **SalesData** in the placeholder provided, and enter a description, if appropriate:

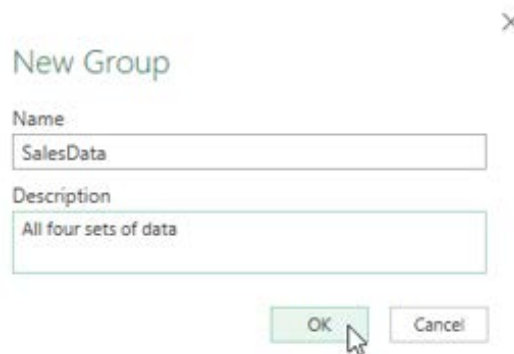


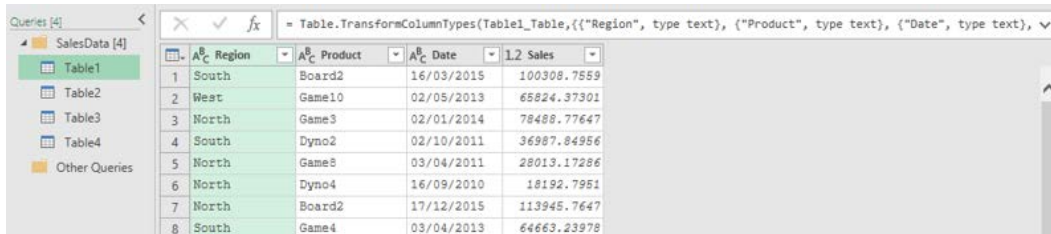
Figure 3.7 – Naming a group

Tip

I find adding a description especially useful as when you load the data back to Excel, you can then see the description in the **Queries & Connections** pane for reference.

5. Click on **OK** to save the group.

6. The group is displayed in the navigation pane along with the main group name, SalesData, at the top. The four tables are listed as follows:



	Region	Product	Date	Sales
1	South	Board2	16/03/2015	100308.7559
2	West	Game10	02/05/2013	65824.37301
3	North	Game3	02/01/2014	78488.77647
4	South	Dyno2	02/10/2011	36987.84956
5	North	Game8	03/04/2011	28013.17286
6	North	Dyno4	16/09/2010	18192.7951
7	North	Board2	17/12/2015	113945.7647
8	South	Game4	03/04/2013	64663.23978

Figure 3.8 – The new group result

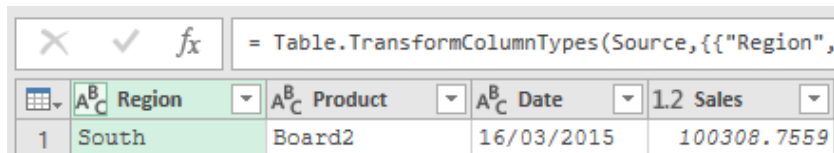
7. The group can be expanded or collapsed using the arrow to the left of the yellow group folder icon.

Data table preview

This is the largest window within Power Query and shows a preview of a selected query from the **Queries** list pane, as well as the state of the query data according to the applied steps shown on the right. It is the view you refer to when rearranging and shaping data to arrive at your data analysis requirements.

At the top of the preview pane, you will see the column headers, and down the left-hand side, you will see the row numbers. Each column header also displays the data type to the left of the column name (refer to *figure 3.12* for an example view).

In the following example, the column headers are labeled Region, Product, Date, and Sales. When you place the mouse pointer over the data type, a square border appears around the data type icon:



	Region	Product	Date	Sales
1	South	Board2	16/03/2015	100308.7559

Figure 3.9 – The column data types

Clicking on this icon will populate a drop-down list of data type options relating to the data in the column you currently have selected. Note that the **Using Locale...** option at the bottom of the drop-down list will allow you to choose a convention from a different location, such as **English (United Kingdom)**:

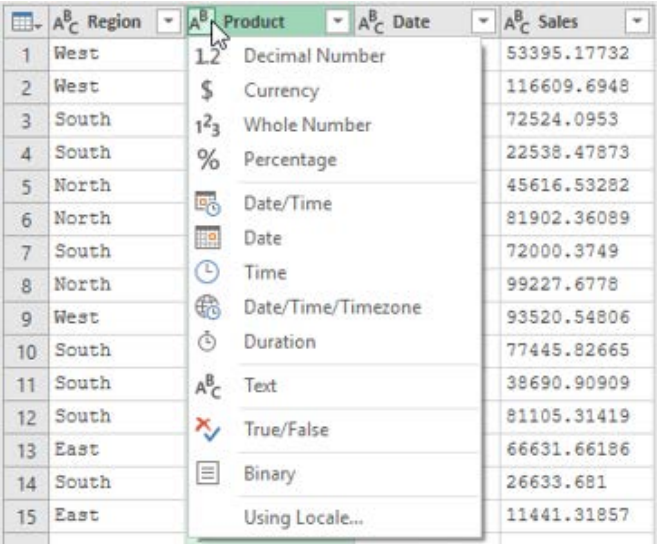


Figure 3.10 – The data type drop-down list

Another method to change the data type is to visit the **Data Type:** drop-down list icon, located in the **Home** tab of the **Transform** group. Remember to select the column you wish to alter first:

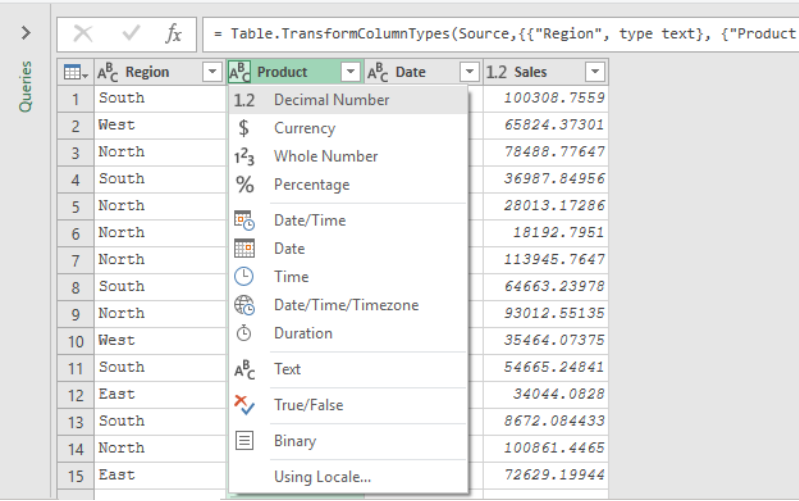


Figure 3.11 – The Data Type: tab option

Navigating and selecting columns, rows, and the data table works exactly the same as you would in Excel 2019. You can use the *Ctrl + A* keys to select the entire table in the preview window.

The Query Settings pane

The **Query Settings** pane is located at the right in the Power Query window. It is separated into two sections, namely **Properties** and **APPLIED STEPS**. If you do not see the **Query Settings** pane toward the right in the preview pane, then you will need to activate it by clicking on the **View** tab and then selecting the **Query Settings** icon:

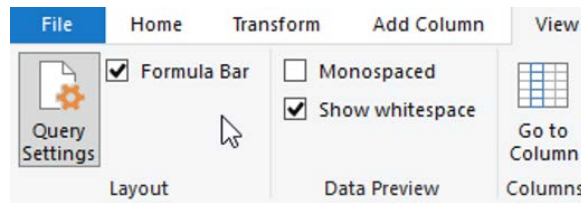


Figure 3.12 – Query Settings

Next, we will see how to change this property.

Changing the query properties

When you open data in Power Query, the **Properties** pane will reflect a table name, which you can rename to suit the nature of your query. When you close and load the transformation back to the Excel table, the new table will take on the same name as the source table from Power Query:

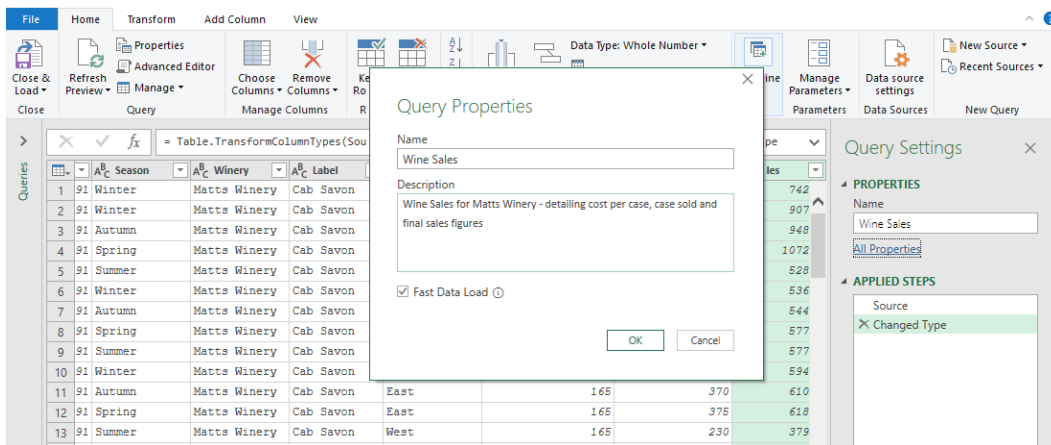


Figure 3.13 – Table properties

The table name is used in the Power Query M code when referencing it.

Working with APPLIED STEPS

APPLIED STEPS is where you will notice all the recorded steps undertaken on the data within the query. These steps are added and labeled whenever you use a feature from the ribbon in the sequence of an application. Steps can include removing a column, making a data connection, or changing the header row; all of these steps are part of shaping data, which constitutes the creation of queries.

Steps can be removed by pressing the *Ctrl + Z* undo keyboard shortcut or by clicking on the **x** icon to the left of the recorded step to remove the last step. When placing the mouse pointer over **x** to the left of the chosen step, the **x** icon will turn red:



Figure 3.14 – Removing a query step

Each step is recorded by a command; these commands are called M queries, which is the language underlying the commands listed behind the ribbon icons used to perform these steps.

Deleting multiple steps

Instead of deleting steps one after the other, which is time-consuming, you can select one point in the **APPLIED STEPS** section to delete until the end. Right-click on a step in the **APPLIED STEPS** list box, then click on **Delete Until End**:

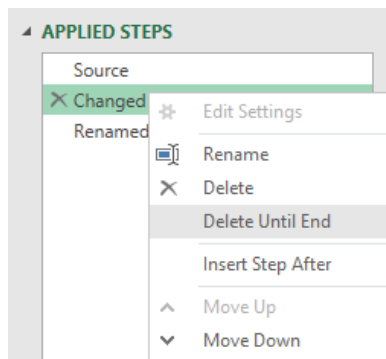


Figure 3.15 – Deleting multiple steps

Next, let's see how to document and rename these steps.

Documenting and renaming steps

This section is extremely important and provides a workflow that you should definitely get used to when working in Power Query. The reason for this is that it helps you keep an accurate record of why you applied certain steps for your own benefit. Furthermore, when working with others on the same set of data or having to pass queries on to other colleagues, everyone can see the documented steps available, which will help them understand why a step was performed.

Any description added to the step properties will be added automatically to the code, which is visible in the **Advanced Editor** window:

Note

You do not need to add any syntax to indicate a code comment; Power Query does this for you.

1. Right-click on a step in the **APPLIED STEPS** list, then click on **Properties...**:

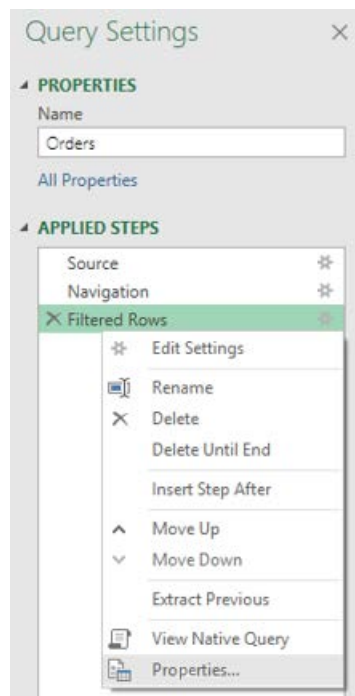


Figure 3.16 – Adding a step description

2. The **Step Properties** dialog box will populate, where you can enter a description for the named step:

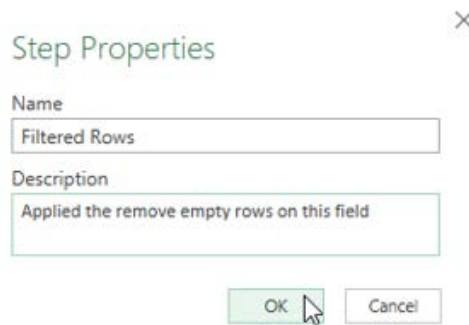


Figure 3.17 – The Step Properties dialog box

3. Click on the **OK** command when you have finished to save the description and return to the **APPLIED STEPS** list.

Apart from adding a step description, it is equally important to understand what each step means when glancing at the list of applied steps.

Often, the step names might not be descriptive enough, so you may need to rename them:

1. Right-click on a step in the **APPLIED STEPS** list.
2. Choose **Rename** from the drop-down list provided.
3. Rename the step by typing a new name into the placeholder provided.
4. Press the *Enter* key to confirm the new name.

If you are not the only person working on a workbook with its associated queries, then renaming steps is something to consider.

Investigating the View settings

We will now look at three important settings located in the **View** tab of the Power Query ribbon:

- Viewing the formula bar
- Changing the font
- Showing query relationships

Let's consider each one in the following sections.

Viewing the formula bar

It is always a good idea to have the formula bar visible when working in Power Query. Although you may not know anything about the Power Query language (M), you will no doubt become very familiar with its commands as you shape data because the formula bar will show any changes to the syntax. This is a perfect way to learn Power Query coding:

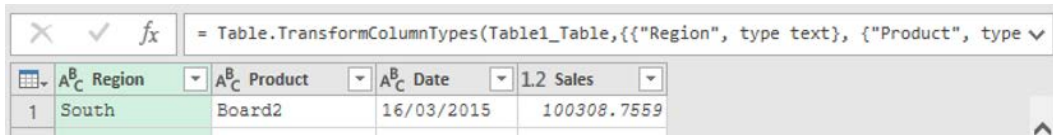


Figure 3.18 – The formula bar

If you can't see the formula bar when you open Power Query, click on the **View** tab and locate the **Formula Bar** icon from the **Layout** group. Check the checkbox to show the formula bar:

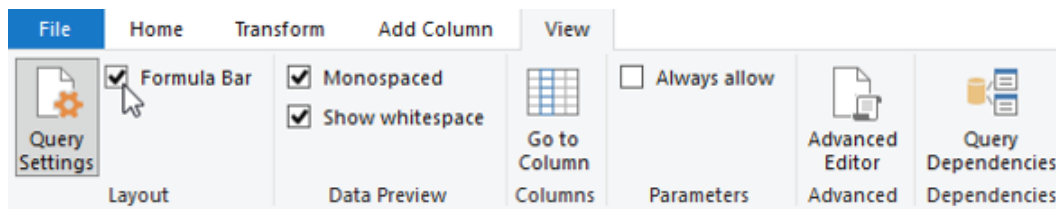


Figure 3.19 – The View settings

Next, we will see how to change the font.

Changing the font

Monospacing is a font that applies uniform spacing to all characters and can help tremendously when dealing with column data such as a long product code as you can see whether all the characters line up. This is perfect to see if and where you can split the content of a field consistently:

1. Open a query in Power Query.
2. Click on the **View** tab, locate the **Data Preview** group, then click on the **Monospaced** icon to activate it. Monospaced fonts will now show in the preview window.

- Note the difference in the spacing. This screenshot shows the default Power Query font:

Product	Product Code	Date	1.2 Sales
Board5	E729TFO605258916421-600PT	06/09/2012	53395.1773
Game6	I883VXW767844292755-246GX	23/02/2016	116609.6948
Game1	P514YXZ415181521507-575CU	20/09/2013	72524.0953
Game5	I938LNZ479834468789-860ON	24/12/2010	22538.4787
Dyno4	H247MKU458573986837-770SK	28/03/2012	45616.5328
Game5	V777ZJI221878500906-685EV	23/03/2014	81902.3609
Board5	Z642BCU387684408203-808CS	12/09/2013	72000.3749
Game10	K756LFU280391331780-387FH	18/03/2015	99227.6778

Figure 3.20 – Product codes without monospacing

The following screenshot shows the monospaced font applied:

Product	Product Code	Date	1.2 Sales
Board5	E729TFO605258916421-600PT	06/09/2012	53395.1773
Game6	I883VXW767844292755-246GX	23/02/2016	116609.6948
Game1	P514YXZ415181521507-575CU	20/09/2013	72524.0953
Game5	I938LNZ479834468789-860ON	24/12/2010	22538.4787
Dyno4	H247MKU458573986837-770SK	28/03/2012	45616.5328
Game5	V777ZJI221878500906-685EV	23/03/2014	81902.3609

Figure 3.21 – Product codes with monospacing

We can clearly see the alignment of the characters with monospacing applied, which makes it a little easier to see where you could, for instance, split the column.

Showing query relationships

At the end of the **View** tab ribbon, you will find the **Query Dependencies** icon. This icon is very useful when working on a data model you did not create but now have to work on. It displays all the tables within the model and shows how the tables relate to each other:

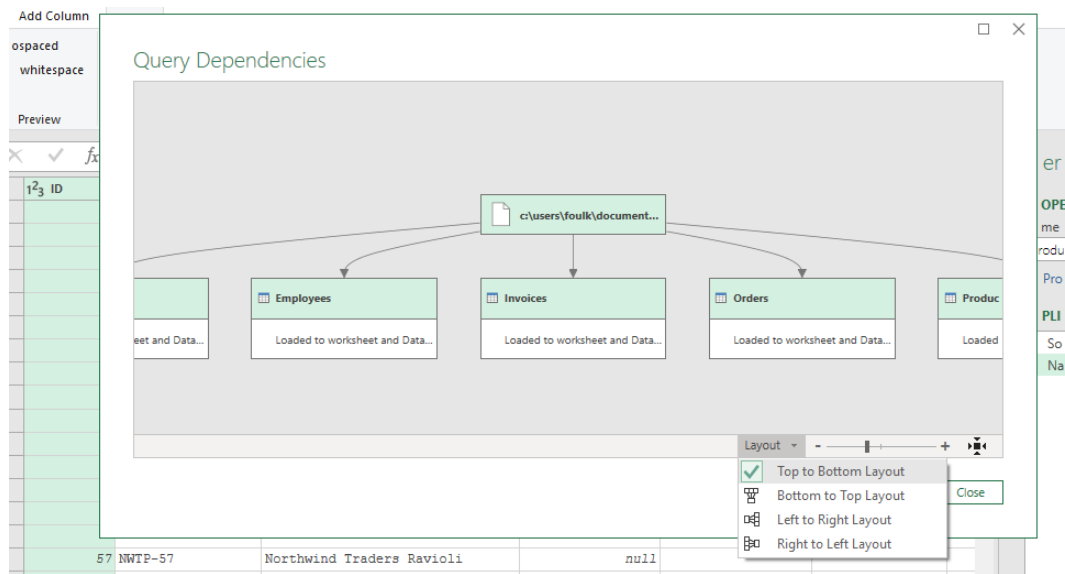


Figure 3.22 – Query Dependencies

The **Query Dependencies** window displays all the relationships between queries. To view all the schematic dependency relationships, click on the **Fit to Screen** icon in the bottom right-hand corner of the window:

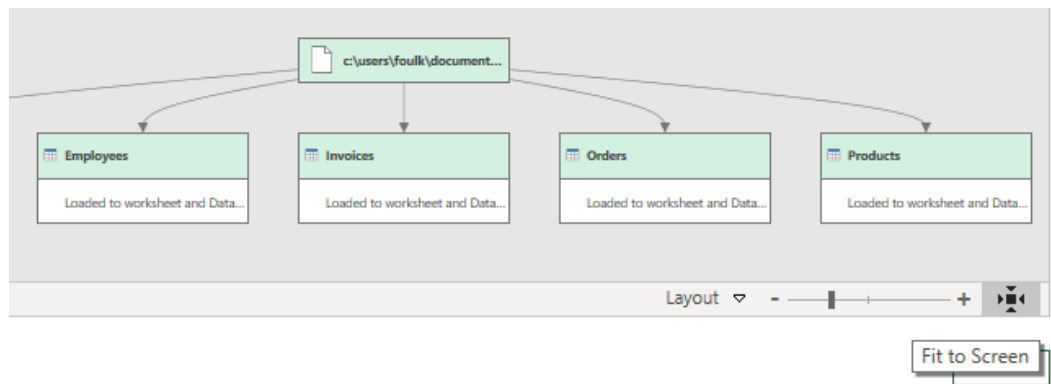


Figure 3.23 – Fit to Screen

To change the layout of the scheme, click on the **Layout** drop-down list icon, then select a layout from the list provided:

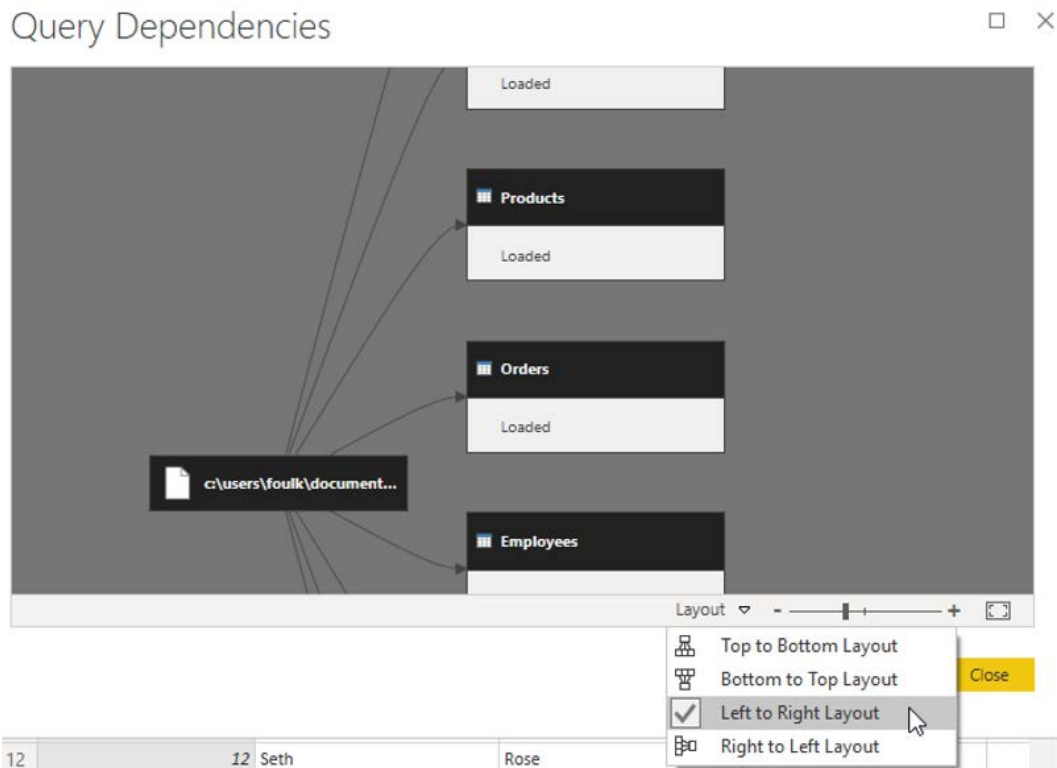


Figure 3.24 – The layout scheme

Now, we will see how to use the **Go to Column** feature.

Using the Go to Column feature

The **Go to Column** option is a very useful tool when working with large tables and if you want to locate a column quickly. Use the **Go to Column** icon, which is located in the **View** tab under the **Column** group:

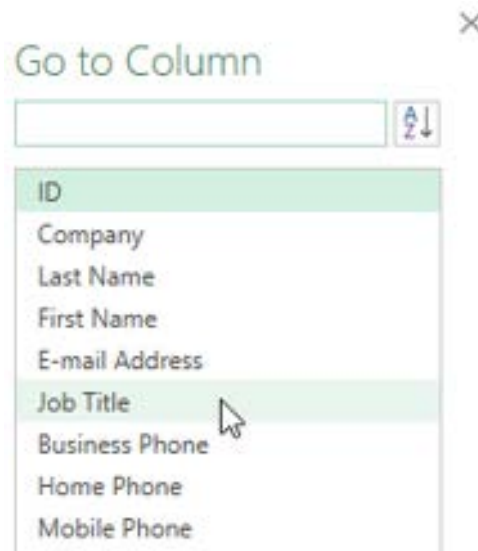


Figure 3.25 – Searching for a column quickly

Now that we have seen how View works, let's move on to **Advanced Editor**.

Using Advanced Editor

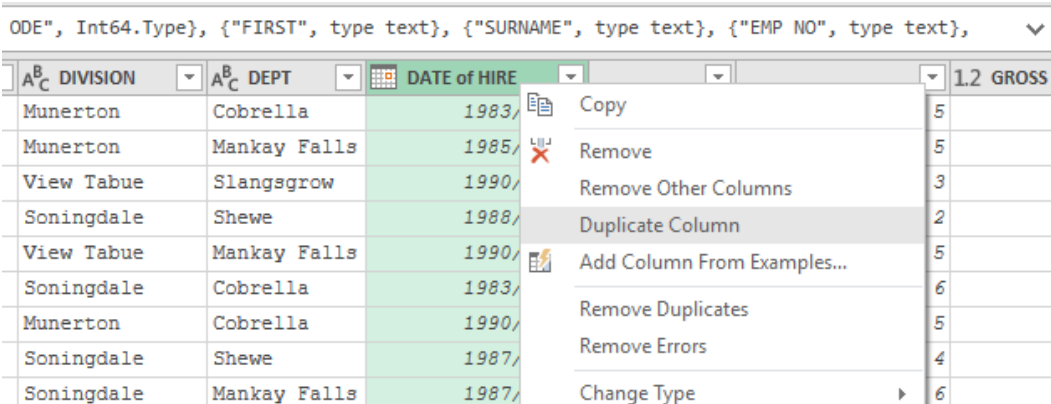
Each time you make an adjustment to data within Power Query, you compile lines of code. These lines of transformation code are stored in the **Advanced Editor** window, where you can create and edit queries using the M Power Query language.

It is crucial to understand **Advanced Editor**. The reason I say this is that you may edit code, reorder columns, and complete a few steps on data, but then when you get to the last step, find that there is an error. To locate the error, you would need to visit the **Advanced Editor** window, and having an understanding of how it works is crucial.

As a simple example introduction to **Advanced Editor**, we will complete a few steps to shape some data and then use **Advanced Editor** to rename a column in a previous step. This will cause an error in our code and we would need to understand why this has happened in order to fix it:

1. Using the `SSGFilter.xlsx` workbook, we will transform the data table in the workbook using Power Query.
2. From the **Data** tab, locate the **From Table/Range** icon under the **Get & Transform** group.

3. We are now working in Power Query. Duplicate the **DATE of HIRE** column so that we can split it:



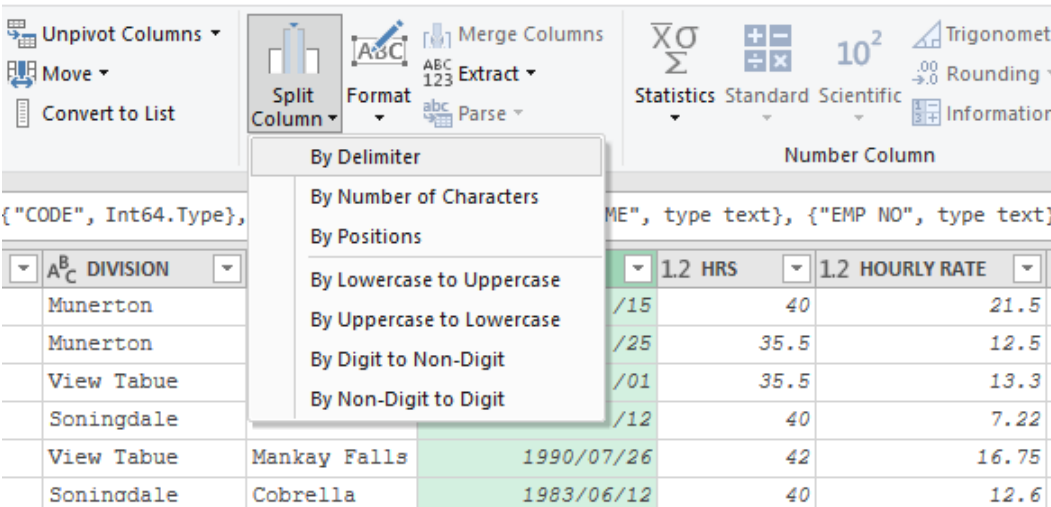
ODE", Int64.Type}, {"FIRST", type text}, {"SURNAME", type text}, {"EMP NO", type text},

DIVISION	DEPT	DATE of HIRE	1.2 GROSS
Munerton	Cobrella	1983/	5
Munerton	Mankay Falls	1985/	5
View Tabue	Slangsgrow	1990/	3
Soningdale	Shewe	1988/	2
View Tabue	Mankay Falls	1990/	5
Soningdale	Cobrella	1983/	6
Munerton	Cobrella	1990/	5
Soningdale	Shewe	1987/	4
Soningdale	Mankay Falls	1987/	6

Copy
Remove
Remove Other Columns
Duplicate Column
Add Column From Examples...
Remove Duplicates
Remove Errors
Change Type

Figure 3.26 – Duplicating a column

4. Click to select the **DATE of HIRE – Copy** column.
5. We will use the **Split Column** icon to split this column, using the **Delimiter** function to break up its data:



Unpivot Columns
Move
Convert to List

Split Column
Format
Merge Columns
Extract
Parse

By Delimiter
By Number of Characters
By Positions
By Lowercase to Uppercase
By Uppercase to Lowercase
By Digit to Non-Digit
By Non-Digit to Digit

Statistics
Standard
Scientific
Trigonomet
Rounding
Information

Number Column

ME", type text}, {"EMP NO", type text}

DIVISION	DEPT	DATE of HIRE	1.2 HRS	1.2 HOURLY RATE
Munerton			/15	40
Munerton			/25	35.5
View Tabue			/01	35.5
Soningdale			/12	40
View Tabue	Mankay Falls	1990/07/26	42	16.75
Soningdale	Cobrella	1983/06/12	40	12.6

Figure 3.27 – Splitting a column

6. Choose to split with the / delimiter and choose **Left-most delimiter** for the split:

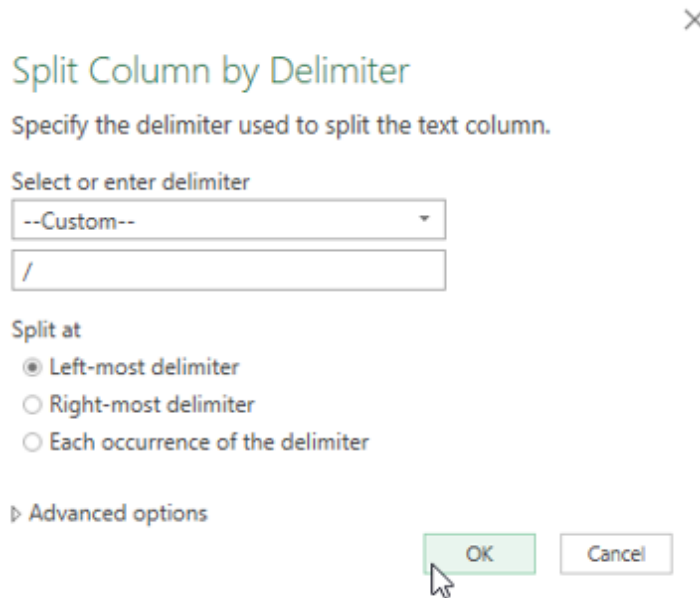


Figure 3.28 – The Split Column by Delimiter specifications

7. Click on **OK** to continue.
8. Remove the extra column by right-clicking on the column and choosing **Remove**, as we only need the Year column.
9. Rename the Year column Year Hired by double-clicking on the column header and typing a new name into the placeholder. Then, press *Enter* to confirm the change.

10. Notice that the steps have been building up in the **APPLIED STEPS** pane to the right of the preview:

DATE of HIRE - Copy.3"}})				
	1.2 HOURLY RATE	1.2 GROSS PAY	1 ² 3 Year	1 ² 3 D
40	21.5	860	1983	
35.5	12.5	443.75	1985	
35.5	13.3	472.15	1990	
40	7.22	288.8	1988	
42	16.75	703.5	1990	
40	12.6	504	1983	
40	21.5	860	1990	
35	24	840	1987	
40	12.6	504	1987	
35.5	12.5	443.75	1986	
35.5	13.3	472.15	1985	
32	5.5	176	1990	
35.5	13.3	472.15	1984	
40	8.75	350	1988	

Query Settings

PROPERTIES
Name
Table1
[All Properties](#)

APPLIED STEPS

- Source
- Navigation
- Changed Type
- Duplicated Column
- Split Column by Delimiter
- Changed Type1
- Renamed Columns
- Removed Columns**

Figure 3.29 – APPLIED STEPS

11. Click on the **Advanced Editor** icon, located in the **Home** tab, to view the code that has been compiled from the steps we performed on the data:

Advanced Editor

Table1

```
let
    Source = Excel.CurrentWorkbook(){[Name="Table1"]}[Content],
    #"Changed Type" = Table.TransformColumnTypes(Source,{{"CODE", Int64.Type}, {"FIRST", type text}, {"SURNAME", type text}, {"EMP NO",
    #"Duplicated Column" = Table.DuplicateColumn(#"Changed Type", "DATE of HIRE", "DATE of HIRE - Copy"),
    #"Split Column by Delimiter" = Table.SplitColumn(Table.TransformColumnTypes(#"Duplicated Column", {{{"DATE of HIRE - Copy", type text
    #"Changed Type1" = Table.TransformColumnTypes(#"Split Column by Delimiter",{{{"DATE of HIRE - Copy.1", Int64.Type}, {"DATE of HIRE -
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type1",{"DATE of HIRE - Copy.3", "DATE of HIRE - Copy.2"}),
    #"Renamed Columns" = Table.RenameColumns(#"Removed Columns",{{"DATE of HIRE - Copy.1", "Year Hired"}})
in
    #"Renamed Columns"
```

✓ No syntax errors have been detected.

Done Cancel

Figure 3.30 – Advanced Editor

12. Take a moment to glance through the steps to understand how they have been formulated in **Advanced Editor**; notice the labels that the code has produced for each step.
13. Edit the Year Hired label in the last step of the **Advanced Editor** window to read YEAR HIRED (change the casing):

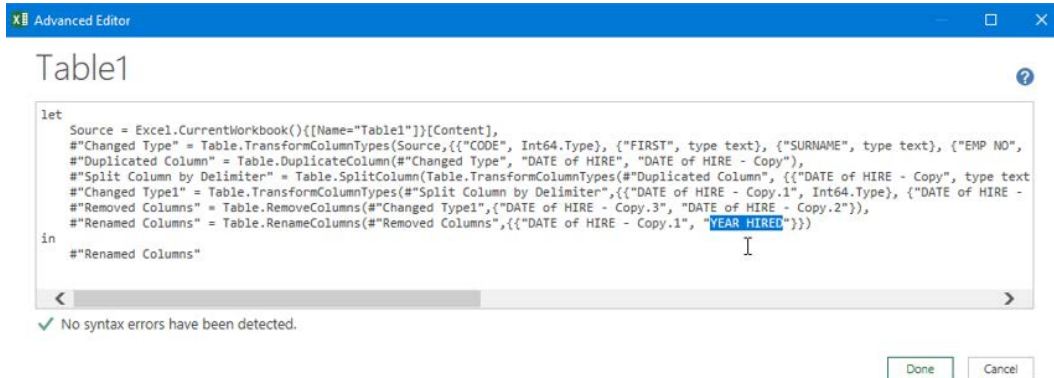


Figure 3.31 – The Advanced Editor labels

- Click on **OK** when you are done and you will see the updated data.
- Move the `YEAR HIRED` column and place it just after the `DATE of HIRE` column:

= Table.RenameColumns(#Removed Columns,{"Year", "YEAR HIRED"})						
ABC DEPT	DATE of HIRE	1 YEAR HIRED	JRLY RATE	1.2 GROSS PAY	1 ² 3 YEAR HIRED	1 ² 3
Cobrella	1983/04/15	40	21.5	860	1983	
Mankay Falls	1985/01/25	35.5	12.5	443.75	1985	
Slangsgrow	1990/02/01	35.5	13.3	472.15	1990	
Shewe	1988/05/12	40	7.22	288.8	1988	
Mankay Falls	1990/07/26	42	16.75	703.5	1990	
Cobrella	1983/06/12	40	12.6	504	1983	
Cobrella	1990/12/30	40	21.5	860	1990	
Shewe	1987/06/05	35	24	840	1987	

Figure 3.32 – Moving a column

16. We forgot to change the column name of **DATE of HIRE** to **HIRE DATE**.
Double-click on **DATE of HIRE** and type **HIRE DATE**, then press *Enter* to confirm.

17. Click on **Insert** to confirm the step. The data looks absolutely fine, with no errors present.

Now, if you click on the last step in the **APPLIED STEPS** pane, you will notice an error immediately appears in the preview window. This error highlights the problem label and gives an indication of the problem with a description.

18. To the right of the error, you will notice a **Go to Error** button, which you can use to show more details about the error. But ultimately, the best thing to do is to visit **Advanced Editor** to identify the code problem:

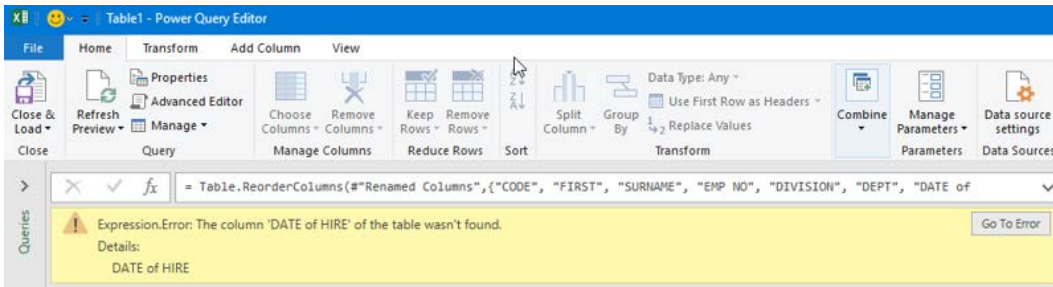


Figure 3.33 – A code error

This is where you will start to understand the code, how it is constructed, and the importance of the order of steps when reshaping data:

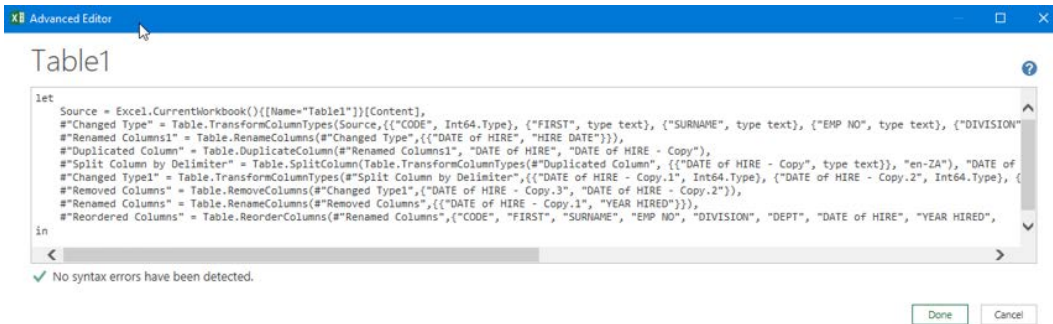


Figure 3.34 – An Advanced Editor code sample

19. Locate the error columnn:



Figure 3.35 – Problem identified

Update the column name that is causing the problem in the editor:

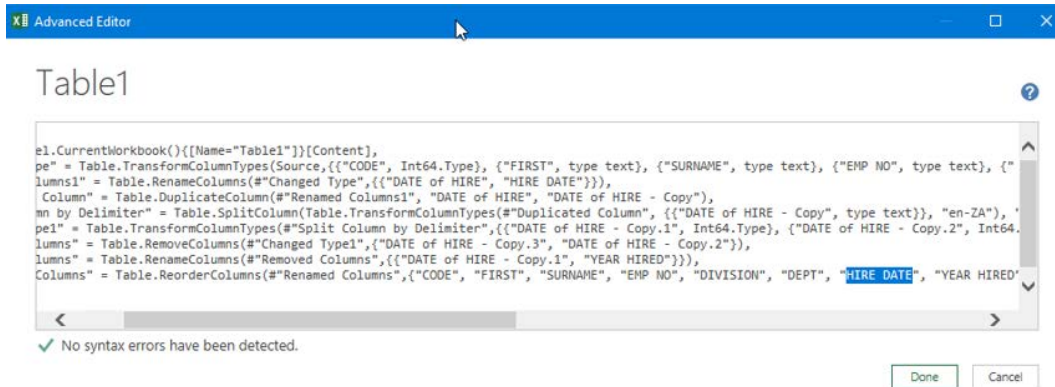


Figure 3.36 – Problem solved

20. Click on **Done** when complete and the error in the preview window should now be fixed and the data will once again be shown.

In the next section, we will see how to add a comment to the **Advanced Editor**.

Adding a comment to the Advanced Editor window

When you are working with code, it is necessary to add comments, either for your own reference or to share with others who have access to the data. Although you will learn more about the Power Query language later in this book, we will introduce commenting here:

1. Open the **Advanced Editor** window.
2. Click at the top of the code.
3. For a single-line comment, type the following onto a new line, then press *Enter* to move to the next line:

```
//this is my comment
```

4. For a multi-line comment, type the following, moving to the next line to continue the comment, after which you need to close the comment:

```
/*this is my
```

```
multi-line comment*/
```

This is how it appears on screen:



Figure 3.37 – Commenting in Advanced Editor

Next, we will add a comment using the formula bar.

Adding a comment using the formula bar

Comments can be made for each step in the code process using the formula bar. This is a very quick method of adding on-the-spot descriptions for steps you have just completed using Power Query. These comments, although added in the formula bar, will only appear in the **Advanced Editor** code after you have added them:

1. Locate and select the **HIRE DATE** column.
2. Click to select the column header and the code will display in the formula bar above.
3. Type the following at the end of the code in the formula bar:

```
/*changed to Date as Time not relevant here*/
```

This is how it appears in the formula bar:

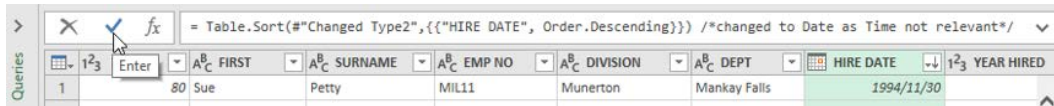


Figure 3.38 – Comments in the formula bar

4. Click on the tick to enter the comment or press *Enter* on the keyboard.

The comment is now added to the **Advanced Editor** window and is removed from the HIRE DATE formula bar code

Keeping comments visible in the formula bar code

To keep commenting visible in the formula bar for a particular step so that it is easier to refer to when working with complex transformations, simply add the comment in the code, rather than at the end of the code, as in the following example:



Figure 3.39 – Formula bar comments

This concludes our tour of all the pertinent elements of Power Query. It's now time to learn how to create a power query with Excel.

Creating a basic power query

In the topics we have covered so far, you learned a lot about the Power Query interface and mastered a number of layout techniques and tips. We will now go through the steps to create a basic power query using an Excel data table from scratch.

First, you need to determine where your data is coming from. For this example, we are going to use the workbook called `MattsWinery.xlsx`:

1. Once you have opened the workbook, press `Ctrl + T` to launch the **Create Table** dialog box for the Excel data (alternatively, use the **Table/Range** icon in the **Get & Transform** group):

	A	B	C	D	E	F	G	H
1	Year	Season	Winery	Label	Region	Cost Per Case	Cases Sold	Sales
2	1991	Winter	Matts Winery	Cab Savon	North	£ 165.00	£ 450.00	£ 74 250.00
3	1991	Winter	Matts Winery	Cab Savon	North	£ 165.00	£ 550.00	£ 90 750.00
4	1991	Autumn	Matts Winery	Cab Savon	North	£ 165.00	£ 575.00	£ 94 875.00
5	1991	Spring	Matts Winery	Cab Savon	North	£ 165.00	£ 650.00	£ 107 250.00
6	1991	Summer	Matts Winery	Cab Savon	South	£ 165.00	£ 320.00	£ 52 800.00
7	1991	Winter	Matts Winery	Cab Savon			5.00	£ 53 625.00
8	1991	Autumn	Matts Winery	Cab Savon			0.00	£ 54 450.00
9	1991	Spring	Matts Winery	Cab Savon			0.00	£ 57 750.00
10	1991	Summer	Matts Winery	Cab Savon			0.00	£ 57 750.00
11	1991	Winter	Matts Winery	Cab Savon			0.00	£ 59 400.00
12	1991	Autumn	Matts Winery	Cab Savon			0.00	£ 61 050.00
13	1991	Spring	Matts Winery	Cab Savon			5.00	£ 61 875.00
14	1991	Summer	Matts Winery	Cab Savon			0.00	£ 37 950.00
15	1991	Winter	Matts Winery	Cab Savon			5.00	£ 38 775.00
16	1991	Autumn	Matts Winery	Cab Savon	West	£ 165.00	£ 240.00	£ 39 600.00
17	1991	Spring	Matts Winery	Cab Savon	West	£ 165.00	£ 260.00	£ 42 900.00

Figure 3.40 – Creating a data table

2. Check that the range selected is the range that you need to use to transform your data.
3. Just under the selected range, check that the **My table has headers** checkbox is selected so that Excel identifies the top row as the header row.
4. Click on the **OK** button to change the worksheet data into a table and launch the Power Query window. The Power Query window is now visible over the Excel worksheet.
5. The first thing we need to do is change the default name given to the table—let's name it `Wine Sales`.

- It is always good practice to add a description of the table, so go to **All Properties** just below the table name placeholder to add the description, then click on **OK** when done:

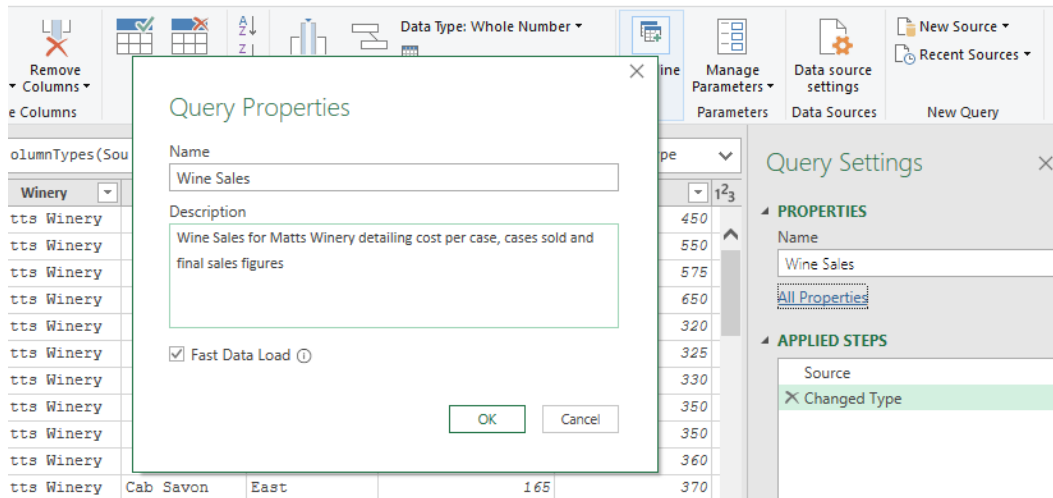


Figure 3.41 – Query description

Now, you are ready to shape and clean the table data.

Observe the following query:

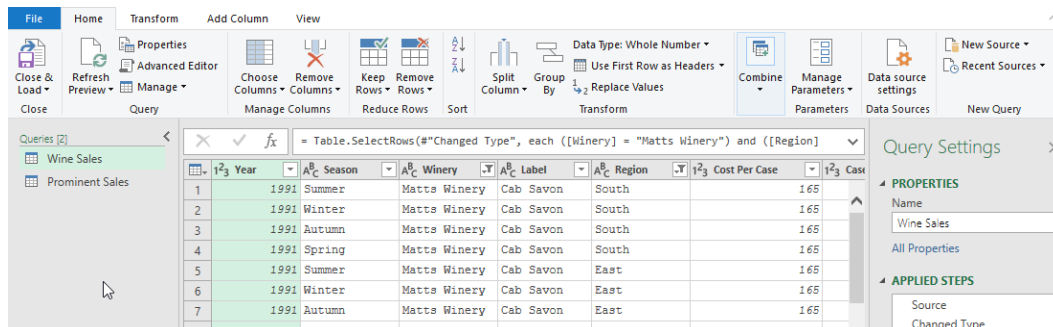


Figure 3.42 – This is how your query will look before the next steps

We will learn all about the different features to do so over the following chapters, but for now, we will just go through a few steps so that you get an idea of how it works:

- Change the **Region** and **Season** columns to uppercase by clicking on the **Transform** tab, locating the **Text Column** group, selecting the **Format** icon, and then selecting **UPPERCASE**.

2. Duplicate the Wine Sales query by clicking on the query navigation pane, right-clicking on the **Wine Sales** query, and selecting **Duplicate**.
3. Rename Wine Sales to Matts Sales and the second query to Prominent Sales, then filter the Winery column so that each field only displays its winery.

Once you have finished transforming your data, you will need to send it back to Excel for further analysis:

1. Click on the **File | Close and Load** option (we will explain all the load options in the next section).
2. Once the queries are loaded back to Excel, you will notice the **Queries & Connections** pane to the right of the workbook data:

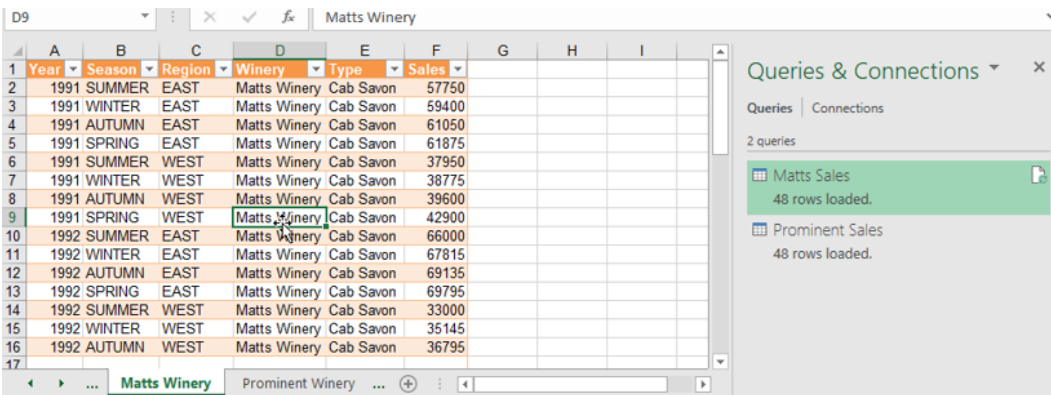


Figure 3.43 – The worksheet and the Queries & Connections pane

Queries are added to separate worksheets automatically as the **Close and Load** option was selected for this example.

3. The **Queries & Connections** pane allows you to see all the queries in Power Query. Double-click on a query to open it in Power Query and continue the transformation process.
4. If you hover the mouse pointer over a query in the **Queries & Connections** pane, a snapshot of the query will be visible with a preview of the data and general information about the query, as well as the option to view the query in the worksheet or edit it in Power Query:

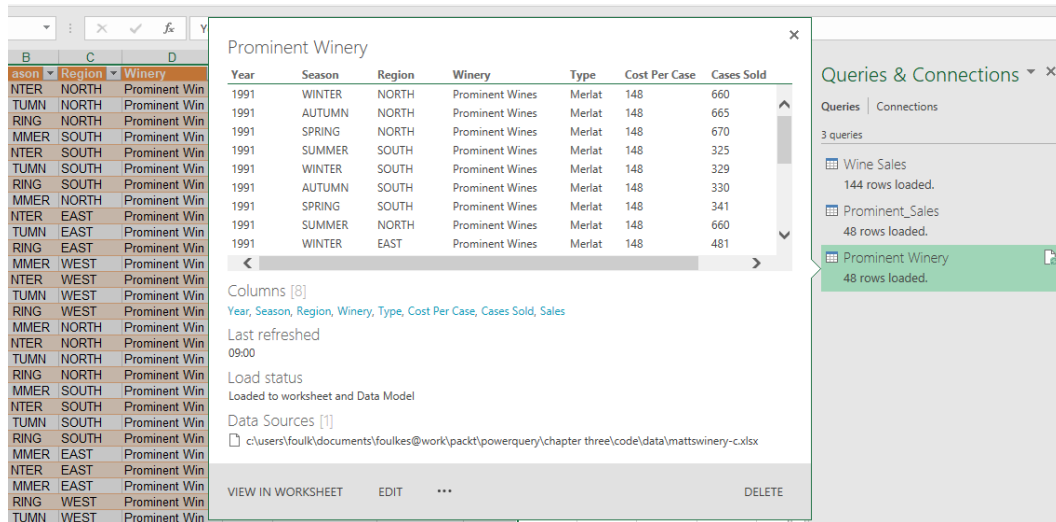


Figure 3.44 – Query preview and related information

5. You will also notice three dots next to the **Edit** option at the bottom of the preview pane. Clicking on this will open a shortcut menu of further actions to take for your query.
6. To the right of the preview pane, you can delete a query.
7. If new data is added to the worksheet in Excel, you can click on the relevant query in the **Queries & Connections** pane, and then select the **Refresh** icon on the very right of the query name to pull in the new data to Power Query.

Now that we can create a simple power query, we will look at the **Load To...** options in the next section to learn how to send this data back to Excel.

Discovering the Load To... options

In this section, we will explore the options in the **Load To...** dialog box and learn about the different ways that data can be imported from Power Query back to an Excel workbook.

Let's first take a look at the default custom load settings in Excel.

Changing the default custom load settings

Standard load options are set in the Excel application. We can visit the **Data** tab in Excel 2019 to explore and change these default settings. The default settings are listed as follows, but can be changed by visiting the default options:

- Single queries are loaded to a new workbook.
- If you are loading multiple queries, they will load to the data model automatically.

Let's investigate where we can change these default load settings:

1. In Excel 2019, click on the **Data** tab.
2. If you are already in Power Query, select the **File** tab to access **Options and settings**, then select **Query Options**:

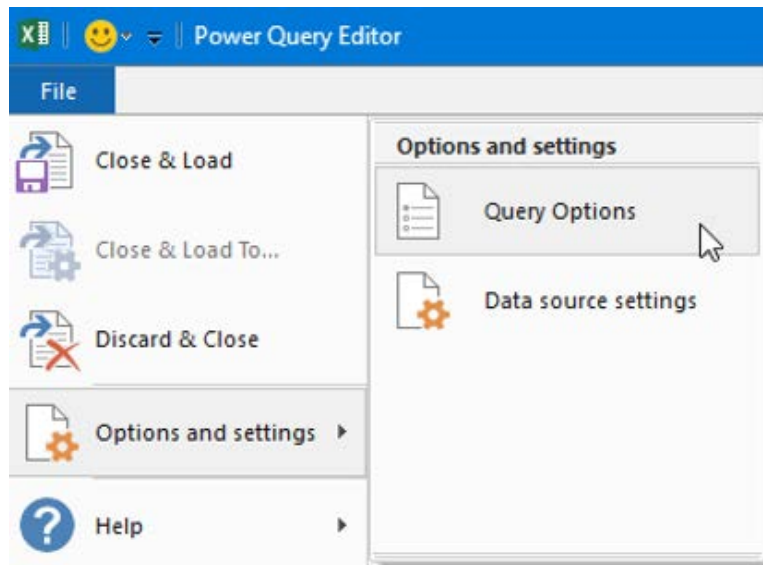


Figure 3.45 – The Power Query Options and settings options

3. Choose **Get Data**, and then from the drop-down list, select **Query Options**:

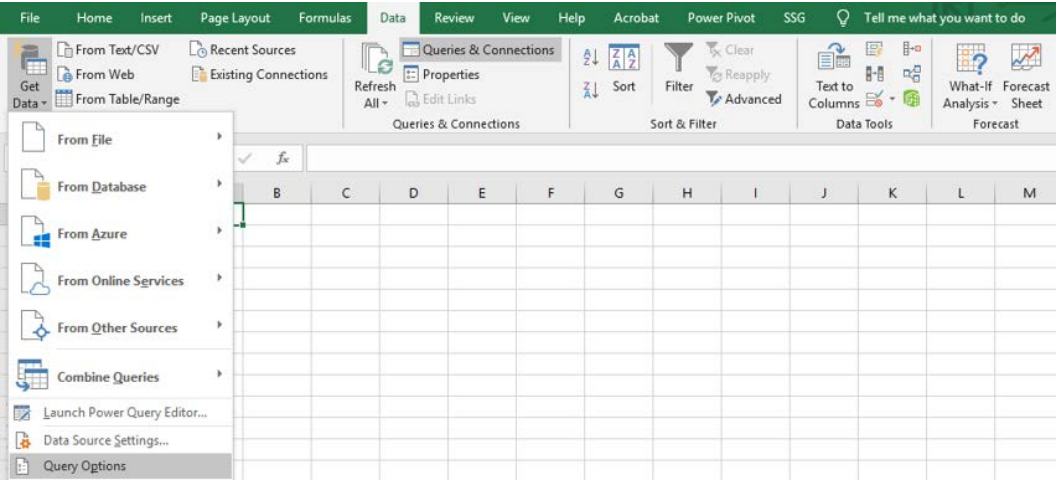


Figure 3.46 – The Power Query options and settings

4. Under the main **GLOBAL** heading, select **Data Load**. You can also set default options for the current workbook only—these options are listed under the appropriate heading at the bottom left of the dialog box.
5. Notice that the **Use standard load settings** option is on by default, which means that the query will load to the worksheet only:

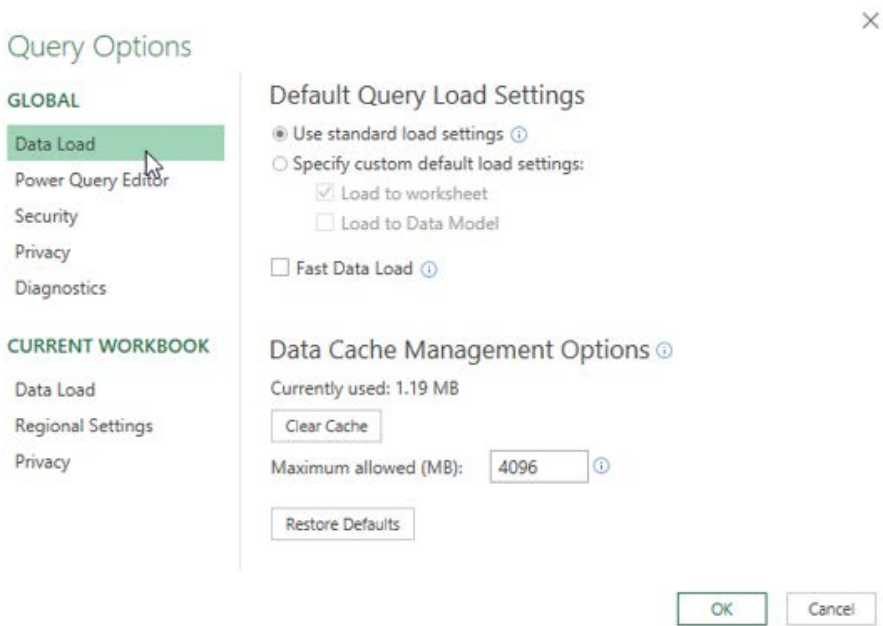


Figure 3.47 – The default query options

6. To set the default options for every query so that queries are loaded to the worksheet and the data model automatically, ensure that you select both of the options under **Specify custom default load settings**.

Note

Do not load unnecessarily to the worksheet; for instance, if you have created 25 queries and choose **Load to the worksheet by default**, all 25 queries will load to different sheets within the workbook. This is resource-intensive and normally, you would not need all of those queries in the workbook.

7. We can load data faster by clicking on the **Fast Data Load** option, located under the checkbox for **Load to Data Model**. Setting this option will load queries much quicker but could have an effect on how Excel 2019 performs while the queries are loading.
8. **Background Data Load** for the current workbook is another important option to consider. Setting this option will allow all queries in the workbook to cache in the background. Of course, you will benefit from quicker movement between steps in a query or moving from query to query, but it must be said that with this setting in Power Query will place a strain on the CPU power and RAM.
9. To set this option within Power Query, select **File | Query Options**, then make sure you select **Data Load** under the **CURRENT WORKBOOK** heading. Then, check the checkbox next to **Allow data preview to download in the background**, then click on the **OK** button to save the change:

Query Options

GLOBAL

Data Load
Power Query Editor
Security
Privacy
Diagnostics

CURRENT WORKBOOK

Data Load
Regional Settings
Privacy

Type Detection

☒ Automatically detect column types and headers for unstructured sources

Relationships

☒ Create relationships between tables when adding to the Data Model for the first time ⓘ
☐ Update relationships when refreshing queries loaded to the Data Model ⓘ

Background Data

☒ Allow data preview to download in the background

Figure 3.48 – The Data Load settings in the current workbook

Another setting that can help tremendously when working with connected sources is to clear the cache. If you receive connection errors when working with connected online sources, the first troubleshooting advice would be to clear the cache in Power Query.

10. To clear the cache, select **File | Query Options**, then make sure you select **Data Load** under the **GLOBAL** heading. Then, click on **Clear Cache** under **Data Cache Management Options**, then click on the **OK** button to save the change:

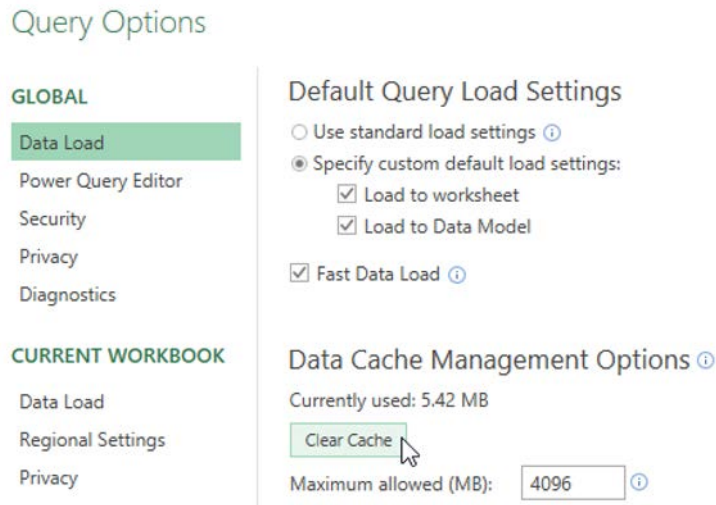


Figure 3.49 – Clearing the cache in Power Query

Now that we have changed the default settings, let's see how to load queries manually.

Loading queries to the worksheet manually

To load queries manually to the worksheet, take the following steps:

1. Open the workbook named `SSGLoadDataM.xlsx`.
2. Click on the data in the worksheet and then select the **Data** tab.
3. Select the **From Table/Range** icon located under the **Get & Transform** group. Power Query will load
4. Make some transformations to the data, then you will be ready to manually send the transformed data back to Excel.

5. Click on the **Home** tab, then select **Close & Load To...**:

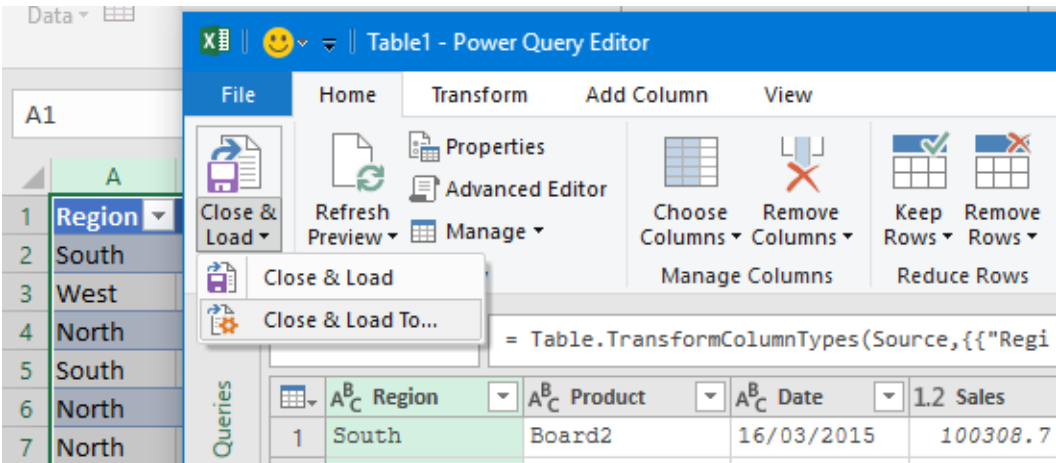


Figure 3.50 – The Close & Load To... option

6. You will be taken back to the Excel interface and the **Import Data** dialog box, containing a range of options, will appear over the data table on the worksheet:

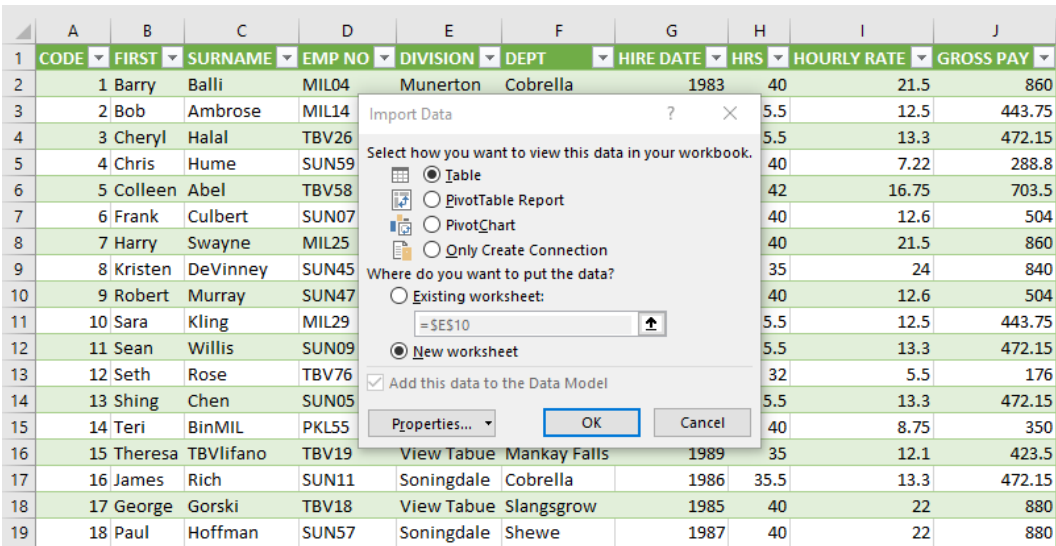


Figure 3.51 – The Import Data option

7. The **Import Data** options relate to how you want to view the data and where you want the data to be placed on the worksheet. The first option is already selected, which allows you to view the data in a table on a worksheet. To decide where to place the data, select **Existing worksheet:**, for this example.
8. Choose a worksheet destination by clicking on the arrow icon, which will take you to the worksheet to select a cell.
9. The data is placed into the cell immediately and the **Queries & Connections** pane will open to the right of the worksheet. Also, note that there is a **Query Tools** contextual menu on the title bar:

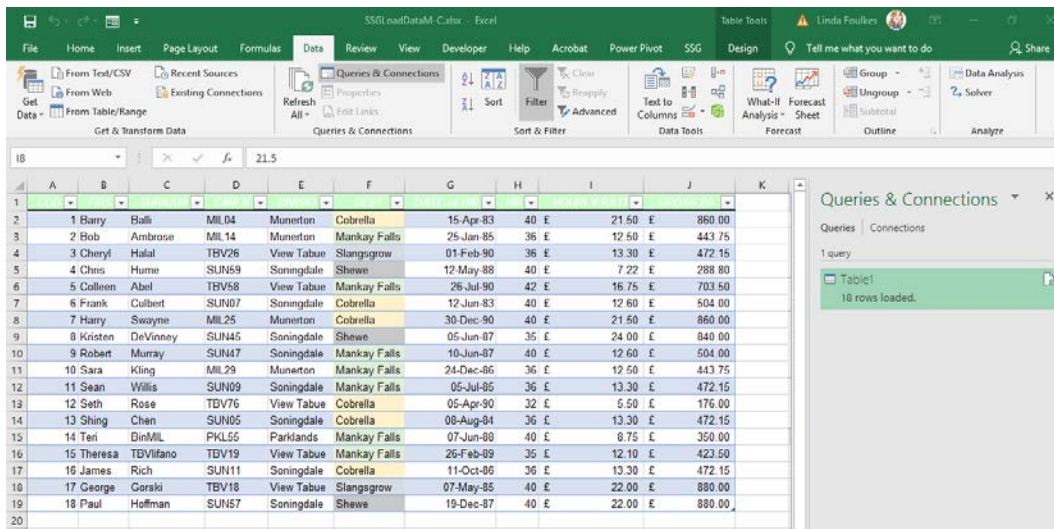


Figure 3.52 – Data with the Queries & Connections pane and Query Tools

The Query Tools contextual menu is only visible when you click on the imported table; once you click off the table onto the worksheet, it will disappear.

10. Once you are in the worksheet, you will also have access to the **Load To...** options as often, you will need to unload a query or choose another load option.
11. To unload the existing query from the worksheet, visit the **Queries & Connections** pane to locate the loaded table; in this case, my table is named **Table1** by default.

12. Right-click on **Table1** and choose the **Load To...** option from the shortcut menu provided:

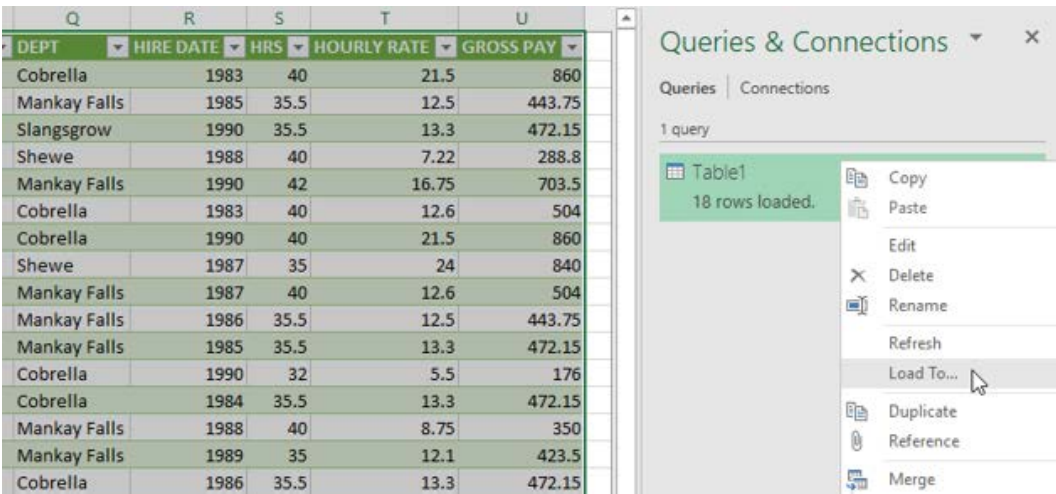


Figure 3.53 – The Load To... options

13. The **Import Data** dialog box will present itself once again.
14. Choose the **Only Create Connection** option. Notice that the **Existing worksheet:** and **New worksheet** options are now grayed out and no longer available:

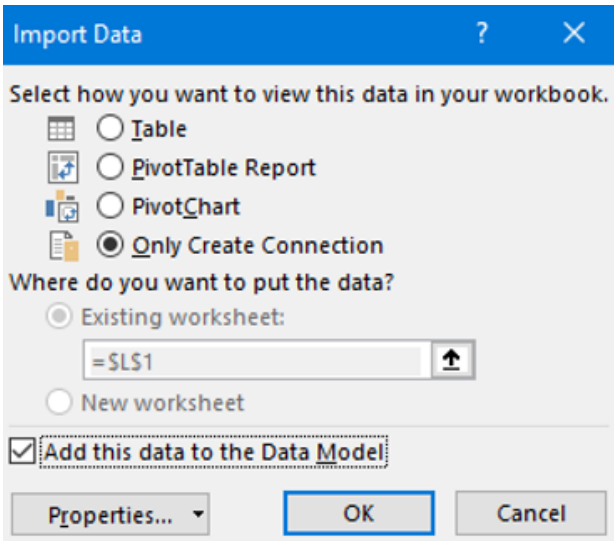


Figure 3.54 – The Import Data dialog box

15. Click on the **OK** button to save the changes.

An information dialog box will appear indicating that the table will be removed from the worksheet:

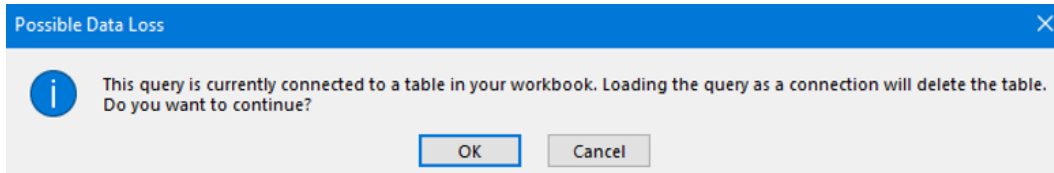


Figure 3.55 – The Possible Data Loss dialog box

16. Click on **OK** to confirm.

The data is removed from the worksheet.

Let's continue by looking at the other options available in the **Import Data** dialog box:

1. This time, we will go to **Load To...** and choose **PivotTable Report**.
2. Right click on the **Table1** query connection.
3. Select **PivotTable Report** from the **Import Data** dialog box.
4. Choose a location to place the PivotTable report—we will place it in cell L1.
5. Click on the **OK** button to save the changes.
6. The blank PivotTable report is placed in the worksheet, along with the PivotTable **Fields** pane.
7. Close the **Queries & Connections** pane as you do not need it right now. You can always open it up again by clicking on the **Queries & Connections** icon located on the **Data** tab.

8. Select fields to add automatically to the PivotTable report to customize it to suite your requirements. Note that when you click on the PivotTable report, the **PivotTable Tools** contextual menu is visible, offering a huge range of customization options:

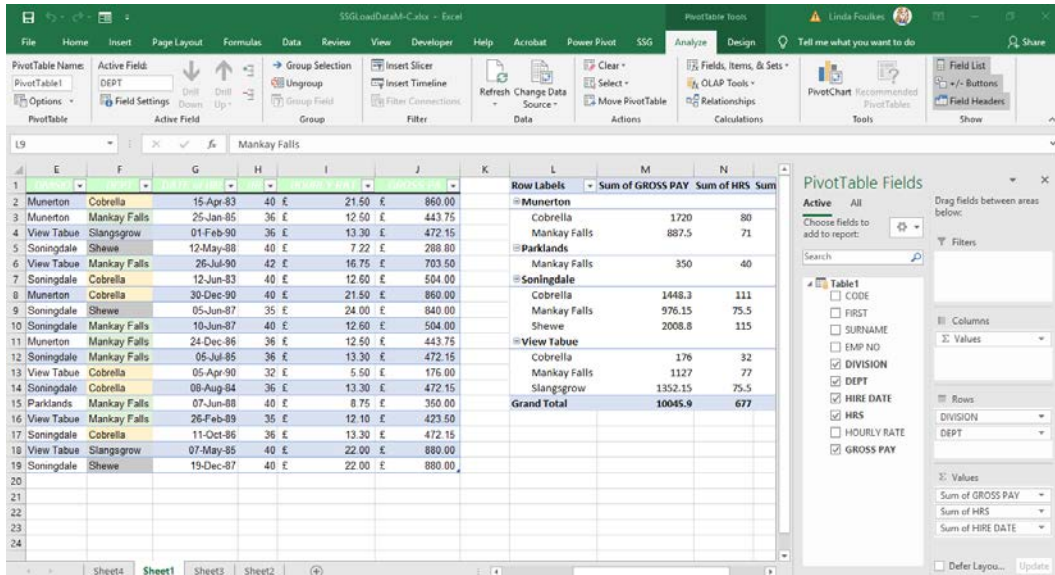


Figure 3.56 – PivotTable report customization

9. If, at any time, you find the **Close & Load To...** option grayed out in Power Query, simply use the **Close & Load** option, then access the **Queries & Connections** pane, where you will be able to right-click on your query connection and access the **Load To...** options to change the method of importing data.

Note

If you are loading data into a data model only, the file sizes are considerably smaller.

Data profiling tips

We will learn about the function of the data profiling icons in the **View** tab in Power Query through Excel and Power BI. We will look at how to display data using monospace fonts, as well as how to add **Column Quality**, **Distribution**, and the **Profile** feature in Power BI and understand what they do. The full set of data profiling options are currently only available through the Power BI desktop. So, why would we use these features? Simply put, we do so to make working in Power Query more visual with a few quick access tools to replace or edit data.

We will be using the `SSGProfiling.xlsx` workbook for all of the following data profiling examples.

Although data profiling is available in Power Query through Excel, it is not as comprehensive as accessing it through Power BI. We will, therefore, explain all of these options using the Power BI application. For reference, we have included the following screenshot of the **View** tab of Power Query through Power BI and Excel to show you how the options differ.

The following screenshot displays the Power Query **View** tab elements through Excel 2019. I want to draw your attention to the **Data Preview** group, where you will notice that the data profiling elements are minimal compared to those in *figure 3.57*:

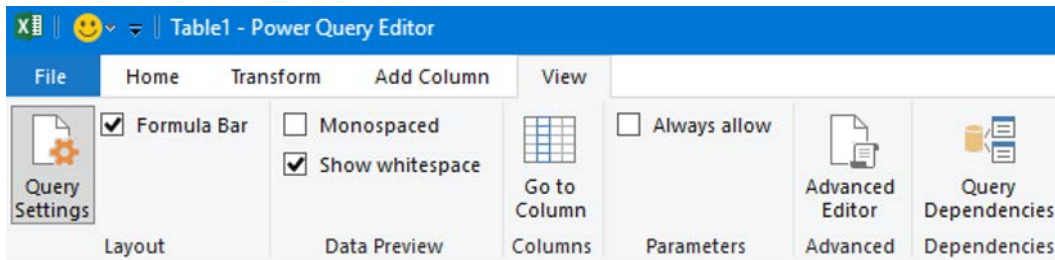


Figure 3.57 – The Power Query data profiling options through Excel 2019

You will see in the following screenshot that the **Data Preview** group through Power BI is much more comprehensive and contains the most important data profiling elements:

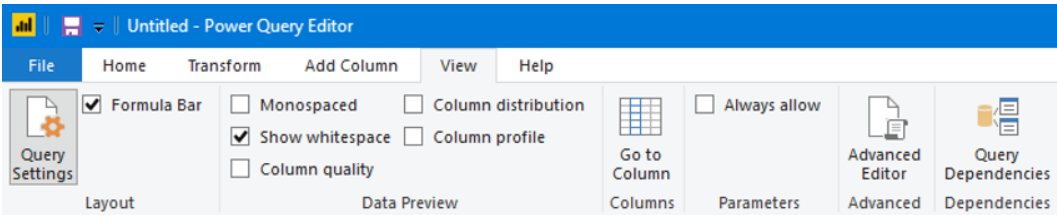


Figure 3.58 – The Power Query data profiling options through Power BI

We will now investigate the different data profiling options using Power Query through Power BI. Make sure you have opened Power BI to follow these examples.

The data profiling options are **Column Distribution**, **Column Quality**, and **Column Profile**. Let's enable all of these features and then look at them individually in Power Query:

1. You should already have the `SSGProfiling.xlsx` workbook open and have created a new query from the table data.
2. In Power Query, click on the **View** tab.
3. From the **Data Preview** group, click on the **Column Quality**, **Column Distribution**, and **Column Profile** checkboxes to activate the features.

You will instantly see some visual changes in the Power Query interface:

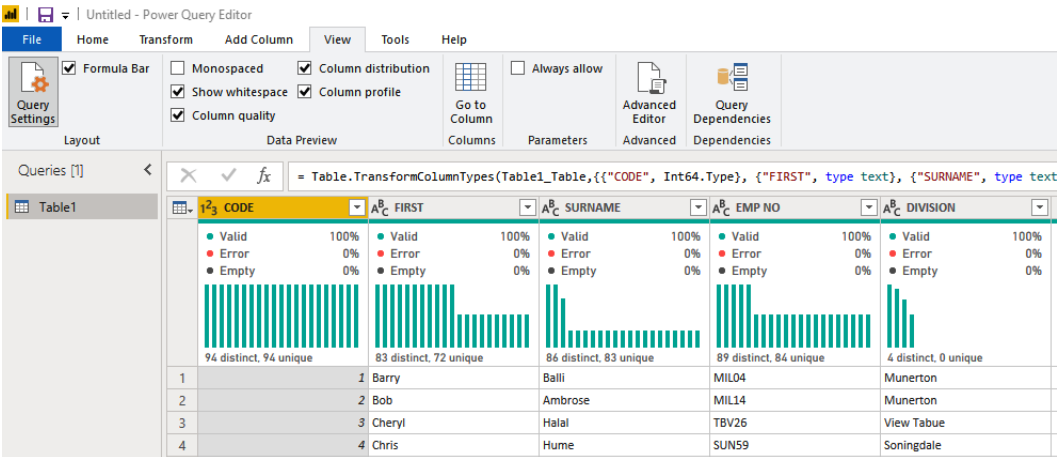


Figure 3.59 – The data profiling options activated

Now, we will learn about each of these features and how they can benefit us. If you are new to these features, it is important to know how they are dispersed in the **Data Preview** pane. The following is a selection of screenshots showing the different areas of application in the preview window:

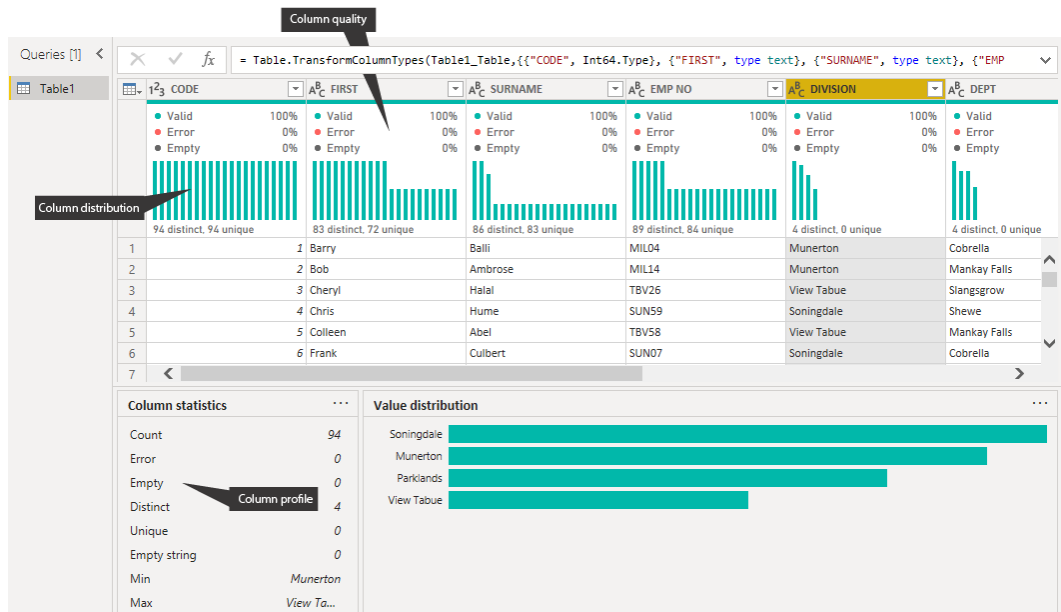


Figure 3.60 – The data profiling options

In the next section, we will see about the columns' profiling, quality, and distribution.

Column profile

To view the column profile, click on a column. The interface will now change to visually display the dataset so that you can browse its structure:

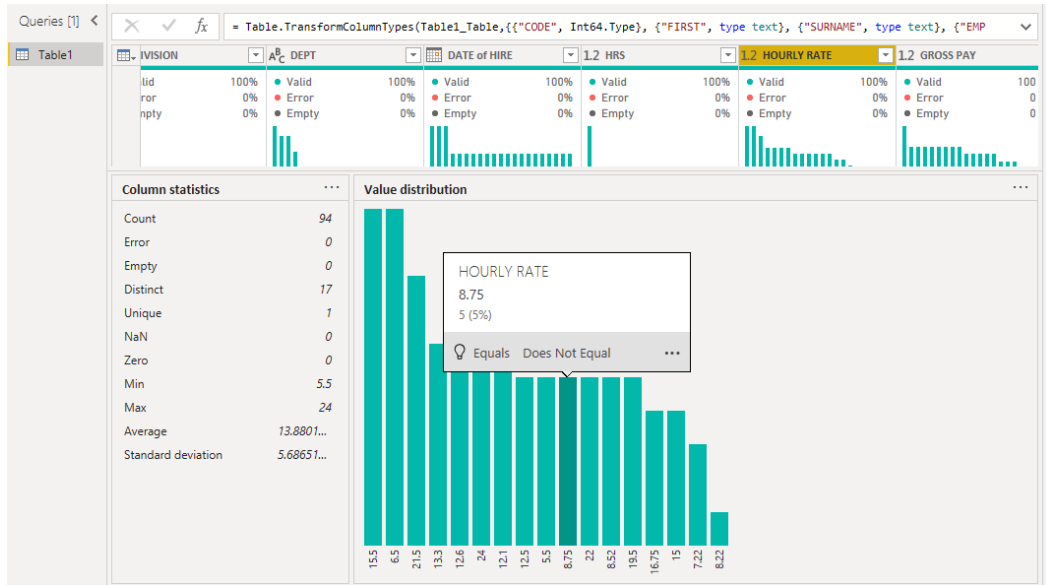


Figure 3.61 – Clicking on a column to see the column profile

Power Query displays column profiling in the top 1,000 rows by default. Edit this setting by left-clicking on the **Status Bar** icon, and then changing the option to **Column profiling based on entire data set**:

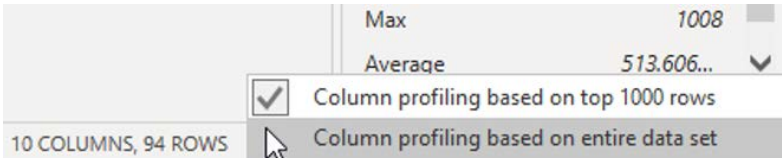


Figure 3.62 – Changing the column profiling settings

The column profile displays column statistics for informational purposes only to the left of value distributions as it does not allow any mouse interaction to further transform the data.

Column quality

Looking at the **CODE** column in the dataset, I can visually see from the column quality representation above the column that there are errors in the **CODE** column, which is displaying 3%. Directly under the **CODE** column header, you will notice a tiny graphical representation of the column health:

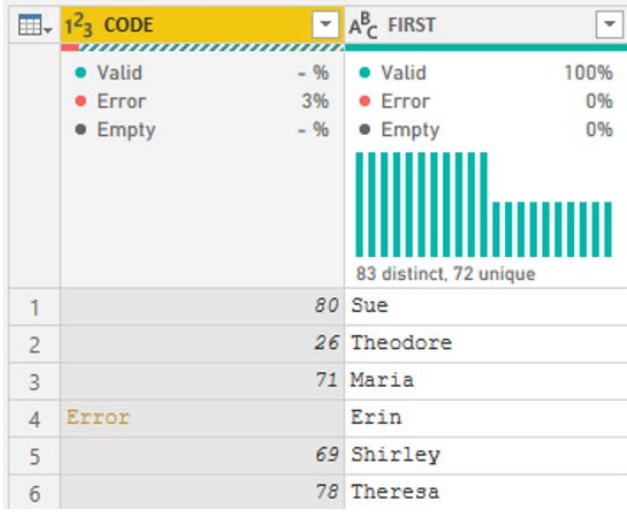


Figure 3.63 – Errors displayed in the CODE column and a graphical representation

If you hover with the mouse pointer over the column quality area of the preview pane, you will notice that a pop-up box appears, offering you the chance to remove the errors in the query. Click on **Replace Errors** to fix the problem in the query, and then replace the values:

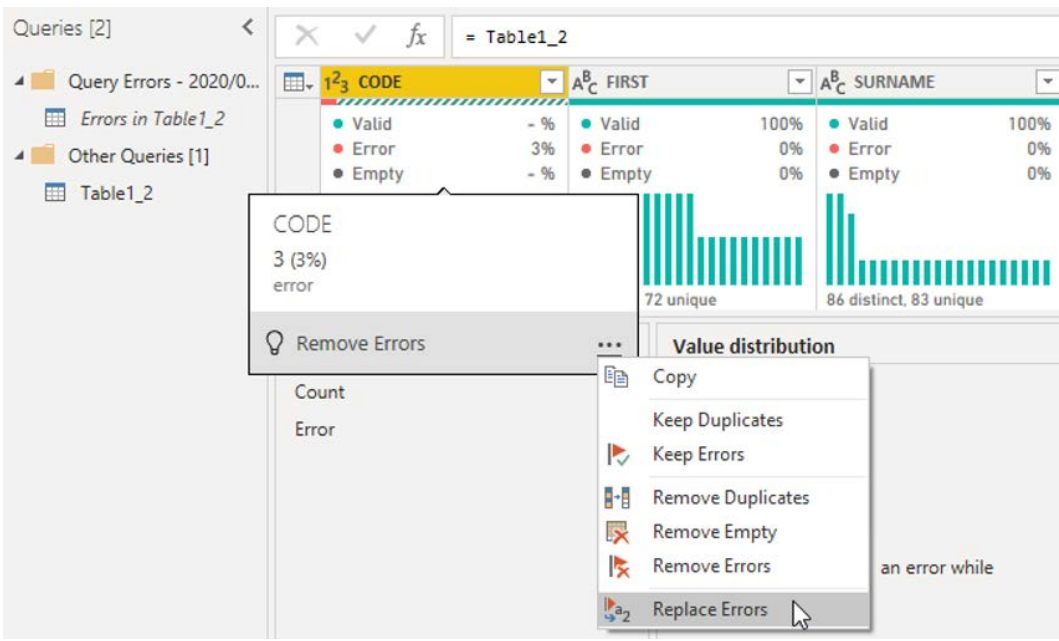


Figure 3.64 – Replacing errors in the CODE column

This is a really easy method to fix errors in your query!

Column distribution

Column distribution is a great visual tool but does not really add any other value at this stage of development. If we right-click on the column distribution visualization, we will see the same options that were offered for column quality:

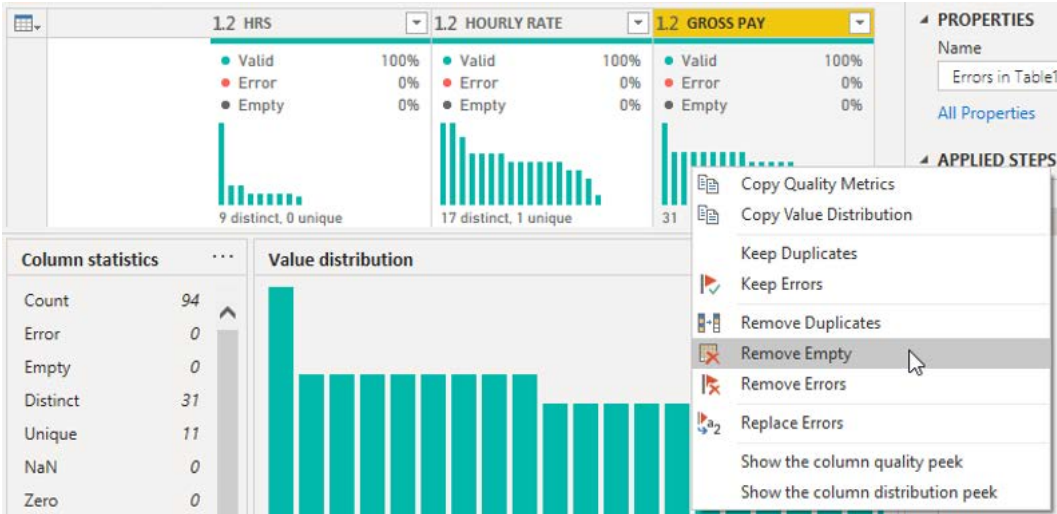


Figure 3.65 – A column distribution example

Column profile is definitely the better tool to use as it is an enhancement of the column distribution feature.

Summary

In this chapter, you got your first detailed look at Power Query. You also learned all about the query settings that you should not ignore when working with Power Query. So, you should now be able to customize the environment to suit your requirements.

You should also now possess the knowledge to complete the required steps to create a basic query and import the queries back to Excel or Power BI using all the **Load To...** options in detail.

We also discussed data profiling tips and learned how these options can enhance the visual interpretation of data within your query.

In the next chapter, we will learn how to connect to numerous data sources, such as a web page, between Access databases and Excel workbooks, import text and comma-separated value files, and discuss multiple connections. We will end the chapter by investigating the data source settings.

4

Connecting to Various Data Sources Using Get & Transform

In this chapter, you will learn how to connect to numerous data sources using the **Get & Transform** tool, known as Power Query, and investigate data source settings. There are a multitude of different ways in which you can connect various data and this chapter will deal with the data sources you can connect to, from a basic Excel file to multiple SQL servers, using both Excel and Power BI.

Throughout this chapter, we will learn about the connection procedures and the optimum data source to use.

In this chapter, we're going to cover the following topics:

- Connecting from a table or range
- Connecting from the web
- Connecting from a relational database
- Understanding custom connections
- Exploring data source settings

Technical requirements

You will need an internet connection to download the relevant files from GitHub.

The code files for the chapter can be found at the following link:

<https://github.com/PacktPublishing/Learn-Power-Query>

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=pyszzEJlpB4&list=PLcLcvrwLe186O_GJEZs47WazXwZjwTN83&index=5&t=24s.

You should be able to navigate the Power Query interface with ease since you toured the interface in the previous chapter. In addition, we're assuming that you have the knowledge and skills to create a basic query and import the queries back into Excel or Power BI using the **Load To...** option.

A brief introduction to databases

There are different types of databases. For one, there are transactional databases, where all the information is stored in rows. So, for example, if you have a product in a table, then everything about that one product will be in one row. Transactional databases are normally built for **Create, Retrieve, Update, Delete (CRUD)** row operations. In these types of databases, there is typically a primary key so that if a product category name needs to be changed at any point, you can change one record that will automatically update all the products related to this. This sometimes happens in data warehousing, where one product is replaced with another product as the initial product is no longer available or has been discontinued. These databases customarily have systems and operators that do data entry and modifications. One of the biggest problems with this type of database is that there are numerous different relationships between the tables, so when you want to perform a query (this can pull up to 25 different tables), this can be slow and not very efficient. Refer to the *Connecting from a table or range* section for more information about tables.

If you look back at the data model that we built in *Chapter 2, Power Pivot Basics, Inadequacies, and Data Management*, you will remember that we had the fact table in the middle (fRegionSales) and then we had the two dimension tables on either side (dProduct and dSalesRep). The following screenshot will help you remember:

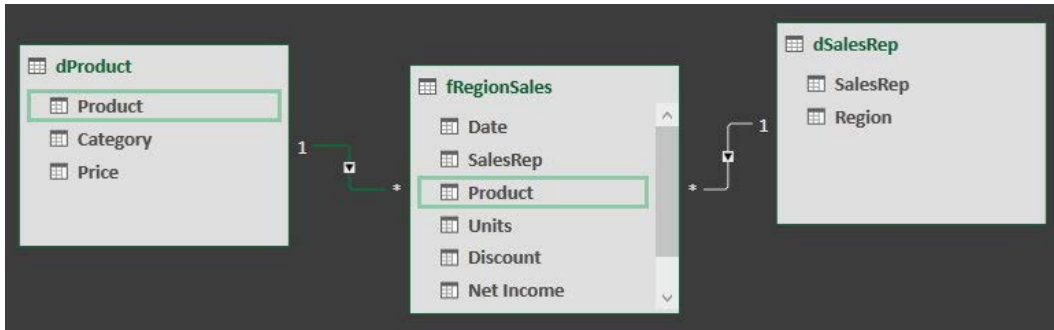


Figure 4.1 – Relational database

From the preceding screenshot, we can observe the following:

- Although we only had a few things in this exercise, always try and keep the data model simple, which will then naturally make it faster. In the middle was our sales information, which is our fact table, and we created one relationship that connected our fact table to the tables around it, which are called *dimension tables*. This is one of the best models to work with in a Power BI system and is called a **star schema**:

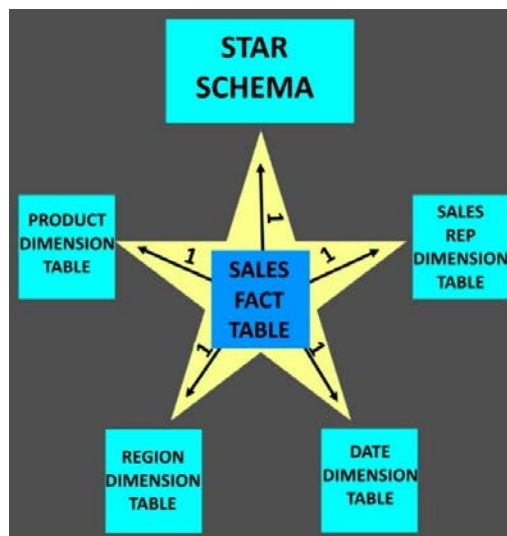


Figure 4.2 – Star schema

- The fact table will generally hold numeric and additive data. In our previous example, our fact table contained how many units were sold, the discount, and the sales amount. These values are generally numeric and can be manipulated in some form. The dimension tables generally hold descriptive information. In our example, a dimension table contained the product, product category, and price, while in the other dimension table, we had the sales representative and the region in which they worked. In bigger databases, each of the dimension tables can also have sub-dimensional tables.
- Up to this point, we have been using natural keys, which are quick and easy to use for our examples regarding relationships between the tables. Although this has a faster implementation, it does have to use real data that already exists in the tables. A problem with this could be that when we try and merge multiple source systems that have different business rules, we will not have control over the warehouse systems that we would like to implement. Because of these pitfalls, it is recommended that you use an anonymous **Integer Primary Key**, which is also known as a surrogate key. The one great thing about them is that they are not based on business or application data. They are obviously numeric, and they need to be sequential. There is no business intention and the only thing that they do is link the dimension table to the fact table. In the fact table, this surrogate key will be stored as a foreign key and since the key is numeric, the size will be much smaller than that of text, which means that it will be quicker to index. The faster the indexing, the quicker the query's performance. It should be observed that primary or natural keys are used in **online transactional processing (OLTP)**, whereas surrogate keys are used in **online analytical processing (OLAP)** schemas.

There was a time where commercial **database management systems (DBMS)** all had the same look and feel from an architectural point of view, but this is changing due to the fact that so many more people are creating their own bespoke databases. There is no wrong or right way of creating a database. Personally, I think there are things that will assist in creating databases with better performance.

Connecting from a table or range

In Excel, the **Get & transform** tool is based on queries. With regard to a database, queries are used to retrieve data from a database. The reason why queries are so powerful is because we might only need a little bit of information from a huge dataset and the query will only give us the information that is required. The following screenshot shows a range of products that have been typed into an Excel worksheet. You will notice that, in column **B**, there are a number of cells that contain null values that we will have to fill in:

	A	B	C
1		Category	Product
2		Action	Dyno5
3			Board4
4			Game7
5		Construction	Dyno1
6			Game1
7			Game6
8		Creative	Dyno2
9			Board5
10			Game2
11			Game8
12			Game9
13		Education	Dyno3
14			Board1
15			Board3
16			Game3
17			Game4
18			Game10
19		Electronic	Dyno4
20			Board2
21			Game5

Figure 4.3 – A range in Excel

One of the ways in which we can do this is with the **Get & Transform** tool, which can be found in the **Data** ribbon. For us to use this in our query, we will have to convert this into a table. We can either select the range and use the *Ctrl + T* shortcut keys, or we can click **Table** in the **Insert** ribbon. We do not need to convert this into a table as **Get & Transform** will use either a range or a table. Let's see how this works:

1. Click anywhere inside the table and then click on the **From Table/Range** icon in the **Data** ribbon, as shown in the following screenshot:

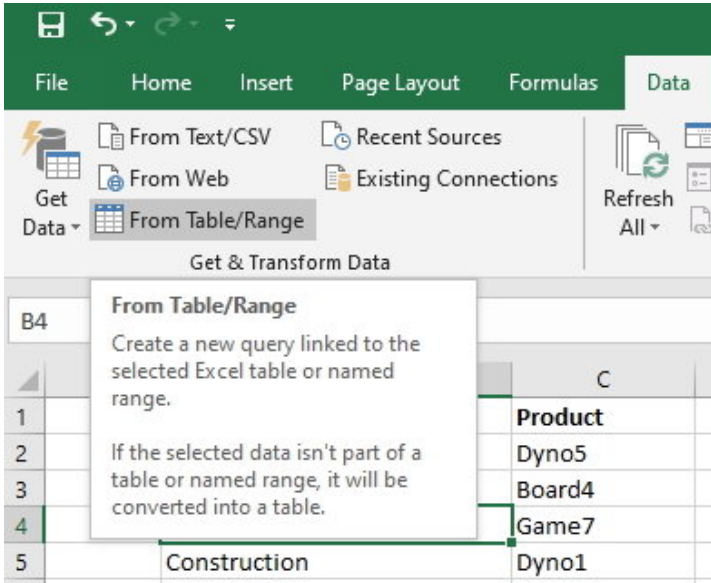


Figure 4.4 – From Table/Range icon

2. Click on the **Table/Range** icon. Excel automatically selects a range for you to convert into a table. If you have already converted your range, this step can be skipped. When we convert the range into a table, Excel will normally select the entire range, but if there are gaps in your range with blank cells or rows, it might not pick up the entire range, so you will need to check that it has selected everything you need. The following screenshot shows that the entire range has been selected:

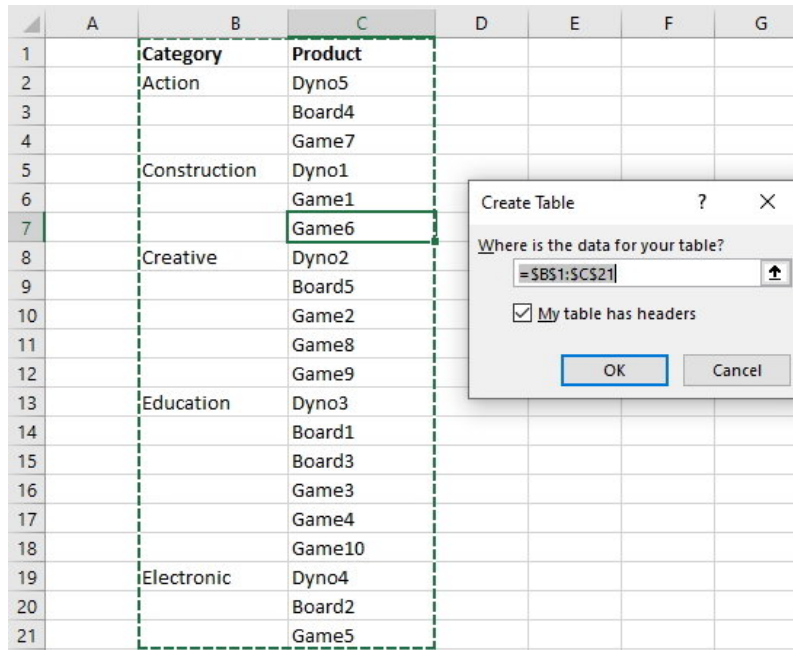


Figure 4.5 – Creating a table

3. Once you've selected **OK**, the **Power Query Editor** will launch and you will be able to see your data.

4. Right-click on Category | **Fill** | **Down**. This will fill all the data down and get rid of all the null values:

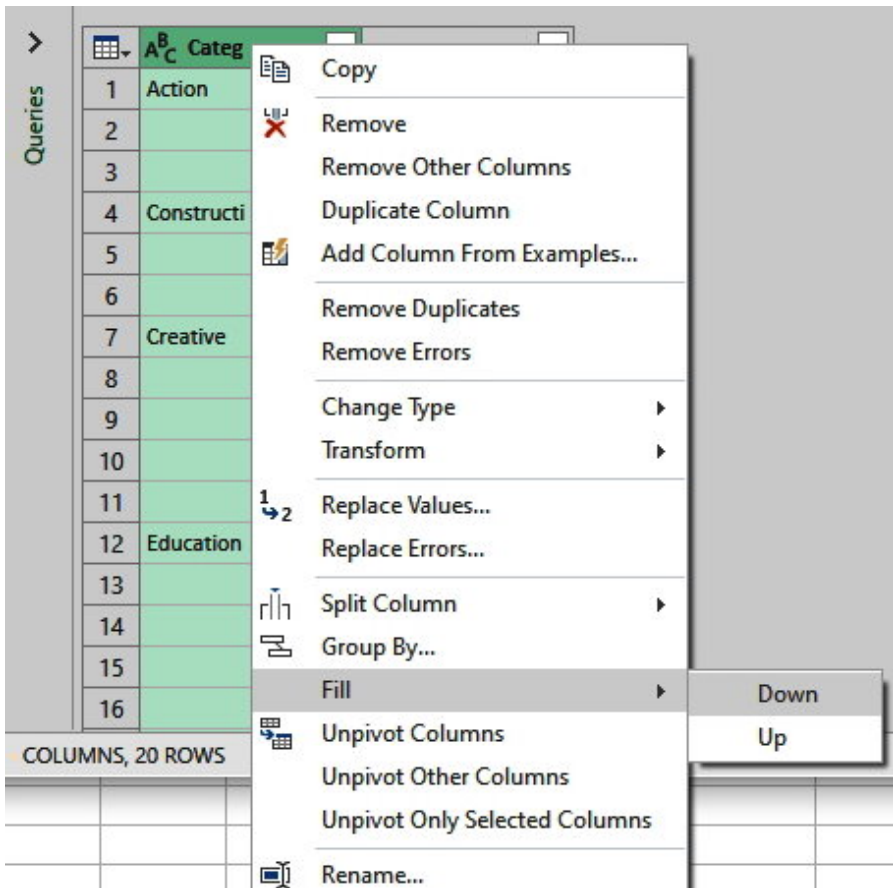


Figure 4.6 – Filling data down

Once complete, we no longer have any null values and we will now be able to use this data:

	A	B	C
1	Category ▼	Product ▼	
2	Action	Dyno5	
3	Action	Board4	
4	Action	Game7	
5	Construction	Dyno1	
6	Construction	Game1	
7	Construction	Game6	
8	Creative	Dyno2	
9	Creative	Board5	
10	Creative	Game2	
11	Creative	Game8	
12	Creative	Game9	
13	Education	Dyno3	
14	Education	Board1	
15	Education	Board3	
16	Education	Game3	
17	Education	Game4	
18	Education	Game10	
19	Electronic	Dyno4	
20	Electronic	Board2	
21	Electronic	Game5	

Figure 4.7 – Filling data down complete

Connecting from a table or range is probably one of the quickest and easiest ways of connecting data since the data is currently already sitting in the Excel document. This means there is no need for us to import from a different data source.

Connecting data to the web

It's great that you can get data from the web. Not only can you get the data, but you can also clean it, getting rid of any unwanted data and formatting it in the correct format before loading it into Excel. By doing this, the next time you want to use this data, you can simply refresh it and automatically get all the new updates.

Note

There is one limitation while getting data from the web: the data must be formatted as HTML tables and not as JavaScript.

We are going to look at two different ways to connect to the web, as follows:

1. The first way will show you how to connect multiple different tables from the same site, although there are not many rows in each table.
2. The other way is to connect to a CSV file that contains just over a hundred thousand rows of data.

Being a typical South African, I am really into sports, especially cricket. Let's have a look at Sky sport's cricket tables, which can be found at <https://www.skysports.com/cricket/tables>:

Cricket Tables

NewsPunditsWomen'sVideoFixturesResultsTablesTeams+CompetitionsCricket on SkyScore CentreBet

ICC CRICKET WORLD CUP 2019 GROUP						
POS	TEAM	PLD	W	D	L	PTS
1	India	9	7	0	1	15
2	Australia	9	7	0	2	14
3	England	9	6	0	3	12
4	New Zealand	9	5	0	3	11
5	Pakistan	9	5	0	3	11

[See Full Table](#)Last Updated: 26/09/19 3:51am[See Full Table](#)

COUNTY CHAMPIONSHIP DIVISION 1 LEAGUE ...						
POS	TEAM	PLD	W	D	L	PTS
1	Essex	14	9	4	1	228
2	Somerset	14	9	2	3	217
3	Hampshire	14	5	6	3	176
4	Kent	14	5	4	5	172
5	Yorkshire	14	5	5	4	165

Last Updated: 26/09/19 5:22pm

COUNTY CHAMPIONSHIP DIVISION 2 LEAGUE ...						
POS	TEAM	PLD	W	D	L	PTS
1	Lancashire	14	8	6	0	233
2	Northamptonshire	14	5	7	2	188
3	Gloucestershire	14	5	6	3	182

ROYAL LONDON ONE-DAY CUP 2019 NORTH G...						
POS	TEAM	PLD	W	D	L	PTS
1	Notts Outlaws	8	6	0	1	13
2	Worcestershire Rapids	8	6	0	2	12
3	Lancashire Lightning	8	5	0	3	10

Figure 4.8 – Sky sport's cricket tables

Note

These tables are current at the time of writing. The actual scores and tables may look very different when you look at the same website.

3. Getting the data from the web is relatively easy. If you are using Excel 2016 or newer, go to the **Data** tab and then select **From Web** from the ribbon.

If you are using 2013 or earlier, you click on **POWER QUERY** and then select **From Web**:

- Paste the `https://www.skysports.com/cricket/tables` link into the **From Web** dialog box and then click **OK**:

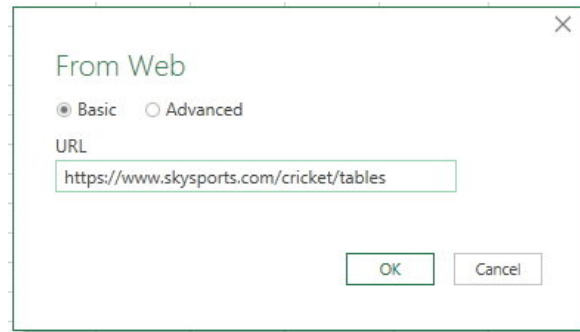


Figure 4.9 – From Web address

- The **Navigator** dialog box will open with a list of all the tables of the World Cups that have been held so far that are available on this website. If we had used the URL `https://www.skysports.com/cricket/tables/3565/icc-cricket-world-cup-2019`, it would have only displayed the **ICC Cricket World Cup** table, but it would have given you a longer list, displaying all the teams and not just the top few. Clicking on the **ICC Cricket World Cup 2019** table will give you a preview of the table:

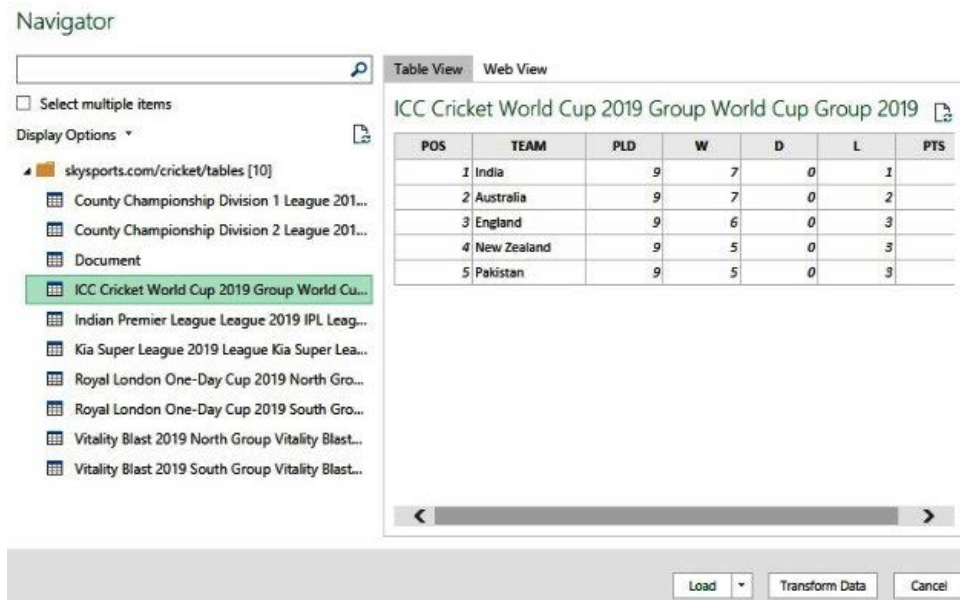


Figure 4.10 – ICC Cricket World Cup table

Note

The table called **Document** does not contain any tables; it only holds the specific page's HTML code. If you click on any of the other tables, they all contain data in tables.

- When you click on the **Web View** tab at the top of the preview, it will show you the website that it is linked to:

The screenshot shows the 'Web View' tab selected in the Power BI interface. On the left, the 'Navigator' pane lists the available tables: 'Document' and 'ICC Cricket World Cup 2019 Group World Cu...'. The main preview area displays the 'sky sports' website, specifically the 'Cricket' section, which shows the 'WORLD CUP GROUP 2019' table. The table lists the top 6 teams in the group: India, Australia, England, New Zealand, Pakistan, and Sri Lanka, along with their points (PLD and PTS).

POS	TEAM	PLD	PTS
1	India	9	15
2	Australia	9	14
3	England	9	12
4	New Zealand	9	11
5	Pakistan	9	11
6	Sri Lanka	9	8

At the bottom of the interface, there are buttons for 'Load', 'Transform Data', and 'Cancel'.

Figure 4.11 – Web View of the data

- If we want to add more than one table in Excel, we can click on **Select multiple items** and then select all the tables that we would like to import.
- Once you have selected the **ICC Cricket World Cup 2019** table, click on **Transform Data**. The reason why we do not select **Load** is because we would like to edit or clean the data before we load it into Excel or the data model.

After completing these steps, **Power Query Editor** will open. Here, there are several things that we can do:

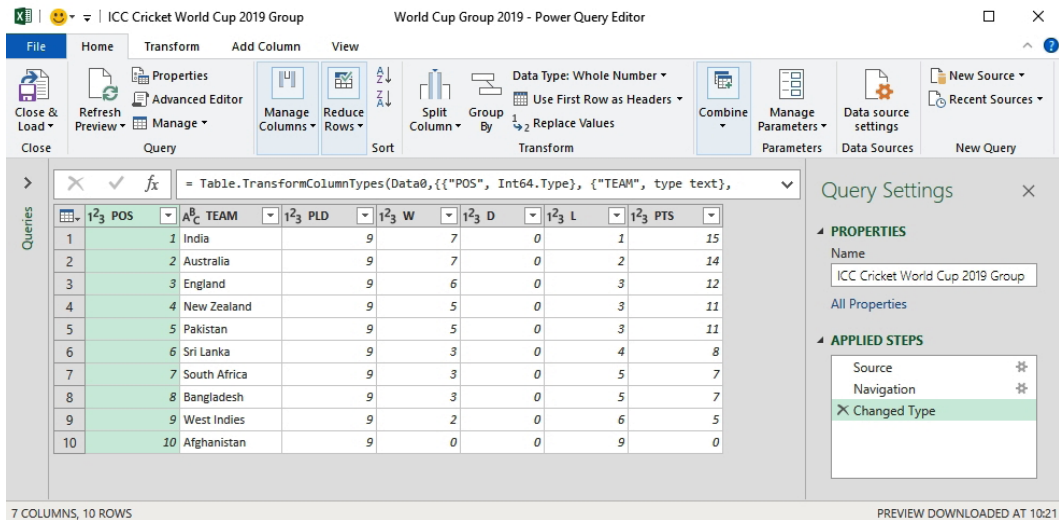


Figure 4.12 – Power Query Editor

9. From the preceding screenshot, we can observe that we can use the Editor to do the following:
 - Manage the query
 - Choose or remove columns and rows in the table
 - Sort data
 - Split columns
 - Group and replace values
 - Combine tables with other data sources
 - Adjust the parameters of the table

10. Once you have edited the information you would like to edit, click on the **Home** tab and select **Close & Load To...**:

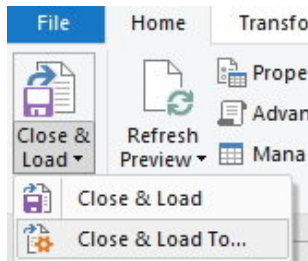


Figure 4.13 – Close & Load To... icon

11. This will open the **Import Data** dialog box, where you have the option to create a **Table**, **PivotTable Report**, **PivotChart**, or **Only Create Connection**. You also have the option of adding this to an existing worksheet or a new worksheet. For now, I am going to create a table in a new worksheet so that we can see the information, as shown here:

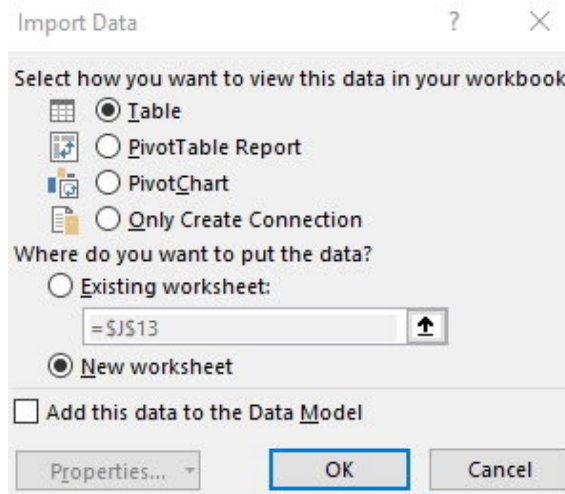


Figure 4.14 – Import Data dialog box

Note

It is important to select **Only Create Connection**. You do not want the same data in your worksheet and in the data model.

12. The appeal here is that when the data changes on the web, you can refresh the data in your sheet and it will automatically refresh the table. This means that if you have to complete the same report every month, you do not have to do anything except refresh the data by selecting the **Refresh All** icon in the **Data** ribbon, as shown here:

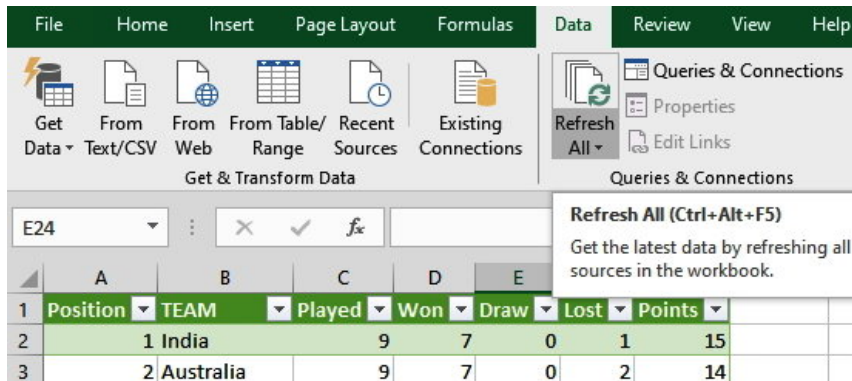


Figure 4.15 – Refresh All icon

13. If you are using more than one data source, you can click on **Queries & Connections**, which will open a window that shows you all the connections from the various data sources to this sheet:

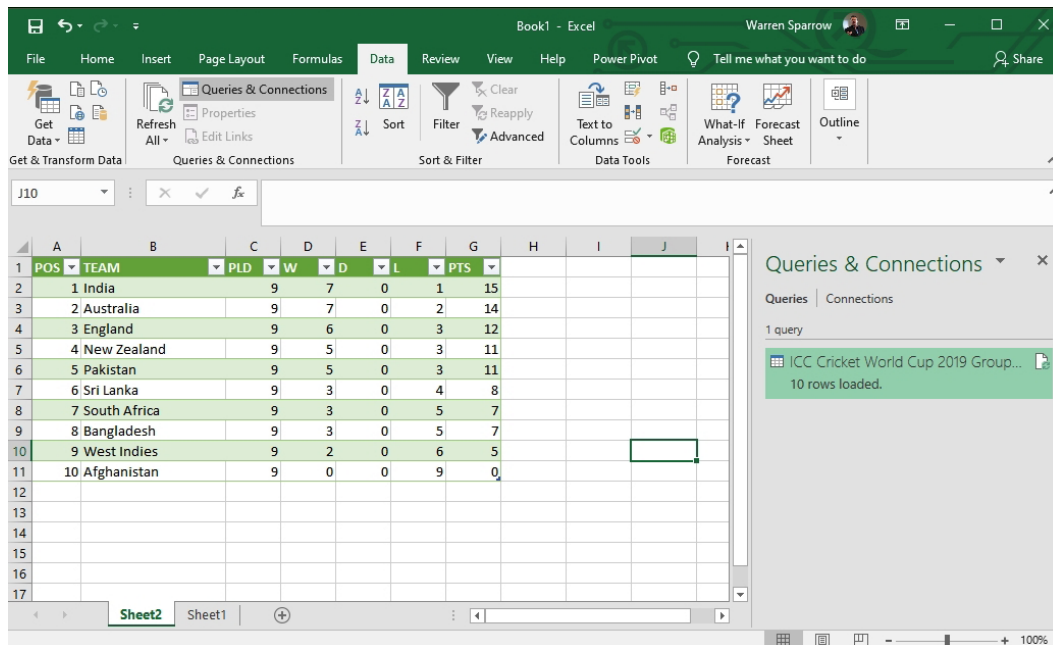


Figure 4.16 – Queries & Connections window

14. If you are using a version of Excel older than 2016, select **Show Pane** from the **POWER QUERY** tab, which will open a **Workbook Queries** window:

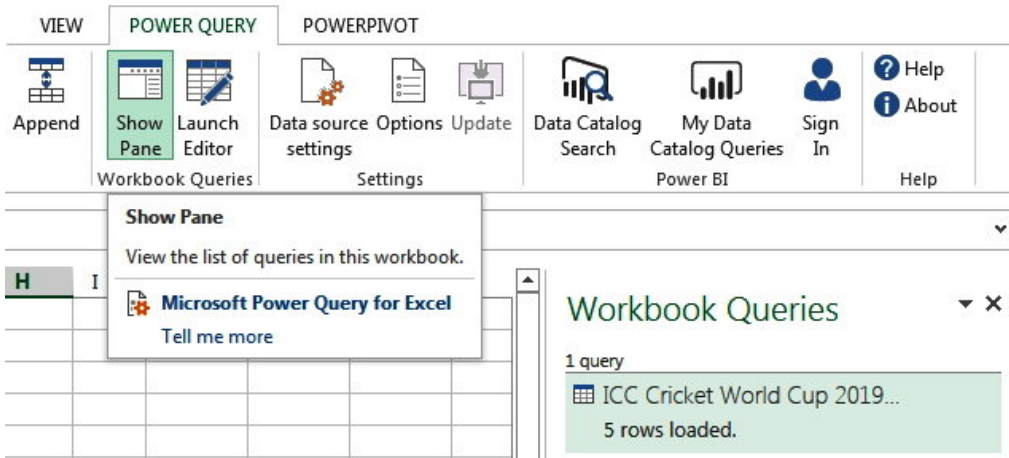


Figure 4.17 – Workbook Queries window

From the workbook query, you are able to see your connections in the same way as you can in Excel 2016 and newer.

There are a few compatibility errors with things that are created in newer versions of Excel and then opened in a version earlier than 2016. You will be asked to either update your version of Power Query or your version of Excel. Normally, if you refresh the data, the error message goes away, but unfortunately, there is no guarantee of this. If this happens, you will only see the data from when it was saved and will not be able to view the refreshed data from the web. If you followed this exercise in a version of Excel earlier than 2016, it will refresh the data from the web without any problems:

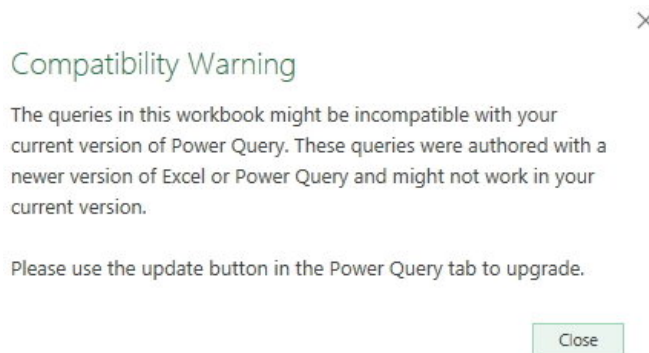


Figure 4.18 – Compatibility Warning regarding versions

To work on these compatibility issues, you can refresh the data from the web in earlier versions of Excel. To do this, you can either click on the **Refresh** button on the right of the connected query or you can right-click and then select **Refresh**:

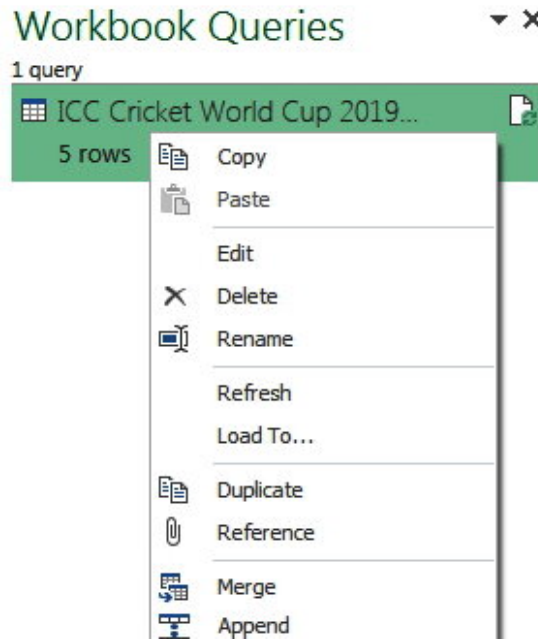


Figure 4.19 – Refreshing the data in earlier versions of Excel

If you open this Excel document in the future and the source data has changed, Excel will notify you of this, telling you that there is newer data and that you have the option of refreshing it. In this way, you will always have the most current information from any website.

Connecting from a relational database

To study an example of connecting from a relational database, I am going to get data from a relational database that uses **Structured Query Language (SQL)**. Although this might seem harder to do than any of the other things we've done thus far, it is very similar, and the hardest thing to do is type your password correctly for the authentication process.

There are different ways to connect to a SQL database; that is, either through Excel's **Get & Transform** tool or through Power BI.

We'll try each of these options in the following sections.

Connecting through Excel's Get & Transform tool

To connect to a SQL database, perform the following steps:

1. Launch Excel.
2. Click on the **Data** ribbon | **Get Data** | **From Database** | **From SQL Server Database**.

At this point, it is pertinent to say that there are huge numbers of databases that you can connect to, as well as to Azure directly, but you can also make connections to online services such as SharePoint, Exchange, and Facebook. This is shown in the following screenshot:

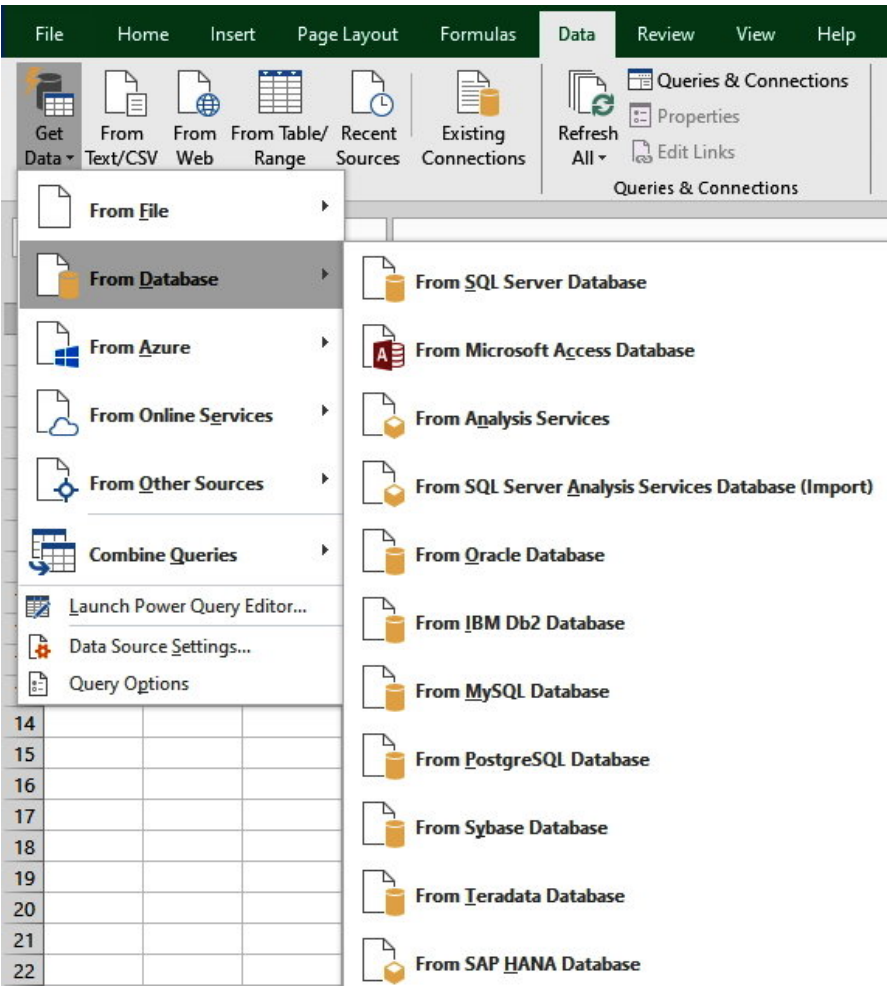


Figure 4.20 – Connecting to a SQL database

The **SQL Server database** connection dialogue box window will open, where you have to type in your credentials to connect to SQL Server:

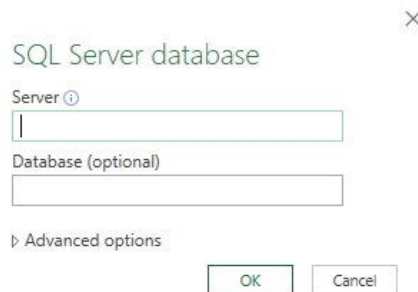


Figure 4.21 – SQL Server authentication

Once connected to the instance, **SQLData** and **Test** are the two databases you will see, and **Orders** and **Customers** will be two of the tables within the database:

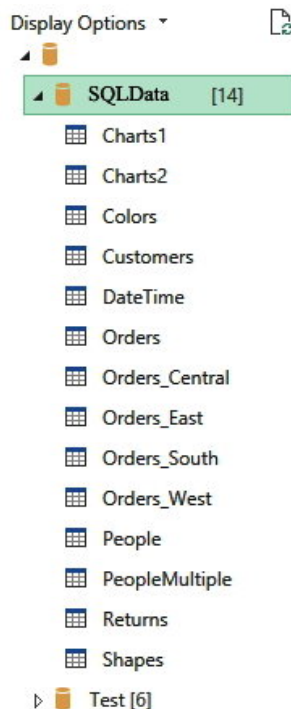


Figure 4.22 – Connected to the SQLData database

Once you've clicked on the relevant table that you require, click **Transform**, which will take you to **Power Query Editor**.

Connecting through Power BI

Although very similar to connecting via the **Get & Transform** tool, Power BI has a few more little tricks that you can implement. Try it out, as follows:

1. In Power BI, click on **Get Data** in the **Home** ribbon and select **More....**
2. From **Get Data | All | SQL Server database**, select **Connect**:

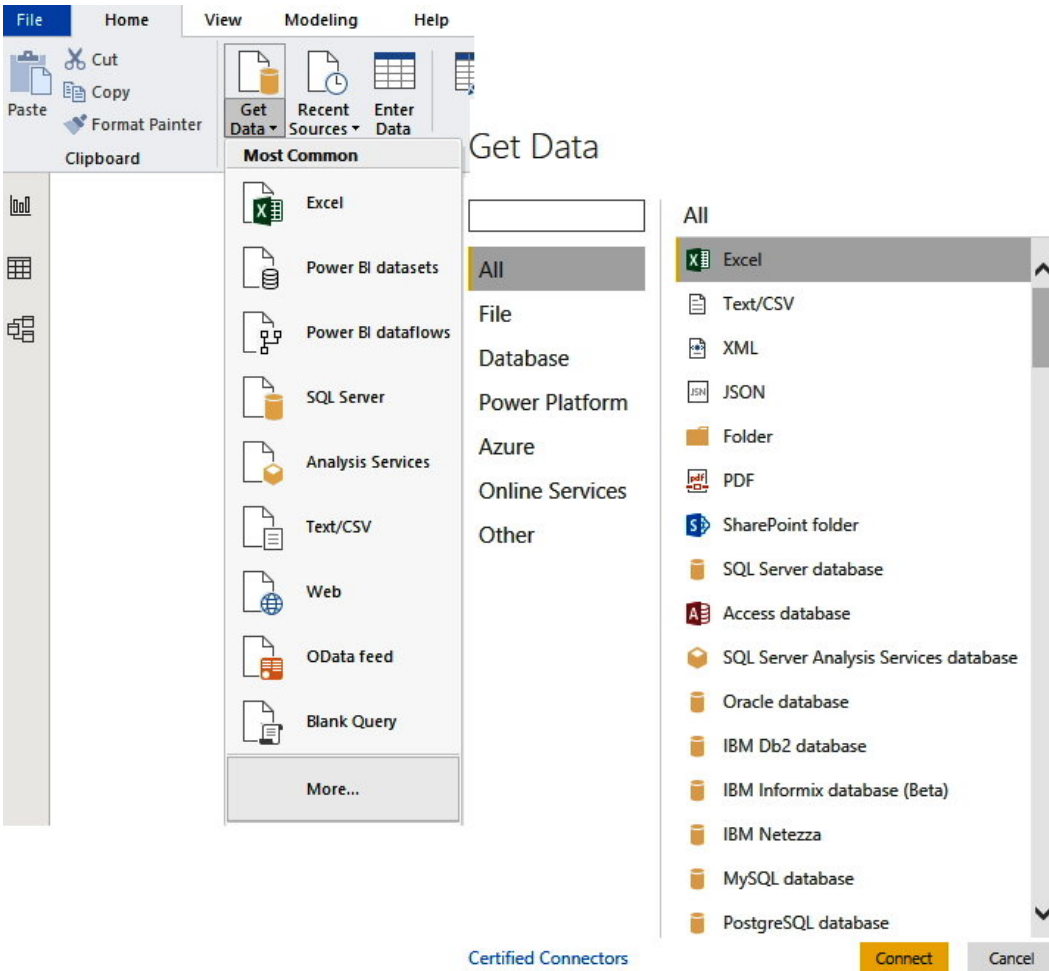


Figure 4.23 – Connecting to the SQL Server database in Power BI

Just like in the previous example, you will have to connect to a database, but Power BI provides an extra option that Excel does not have: you can choose to **Import** or use **DirectQuery**. When you use **DirectQuery**, it does not copy or import data into Power BI, it is always using the most current data. This means that if you are creating or interacting with visualizations, Power BI queries the data source. This means you always have the most up-to-date information, which allows you to build visualizations with very large datasets as you do not need to import the data into Power BI. One of the other advantages is that the 1 GB dataset limit does not apply if you are using **DirectQuery**.

However, this means that apart from having a good internet connection, every time a visual is refreshed, it needs to rerun the query and, depending on the backend, this might take a little while. If it takes between 5 and 30 seconds, then this is acceptable, but anything over a few minutes will time out and the user will receive an error message. Naturally, the number of concurrent users will also affect the performance of the published report, so this will need to be taken into account.

If you are using the **Import** option, then you are saving a copy of the data in your database model and using this to create or interact with your visualizations. Power BI needs to refresh the data for you to see the live or most up-to-date version:

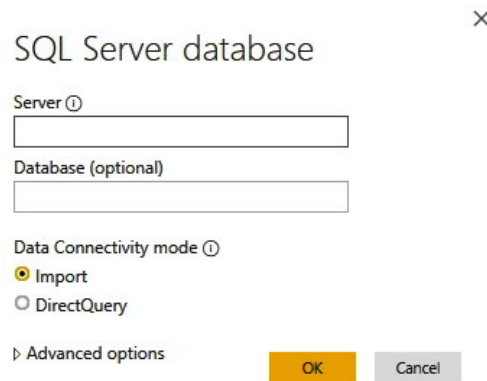


Figure 4.24 – Import or DirectQuery

Once connected to the server, you will need to authenticate yourself and then you can select the tables that you would like to load or transform, which is very similar to what we did when we connected to the SQL Server database in Power BI.

If you click on **Transform Data**, you will be taken to **Power Query Editor**, where you can clean your data. If you click on **Load**, it will import the data into Power BI:

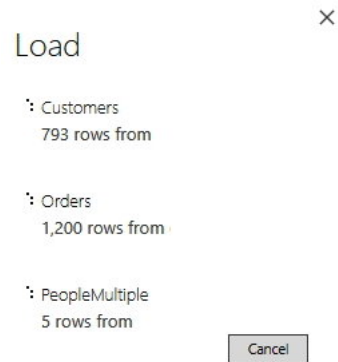


Figure 4.25 – Importing the data into Power BI

The data that you have imported into Power BI is displayed as individual datasets in the data view, which can be found on the right-hand side of the screen:

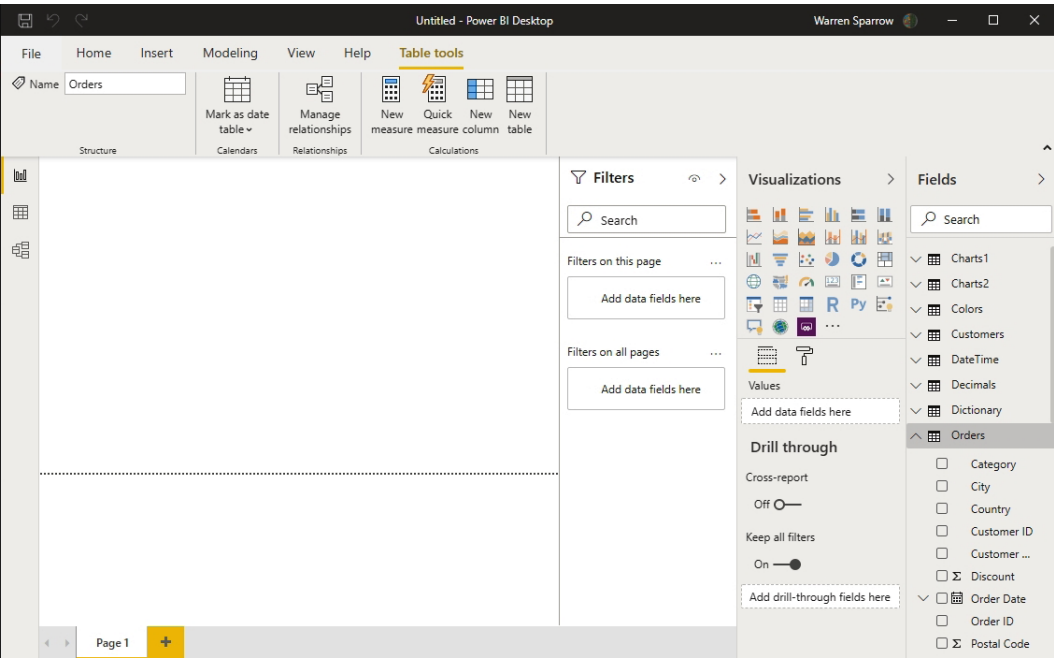


Figure 4.26 – Datasets in the data view

If you click on **Edit Queries**, this will take you to **Power Query Editor**:

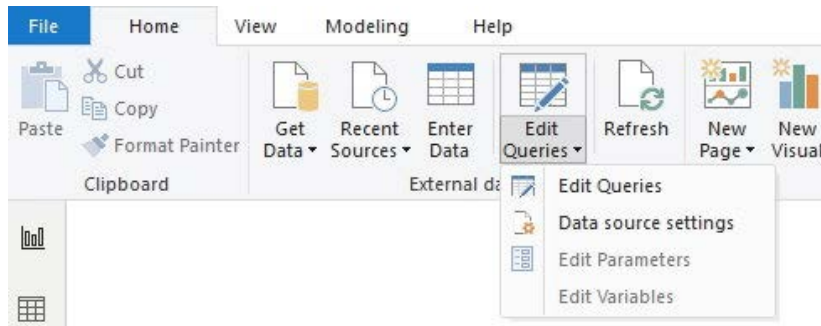


Figure 4.27 – Edit Queries menu

This is one of the most powerful ways to connect data from your SQL server as it is always up to date and numerous people can work using the same data. Of course, you also have the extra security of needing to authenticate with usernames and passwords.

Understanding custom connections

There are various ways in which we can connect data to Power BI and Power Query. Some of the most common ones can be found in the **Get Data** menu on the **Home** ribbon, but we can load other data that is not as obvious. We are going to load data from the web again, but this time, we will load a CSV file so that you can see that we will have to clean the data before we can use it. Click on the **Data** tab and select the **From Web** option, the same way in which we did this earlier. Paste in the following URL, <https://github.com/PacktPublishing/Learn-Power-Query/Chapter4-Data/master/SalesData10.csv>, and then click **OK**:

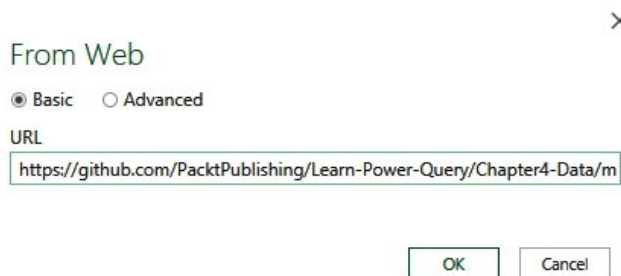


Figure 4.28 – From Web URL

When the data has loaded, Excel is clever enough to know that this is a **Comma Delimiter** file. If you are connecting to a different type of file, or the **Delimiter** type is not a comma, you can change this by selecting the appropriate type from the **Delimiter** drop-down box, as shown here:

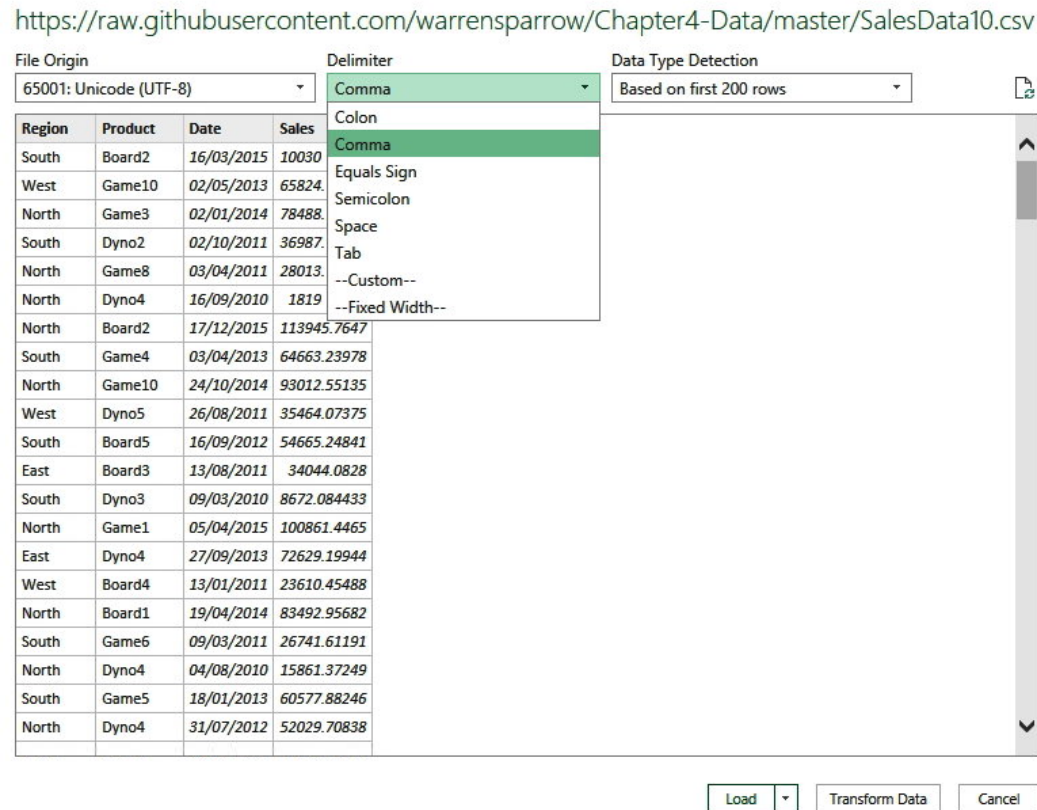


Figure 4.29 – Changing the Delimiter type

Click on **Transform Data** and then make the appropriate changes in **Power Query Editor**. When you have finished, click on **Close and Load To**, at which point you can create a table or load it into your data model:



	A	B	C	D	E
1	Region	Product	Date	Sales	
2	South	Board2	16/03/2015	£ 100,308.76	
3	West	Game10	02/05/2013	£ 65,824.37	
4	North	Game3	02/01/2014	£ 78,488.78	
5	South	Dyno2	02/10/2011	£ 36,987.85	
6	North	Game8	03/04/2011	£ 28,013.17	
7	North	Dyno4	16/09/2010	£ 18,192.80	
8	North	Board2	17/12/2015	£ 113,945.76	
9	South	Game4	03/04/2013	£ 64,663.24	
10	North	Game10	24/10/2014	£ 93,012.55	
11	West	Dyno5	26/08/2011	£ 35,464.07	
12	South	Board5	16/09/2012	£ 54,665.25	
13	East	Board3	13/08/2011	£ 34,044.08	
14	South	Dyno3	09/03/2010	£ 8,672.08	
15	North	Game1	05/04/2015	£ 100,861.45	
16	East	Dyno4	27/09/2013	£ 72,629.20	
17	West	Board4	13/01/2011	£ 23,610.45	
18	North	Board1	19/04/2014	£ 83,492.96	
19	South	Game6	09/03/2011	£ 26,741.61	
20	North	Dyno4	04/08/2010	£ 15,861.37	
21	South	Game5	18/01/2013	£ 60,577.88	
22	North	Dyno4	31/07/2012	£ 52,029.71	
23	North	Dyno4	12/07/2015	£ 106,047.48	
24	North	Game3	10/05/2016	£ 121,177.62	
25	North	Board4	16/12/2012	£ 58,902.20	
26	South	Game10	31/07/2011	£ 34,016.97	
27	West	Game7	21/05/2015	£ 103,118.11	
28	North	Game9	29/07/2013	£ 70,349.70	

Figure 4.30 – A table from a CSV file from the web

Custom connections can be made in two ways, as we will see in the following sections.

Connecting from Workbook

Every now and again, you will want to merge existing information from one worksheet into a table, but there are too many gaps and it would be difficult to create just one table. The following screenshot shows a survey that was done over a few days about one of our products, namely, **Game1**:

	A	B	C	D	E	F	G
1	Product : Game1						
2							
3			Monday		Overall impression of the product	Quality of the product	Value for money
4					4	3	3
5					3	3	4
6					4	4	5
7					4	4	
8					4	4	4
9							
10			Average		3.8	3.6	4.0
11							
12							
13			Tuesday		Overall impression of the product	Quality of the product	Value for money
14					3	4	4
15					3	4	4
16					3	3	4
17					4	3	3
18					3	4	4
19							
20			Average		3.2	3.6	3.8

Figure 4.31 – Survey about the Game 1 product

In the survey, each criterion is based on a scale of 1 to 5, with 5 being the best and 1 being the worst. It would take too long to do this using conventional methods, but it is possible to import the entire sheet into **Get & Transform** so that we can clean this data. Let's see how this is done:

1. Start by creating a new sheet and then click on the **Data** tab. Then, click on **Get Data | From File | From Workbook**:

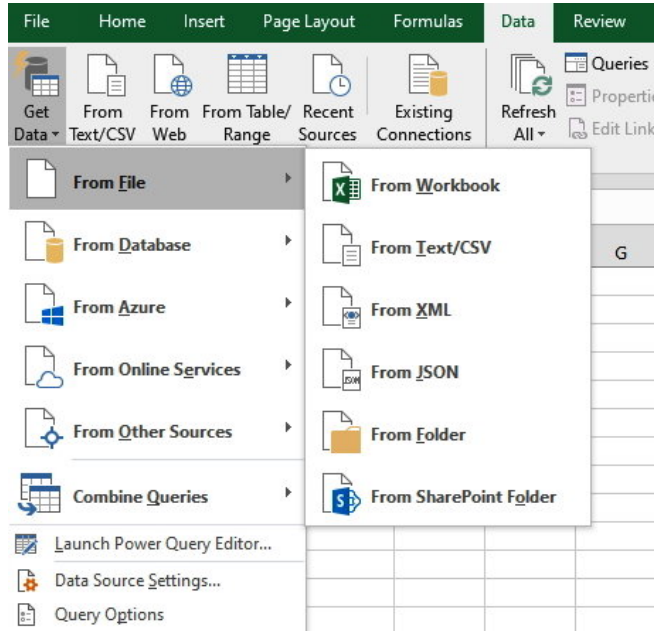


Figure 4.32 – From Workbook selection

2. Select the file that you would like to import and then click on **Import**. Once the file has loaded, select the worksheet and then click **Transform**, which will bring up the **Power Query Editor** window:

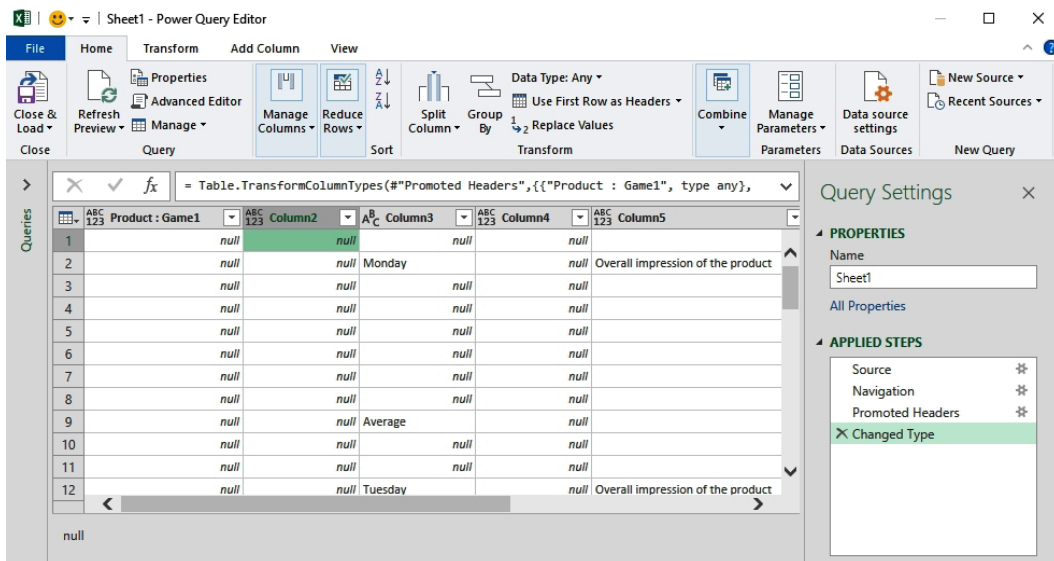


Figure 4.33 – Power Query Editor

3. This is where we can clean the data. We will go into the process of cleaning the data in much greater detail in *Chapter 7, Automating Reports in Power Query*. This chapter deals with importing the data into a data model or table.
4. To look at this very briefly, to clean some of the data, you can right-click on **Column2** and then select **Remove** as we do not need this column. You can do the same with **Column4** and the **Product:Game1** column if you wish.
5. Select the new **Column2** and then right-click and select **Fill Down**. You can rename the column headings at the top and you can also filter the rows, thus getting rid of the averages and the null values.
6. One of the important things to realize is that **Power Query Editor** keeps a record of everything that you do in the **APPLIED STEPS** window. The reason for this is that if you now import or merge a different worksheet into this table, all the steps that you created will automatically be applied to the other tables that are imported:

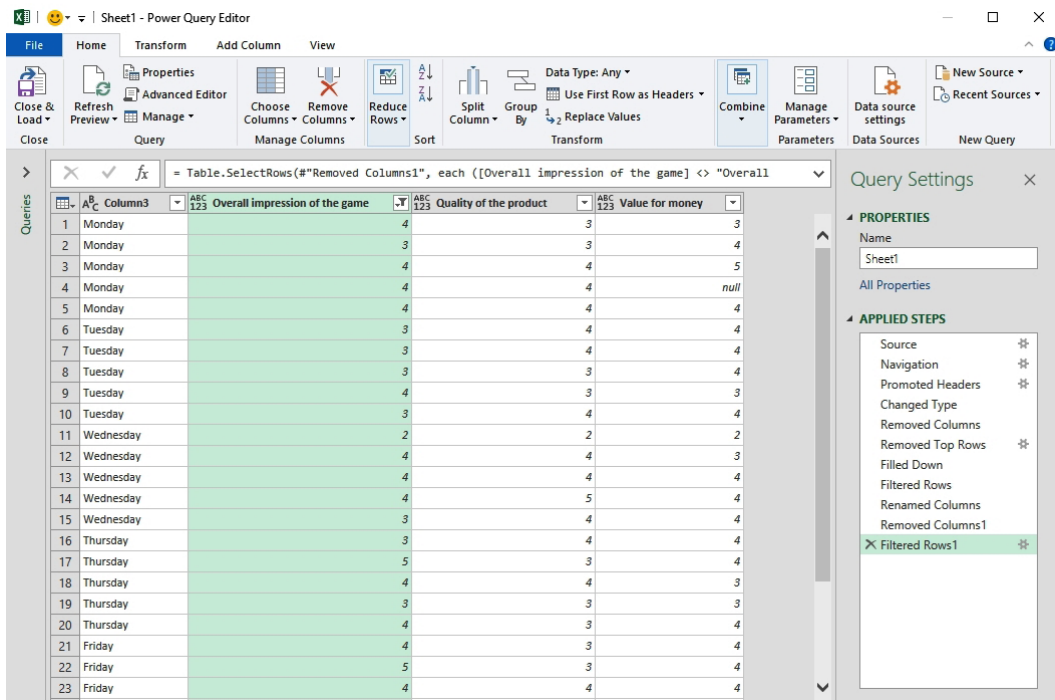


Figure 4.34 – Power Query Editor Applied Steps – part 1

7. Click on **Close and Load** to insert your newly formatted table. This was a great deal easier to clean using **Get & Transform** than trying to edit it manually in Excel:

	A	B	C	D
1	Column3	Overall impression of the product	Quality of the product	Vlaue for money
2	Monday	4	3	3
3	Monday	3	3	4
4	Monday	4	4	5
5	Monday	4	4	
6	Monday	4	4	4
7	Tuesday	3	4	4
8	Tuesday	3	4	4
9	Tuesday	3	3	4
10	Tuesday	4	3	3
11	Tuesday	3	4	4
12	Wednesday	2	2	2
13	Wednesday	4	4	3
14	Wednesday	4	4	4
15	Wednesday	4	5	4
16	Wednesday	3	4	4
17	Thursday	3	4	4
18	Thursday	5	3	4
19	Thursday	4	4	3
20	Thursday	3	3	3
21	Thursday	4	3	4
22	Friday	4	3	4
23	Friday	5	3	4
24	Friday	4	4	4
25	Friday	4	3	5
26	Friday	4	4	3
27	Saturday	5	4	5
28	Saturday	3	3	4
29	Saturday	4	4	4
30	Saturday	4	3	4

Queries & Connections

Queries | Connections

1 query

Sheet1

30 rows loaded.

Figure 4.35 – Power Query Editor Applied Steps – part 2

That's it – we are done connecting from Workbook.

Connecting from a folder

This is one of the easiest ways to get information into your table or data model. The one bit of advice we will provide is that you must have the files that you want to use in the database in the same folder. It is worth creating a dedicated folder for you to use when you do this. It will save you time and effort later. Also, make sure that you provide the folder with a suitable name, as you might end up with many of these folders. You will notice from the following screenshot that I have CSV files in the folder, but that I may also put Excel and text files in there too:

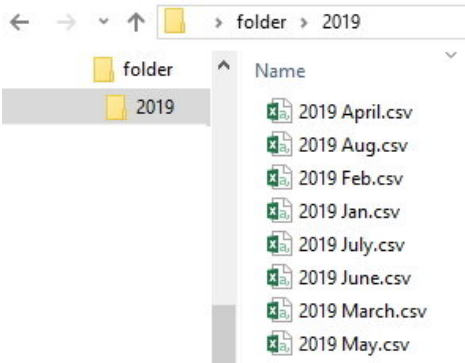


Figure 4.36 – Folder containing all the relevant files

Each of these files is a months' worth of sales for our business and contains approximately 10 to 15,000 rows' worth of data. Each file contains four columns, but you could import many more if required.

Note

Each file needs to have the same number of columns, with the same headers in the same order. Generally speaking, if you are exporting from a CRM, POS, or other such system, this would normally be the case anyway.

The following screenshot snippet shows this:

	A	B	C	D
1	Region	Product	Date	Sales
2	East	Game1	30/06/2019	13490.16
3	West	Game8	30/06/2019	13933.9
4	West	Game10	30/06/2019	13673.88
5	North	Board4	30/06/2019	13864.81
6	South	Prod T	30/06/2019	14153.97
7	North	Dyno2	30/06/2019	14404.17

Figure 4.37 – A CSV file with the same headings

We can do this as follows:

1. Launch Excel and click on the **Data** ribbon. From here, select **Get Data**, then **From File**, and then click **From Folder**.
2. Excel versions prior to 2016 look a little bit different to this as you have to click on **POWER QUERY**, select **From File**, and then click on **From Folder**:

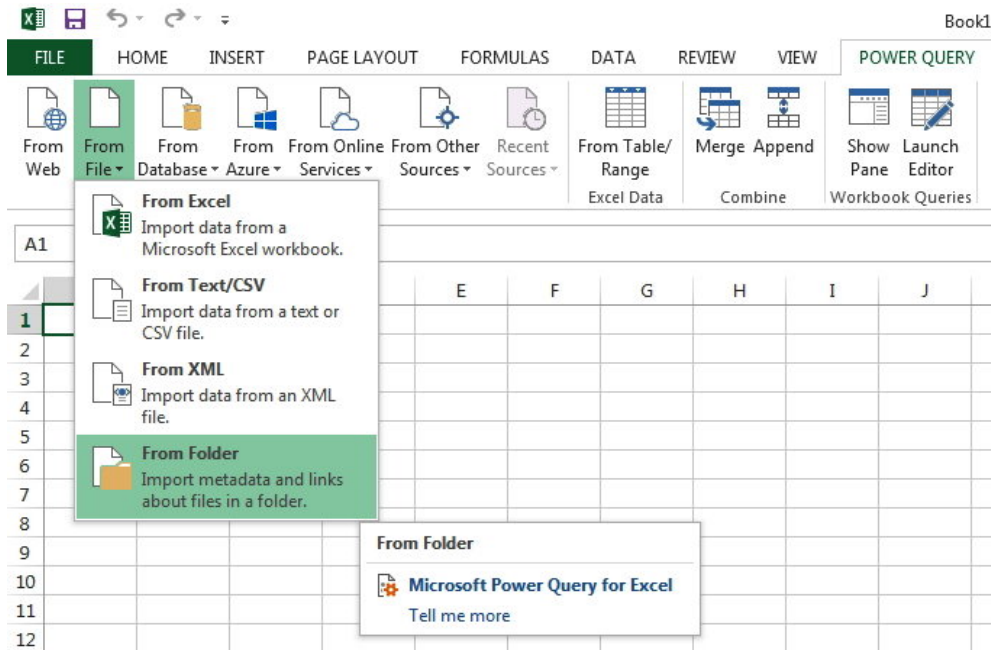


Figure 4.38 – Excel (prior to 2016) From Folder menu

3. Either copy and paste the folder path or click on **Browse** and find the folder containing all your files. Then, click **OK**.

You will see that a dialog box appears containing the different files that you have in your folder. You now have the option of doing the following:

- **Combine**
- **Combine & Edit**
- **Combine & Load**
- **Combine & Load To...**
- **Load**

- **Load to**
- **Transform Data**
- **Cancel**

I usually select **Transform Data**, which is the same as **Combine & Edit**:

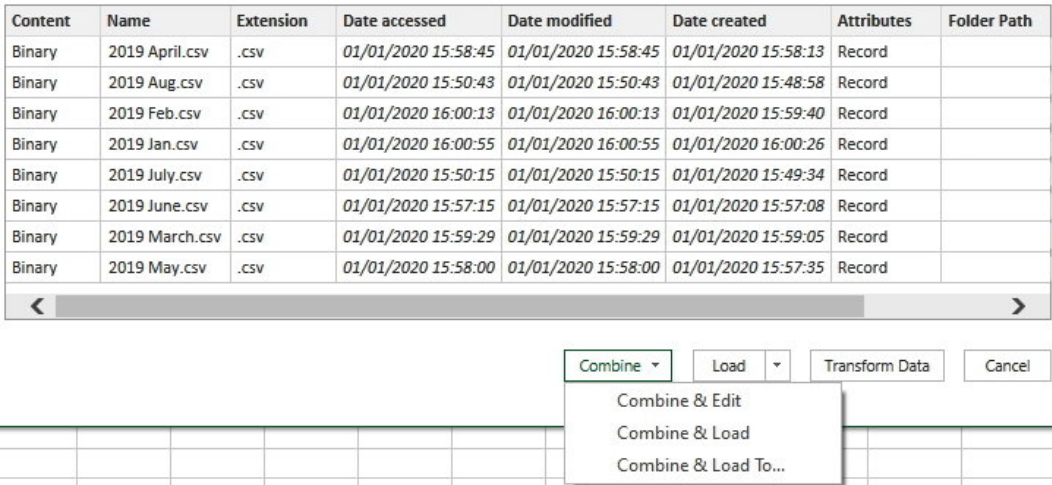


Figure 4.39 – Combine & Edit menu

4. At this point, if there are files that you do not need, you would select those files and remove them. But we need all these files, so I am going to combine the files by clicking on the two down arrow keys on the **Content** column:

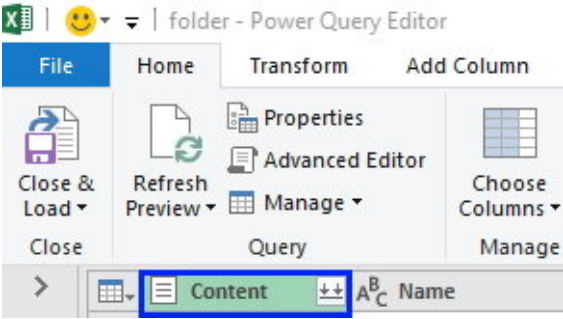


Figure 4.40 – Content column option

- When the **Combine Files** dialogue box opens, we will see a preview of our first file. Since these are CSV files, we will need to make sure that they are in the correct format. Selecting the file format helps Power Query know what format things such as the date and time should be in. If the data doesn't look correct, make sure that **Delimiter** is set to the correct type:

Combine Files

Specify the settings for each file. [Learn more](#)

Example File:

First file ▾

File Origin	Delimiter	Data Type Detection
1252: Western European (Windows) ▾	Comma ▾	Based on first 200 rows ▾

Region	Product	Date	Sales
North	Game9	30/04/2019	10636.5783
West	Game8	30/04/2019	10957.37768
East	Dyno1	30/04/2019	10703.01231
North	Dyno5	30/04/2019	10672.33469
South	Board5	30/04/2019	11274.0915
South	Board2	30/04/2019	11342.5543

Figure 4.41 – Combine Files dialogue box

To combine files, perform the following steps:

- Start by selecting which of the files you would like to use as the sample. Provided that they are all in the same format, it should not make any difference.
- In the case of CSV files, there will be a comma delimiter, but you might have text files that have space or tab delimiters.

3. Lastly, make sure that the first 200 rows will be enough for Power Query to detect the data type. Once completed, click on **OK**:

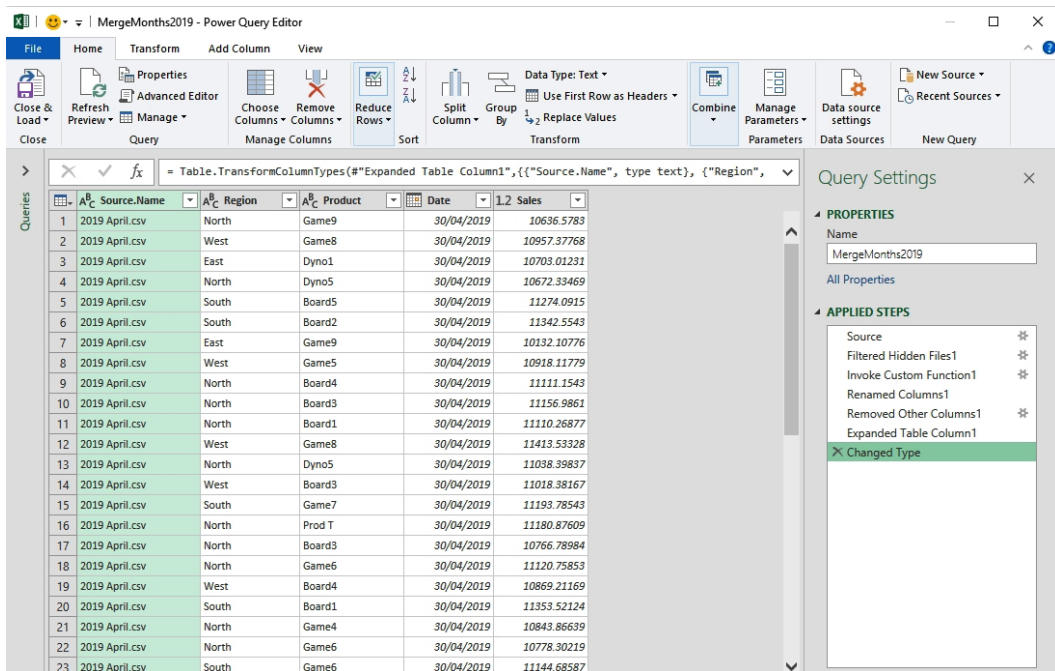


Figure 4.42 – Power Query Editor

From the preceding screenshot, you can see the following:

- Power Query has taken the files and merged them into one table. On the left-hand side of the preceding screenshot is the queries that Power Query generated when you clicked on the **OK** button previously.
- The **Transform Sample** file from CSV is an important query as this is the basis of all the other files and is used as the template for all existing and future files that are added to this folder. If you wanted to change what this query looks like, this is the query you would need to modify.
- **csv** at the bottom-left of the queries panel is the final query that merged the files into one table. This is the query that you are currently looking at. We can now transform this in the same way we transformed the other tables in **Power Query Editor**. Right-click on **Source.Name** and then select **Remove**. You can also change the data type of the **Sales** column to **Currency**.

- On the right-hand side, you will see **Query Settings**. I would rename **Name** under **PROPERTIES** to something more appropriate. Once you have renamed this, you will notice that the named **csv** query on the left will also change to the same name that you have just typed in.
- Underneath this is **APPLIED STEPS**, where you will see a list of the steps that you have completed to clean your data:

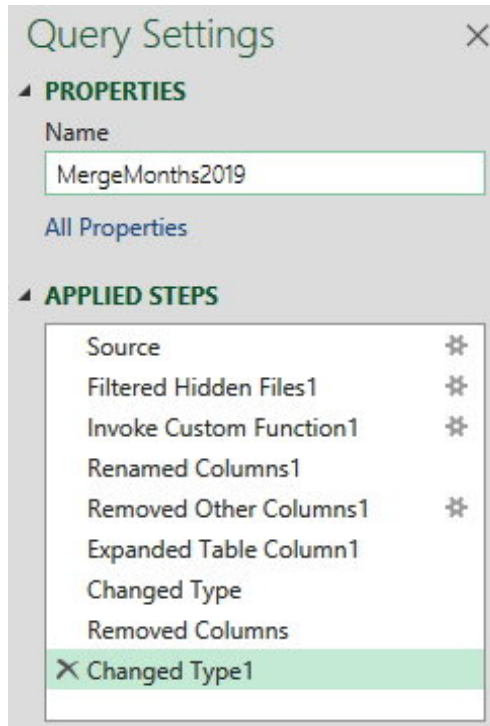


Figure 4.43 – Query Settings window

- Once completed, click on **Close & Load** at the top-left of the window. From here, you have the option of clicking **Close & Load**, which will create a table in a new worksheet with the query you have created, or you can click on **Close & Load To**, which will load this in the Power Pivot Data model. I used the latter option and added this to the data model, as shown here:

	A	B	C	D
1	Region	Product	Date	Sales
2	North	Game9	30/04/2019	10636.5783
3	West	Game8	30/04/2019	10957.3777
4	East	Dyno1	30/04/2019	10703.0123
5	North	Dyno5	30/04/2019	10672.3347
6	South	Board5	30/04/2019	11274.0915
7	South	Board2	30/04/2019	11342.5543
8	East	Game9	30/04/2019	10132.1078
9	West	Game5	30/04/2019	10918.1178
10	North	Board4	30/04/2019	11111.1543
11	North	Board3	30/04/2019	11156.9861
12	North	Board1	30/04/2019	11110.2688
13	West	Game8	30/04/2019	11413.5333
14	North	Dyno5	30/04/2019	11038.3984
15	West	Board3	30/04/2019	11018.3817
16	South	Game7	30/04/2019	11193.7854
17	North	Prod T	30/04/2019	11180.8761
18	North	Board3	30/04/2019	10766.7898
19	North	Game6	30/04/2019	11120.7585
20	West	Board4	30/04/2019	10869.2117
21	South	Board1	30/04/2019	11353.5212
22	North	Game4	30/04/2019	10843.8664
23	North	Game6	30/04/2019	10778.3022
24	South	Game6	30/04/2019	11144.6859
25	West	Game5	30/04/2019	11070.3305
26	North	Game3	30/04/2019	11309.3888
27	South	Board3	30/04/2019	11309.8237
28	North	Prod T	30/04/2019	11340.7097
29	West	Game6	30/04/2019	11052.1908
30	South	Dyno5	30/04/2019	11394.4389

Figure 4.44 – Completed table from the query

- The actual charm with this is that we had completed the first three quarters of the yearly sales data and we now have the last quarter that we would like to add to this.
- Insert the other files into the same folder that contains the other files:

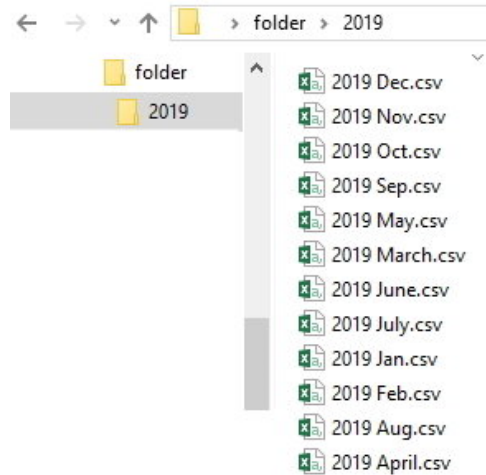


Figure 4.45 – Inserting additional months into the same folder

- In Excel, right-click anywhere in the table and click **Refresh** from the shortcut menu provided:

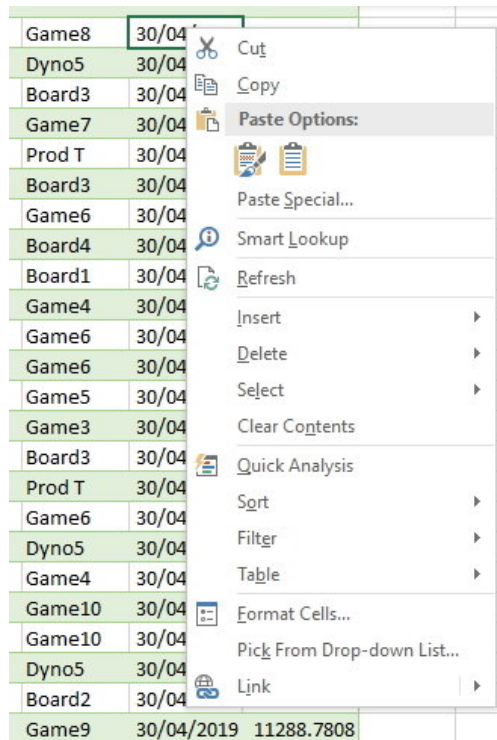


Figure 4.46 – Refreshing the query data

- Although it refreshes very quickly since it already contains the query template, it actually reloads all the data in the folder and runs the steps that you set up in **Power Query Editor** before reloading the merged table in Excel.
- If you loaded this into the Power Pivot data model, you can right-click on **MergeMonths2019** and then select **Refresh** or click on the **Refresh** icon in the **Queries & Connections** window, as shown here:

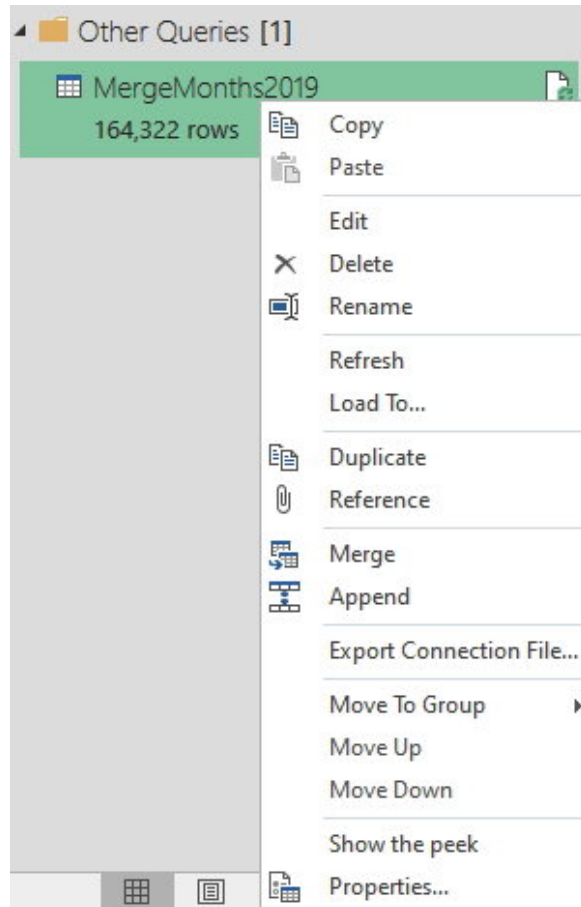


Figure 4.47 – Refreshing the query data from the Queries & Connections window

Next, we move on to understanding the various data source settings.

Exploring data source settings

Once you have the connections to the data, it is possible to edit the credentials for each connection you have used, not just for the active connection, but for all the connections that you have made. This means that if your data source is located in a folder called `sales`, but now you want the connection to be in a subfolder of `sales`, you need to be able to change the source settings so that Power Bi/Power Query knows where to look. We will learn how to do this in Excel next.

From Excel

There are different ways in which you can get to the same destination. I have added three here:

- If you are in Excel and you have loaded the query into a table, then you can click anywhere in the table and select **Edit** from the **Query** ribbon. This will open **Power Query Editor**:

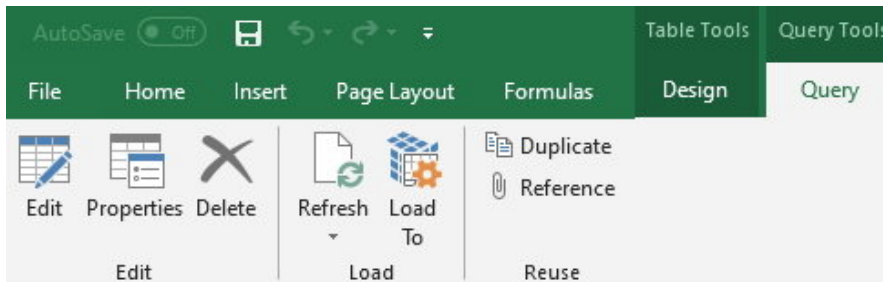


Figure 4.48 – Query ribbon

- Alternatively, you can right-click the query in the **Queries & Connections** window and select **Edit**, which will also open **Power Query Editor**.
- Finally, in **Power Query Editor**, select **Data source settings** from the **Home** ribbon, which will open the **Data source settings** window:

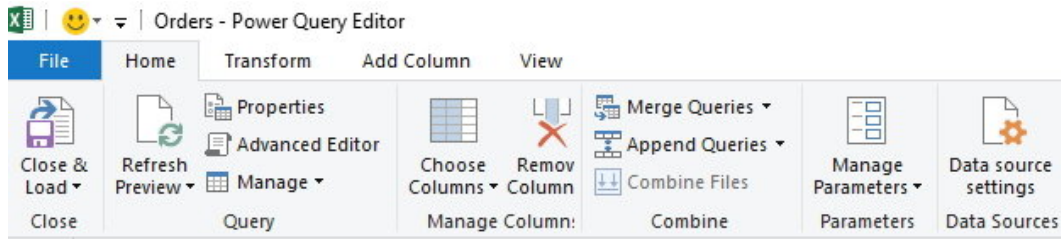


Figure 4.49 – Power Query Editor

In the **Data source settings** window, you can change the source, edit the permissions, and clear the permissions:

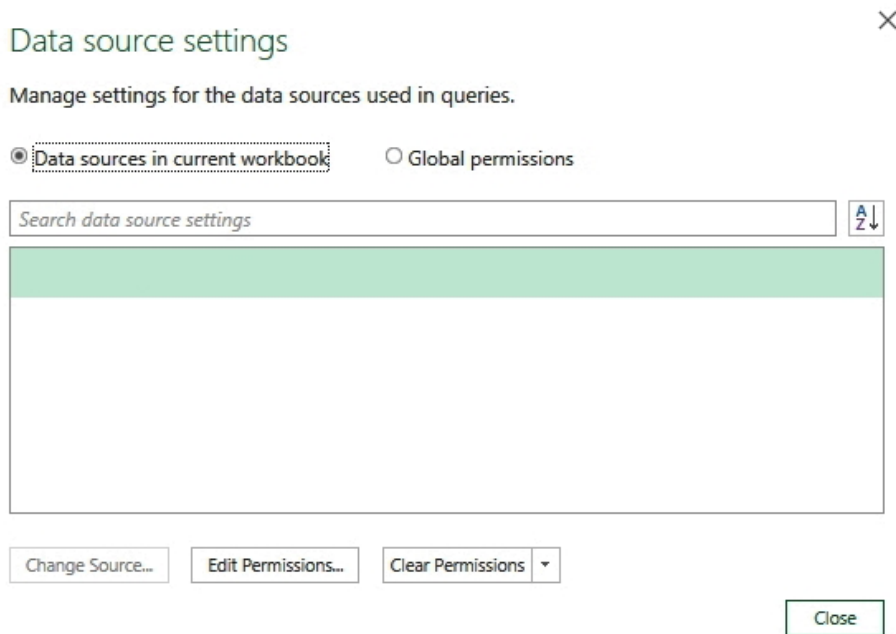


Figure 4.50 – Data source settings window

This is how it goes:

- When you click on the **Change Source...** button, you can change the source from your current data source to a different data source. For this, you would follow the same steps that you followed when you created the data source.
- When you click on **Edit Permissions...** to edit the database credentials, this allows you to change your credentials for when you log in or authenticate your credentials, either through **Windows**, the **Database SQL**, or your **Microsoft Account**.

This is shown here:

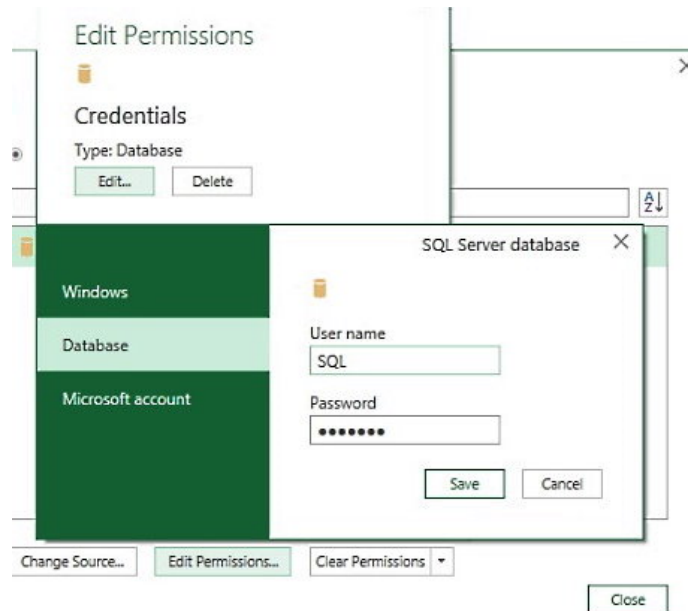


Figure 4.51 – Edit Permissions window for the SQL Server database

Once you have created the connections that you need, you generally don't need to change the data source settings. Of course, if you have created a Power BI report and you have published it and you now need to create another Power BI report on different data that is similar in structure to the one you have already created, then by changing the data source, you can quickly create a report on the new data. When changing the data source settings, you may also need to reauthenticate with your SQL Server or change permissions.

Summary

From this chapter, you have learned about the numerous ways in which you can connect data to Power BI and Excel. One of the greatest advantages of this is the different types of connections that Power BI can connect to, including from a single file, folder, SQL database, Azure to Online services, and so on. One of the more powerful features is that you can connect to other data sources that are not listed and that you can create custom connections. Power BI creates datasets that are both the data source and the data itself. Every time you connect to your data, Power BI creates a dataset. This is then used for creating your reports and visualizations.

In the next chapter, you will learn how to transform the data that you now have in Power BI and Power Query.

Section 2: Power Query Data Transformations

In this section, you will learn how to transform data using the Power Query editor and write functions in Power Query. We will discover a number of functions, including `IF`, `index`, and `modulo`, and create parameters to alter query paths in a table. You will work with dashboards, as well as learn how to create multi-dimensional reporting and automated reports.

This section comprises the following chapters:

- *Chapter 5, Transforming Power Query Data*
- *Chapter 6, Advanced Power Queries and Functions*
- *Chapter 7, Automating Reports in Power Query*
- *Chapter 8, Creating Dashboards with Power Query*

5

Transforming Power Query Data

In this part of this book, you will learn how to transform data using Power Query in a multitude of ways and understand why it is important to prepare your dataset prior to analysis. You will apply **Unpivot** and **Pivot** to a dataset to structure data in the correct tabular format; work with row and column tools such as split, merge, duplicate, and extract; and use conditional columns to display the output you desire from `if...then...else` conditions. The automatic background refresh setting will also be discussed in this chapter. We will learn how to extract ages from a date column, which saves a huge amount of time as you would normally have to work this out for each individual column from a date column entry. In addition, we will look at using various delimiter constraints, as well as grouping data from various rows into a single value.

In this chapter, we're going to cover the following main topics:

- Turning data with the unpivot and pivot tools
- Basic column and row tools
- Merging and appending tools
- Grouping data
- Working with extraction tools

Technical requirements

With the chapters we've studied so far, you should now be a confident user of Excel and be able to manipulate data, rows and columns, as well as be proficient with the advanced sort and filter tools available. You should be able to create a query using the **Get & Transform** tool, the **Close & Load** option, and also launch and operate the **Queries & Connections** pane in Excel.

The examples used in this chapter can be accessed from <https://github.com/PacktPublishing/Learn-Power-Query/>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=n1swG3Z44mM&list=PLcLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=6&t=515s.

Turning data with the unpivot and pivot tools

Transforming data means to shape data by renaming tables or columns or making the data presentable for analysis. You will master the use of the Pivot and Unpivot tools to transform tabular data into an accepted tabular format. After using these tools, we will delete any unnecessary columns and rename column headers, and name queries. We will take note of the applied steps and learn how to refresh data sources in Power Query. These steps are all necessary to get the data prepared for further analysis or reporting.

Data is presented in many forms for many different reasons. Some could be made visually appealing, such as a simple financial budget report with financial years as rows and months as column headers, while others could be more complex and used for analysis using PivotTables and/or storing data in a data storage application such as Power BI. Always envisage the aim of the dataset you are working with and prepare the data prior to performing any calculations or analysis. Power Query is definitely the go-to tool for data transformation in such cases.

The pitfall of not structuring a dataset correctly could lead to many analysis problems, and we simply end up wasting valuable time having to construct many calculations. You would not have to do this if the dataset was prepared correctly from the outset.

When getting data ready for use in PivotTables, for example, there are numerous steps to ensure you get the most out of summarizing and analyzing large datasets. We should adhere to these prior to using any analysis tool:

- Make sure that there is no duplicated data in the worksheet.
- Remove any filters you have applied to the data and ungroup any cells you have grouped using the Outline feature in Excel.
- Format the data type for each column in the worksheet. This means that we need to format dates with the appropriate date format, with values as either number, currency or accounting, and so on. Make sure that the column headings are labeled correctly so that they are easily understood and are not long-winded descriptions.
- Never include total calculations, subtotals, or average columns. The reason for this is that once we have cleaned and transformed data in Power Query, it is then analyzed by Power Pivot, Power BI, or Excel tools to reach different results.
- Get rid of any blank cells from the data source.
- The last thing you need to do is format your data as a table – the reason that this is so important was explained in *Chapter 4, Connecting to Various Data Sources Using Get & Transform*. It is extremely important that the data is formatted as a table, either prior to or when using the **Get & Transform** option, as it will ensure that any extra rows or columns that are inserted are automatically included when refreshing the data from various sources.

Let's have a look at an example of a poorly designed dataset and how we can fix this using the Unpivot tool.

The following screenshot shows a source dataset that has been poorly designed for analysis in Power Pivot or any other analysis tool. The dataset does not comply with the rules outlined in the preceding list:

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
						QUARTER ONE			QUARTER 2			QUARTER 3						
1	Season	Winery	Label	Region	Cost Per Case	JAN SALES	FEB SALES	MAR SALES	Q1 SALES	APR SALES	MAY SALES	JUNE SALES	Q2 SALES	JULY SALES	AUG SALES	SEPT SALES	Q3 SALES	TOTAL SALES
2	Winter	Matts Winery	Cab Savon North	£165.00	450	526	926	£313 830.00	779	1144	929	£470 580.00	305	1148	333	£294 690.00	£1 079 100.00	
3	Winter	Matts Winery	Cab Savon North	£165.00	550	1038	409	£329 505.00	416	724	971	£348 315.00	530	565	327	£234 630.00	£912 450.00	
4	Autumn	Matts Winery	Cab Savon North	£165.00	575	1025	331	£318 615.00	790	364	1198	£388 080.00	983	892	861	£418 440.00	£1 125 135.00	
5	Spring	Matts Winery	Cab Savon North	£165.00	650	523	723	£312 040.00	774	392	949	£349 975.00	1105	530	576	£364 815.00	£1 028 630.00	
6	Summer	Matts Winery	Cab Savon South	£165.00	320	800	306	£235 290.00	360	363	778	£247 665.00	391	368	1043	£297 330.00	£780 285.00	
7	Winter	Matts Winery	Cab Savon South	£165.00	325	938	558	£300 465.00	723	351	572	£271 590.00	747	1196	1069	£496 980.00	£1 069 035.00	
8	Autumn	Matts Winery	Cab Savon South	£165.00	330	420	648	£230 670.00	851	342	395	£262 020.00	451	403	519	£226 545.00	£719 235.00	
9	Spring	Matts Winery	Cab Savon South	£165.00	350	804	1015	£367 885.00	680	534	529	£287 595.00	792	978	737	£413 855.00	£1 059 135.00	
10	Summer	Matts Winery	Cab Savon East	£165.00	350	796	593	£281 985.00	743	827	508	£342 970.00	768	651	870	£377 685.00	£1 002 540.00	
11	Winter	Matts Winery	Cab Savon East	£165.00	360	1083	851	£378 510.00	449	888	355	£279 180.00	562	562	892	£332 640.00	£990 330.00	
12	Autumn	Matts Winery	Cab Savon East	£165.00	370	780	1164	£381 810.00	532	373	938	£394 095.00	964	1039	864	£473 055.00	£1 158 960.00	
13	Spring	Matts Winery	Cab Savon East	£165.00	375	860	412	£271 755.00	897	830	521	£370 920.00	764	534	969	£374 055.00	£1 016 730.00	
14	Summer	Matts Winery	Cab Savon West	£165.00	230	1080	489	£296 835.00	669	732	426	£301 455.00	760	1018	959	£451 605.00	£1 049 895.00	
15	Winter	Matts Winery	Cab Savon West	£165.00	235	1958	1083	£393 690.00	618	1082	727	£490 455.00	879	787	696	£388 080.00	£1 182 225.00	

Figure 5.1 – Poorly designed source data

It contains unnecessary calculation columns per quarter, as well as the final total sales. The merged column headings can also be removed. Once this is complete, the **Region** column can also be converted from rows into columns within Power Query.

After following the customizations to prepare the data for analysis, the dataset looks much more presentable, but we need to tweak it a little more using the Unpivot tool in Power Query. To demonstrate the Unpivot tool, we will use the sales columns from January to September. Instead of having separate columns for each of the sales figures for the three quarters, we will translate the data from all nine columns into two columns with rows. Let's get started:

1. Open the dataset named `YearlyProductSales.xlsx` in Power Query. You can achieve this using Excel or Power BI.
2. Select all nine sales columns in the dataset. Then, click on **Transform | Unpivot | Unpivot Columns** from the **Any Column** group:

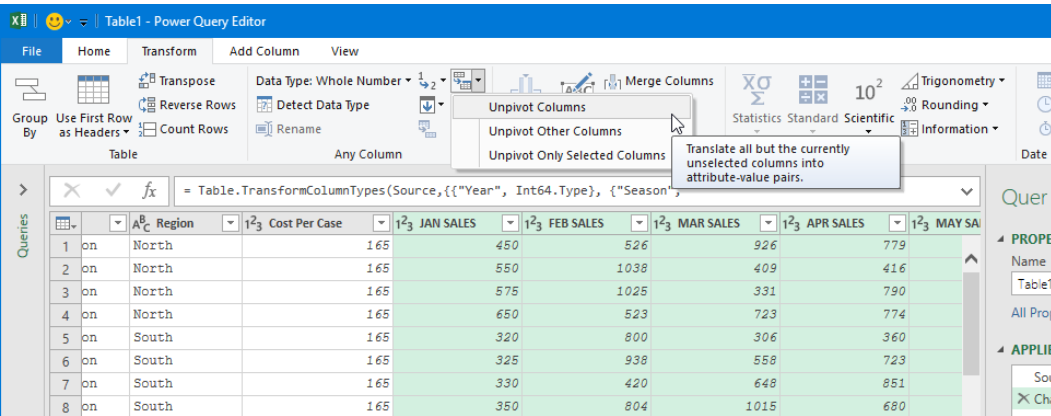


Figure 5.2 – Using the Unpivot tool

3. The result provides a column for **Attribute**, listing the months, and a column for **Value** instances, which are associated with each month:

Cost Per Case	Attribute	Value
165	JAN SALES	450
165	FEB SALES	526
165	MAR SALES	926
165	APR SALES	779

Figure 5.3 – Attribute and Value columns after Unpivot

4. Rename the columns so that they suit the table's content. Simply double-click on the **Attribute** column heading and replace it with text of your choice. Do the same for the **Value** column.

5. If you need to undo a previously performed step or if you would like to identify whether a certain step was performed on the query data, simply visit the **APPLIED STEPS** window to view, delete, rename, or reorder steps you have applied to the data. To delete a step, click on the red cross to the left of the applied step, or right-click on a particular step to access various options.
6. Let's perform the Pivot column option, which does the opposite of the Unpivot feature, by using the names of data located in rows to display across columns. Make sure you are using the *SafestSolutionsLaw.xlsx* data source for this example.
7. Select the **CASE TYPE** column to use the items within the column to base the new columns on. Then, click **Transform | Pivot**:

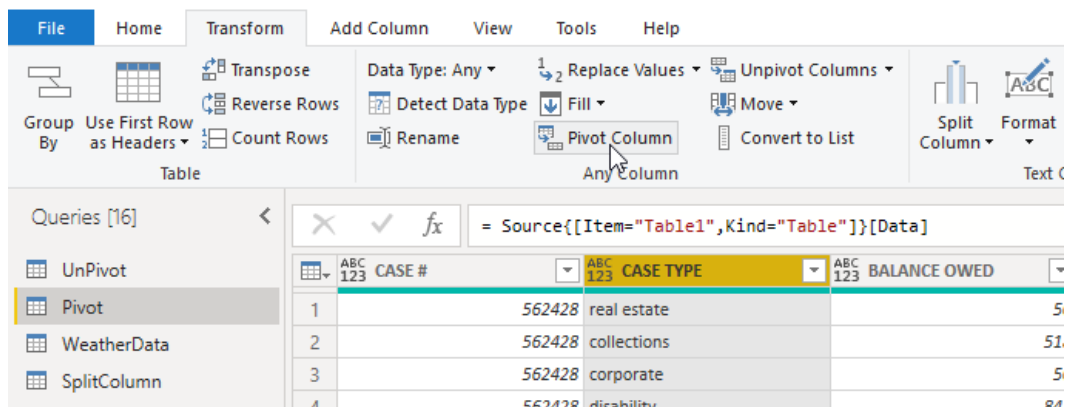


Figure 5.4 – Transform Pivot

8. For **Values Column**, select **BALANCED OWED** from the list provided:

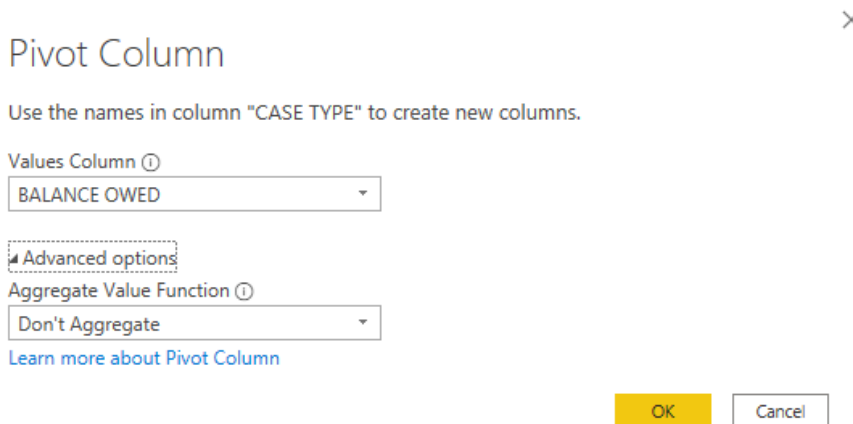
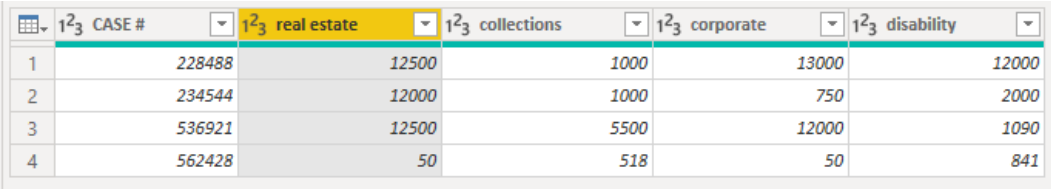


Figure 5.5: Result of the Pivot column feature

9. Select **Advanced** options and choose **Don't Aggregate** for the **Aggregate Value Function** since we do not want to sum values at this point.
10. Click on **OK** to view the result of the Pivot:



	1 ² ₃ CASE #	1 ² ₃ real estate	1 ² ₃ collections	1 ² ₃ corporate	1 ² ₃ disability
1	228488	12500	1000	13000	12000
2	234544	12000	1000	750	2000
3	536921	12500	5500	12000	1090
4	562428	50	518	50	841

Figure 5.6: Refresh Notification Bar

We will now see two created queries in the **Queries** pane. Rename the first query to **UnPivot** and the second query to **Pivot**:

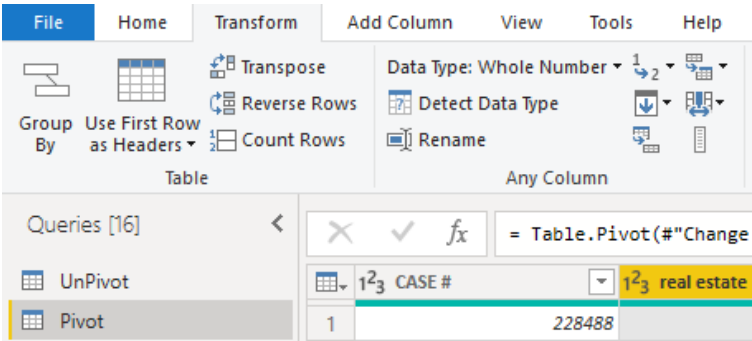


Figure 5.7 – Renamed queries in the Queries pane

With that, you should be confident in using the **Pivot** and **UnPivot** options within Power Query to transpose row and column data. In the next section, we will investigate the various refresh options and understand how the refresh process works.

Refreshing data

Once we have created the queries, the source data behind the queries is often updated by entering and manipulating the new data. For example, new data files could be received by the finance department that would need to be added to the existing datasets, live web data is updated, or the user adds more data to the dataset manually. The beauty of refresh is that you create queries only once and then use that query over and over again just by refreshing the connection to the data source, wherever that may be. When calculating in Excel, the data engine will recalculate automatically, every time something changes in the workbook data. Power Query, however, needs to be told when to refresh data. It is a critical step to understand refresh so that the end result of your data transformation is accurate.

With refresh, you do not have to remember any steps you performed the previous time you worked on the data. As you already know, Power Query records these steps as **APPLIED STEPS** using the M language code. Therefore, when you refresh it, it is a simple process for Power Query to refresh everything by loading or importing new data and then performing the steps once again, including the new data.

If we have connections to Excel workbook data, then the **Data | Refresh All** option from the **Queries & Connections** group in Excel is the way to refresh connections.

To refresh all queries directly within Power Query, click on the **Home | Refresh Preview** icon, and then select **Refresh All**. Alternatively, select **Refresh Preview** to update only the current query. The refresh notification bar will populate just above the formula bar to remind you to refresh your queries when working within Power Query, as shown in the following screenshot:

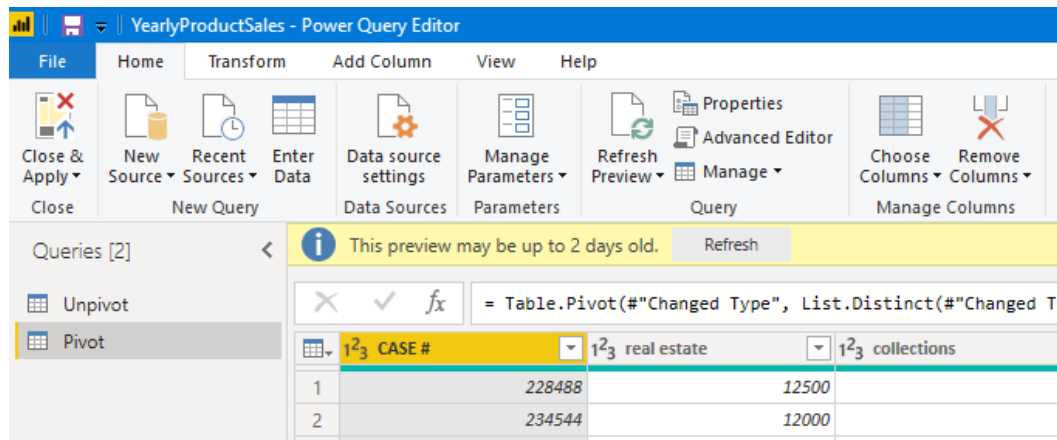


Figure 5.8 – The Refresh notification bar

Remember that when we refresh all queries, this could take a while, especially if the datasets are huge. Another very important fact to know is that when refreshing all queries, the sequence of query refresh is conducted in the correct order. When we choose to refresh queries individually and manually, you would need to do so in the precise order and at the accurate time.

If your data source is located in an external file, then only the last saved version of the source is loaded into Power Query. Even if the external file is open on the computer and you have recently edited the file, you would need to save the file to include recent edits in the refresh. If, however, the source data and the queries are located in the same workbook, any edits would be fused automatically, without them being saved before refreshing. Be careful when refreshing a query based on another query as not all the queries will be refreshed.

We can set the background refresh option to update automatically while we continue working. This method has its pros and cons. One pro would be that you can work while the refresh is happening, but while you are working, you may change a formula or recalculate or update Pivot tables, and these changes might not be included if the background refresh is not complete yet. Follow these steps to learn how to refresh:

1. To set the automatic background refresh, make sure you have Excel open with an active query and connection. Click on **Data | Refresh All Connection Properties....**
2. In the **Query Properties** dialog box, locate the **Refresh** control heading on the **Usage** tab.
3. Ensure that **Enable background refresh** is active. Then, choose a **Refresh** duration in **minutes**:

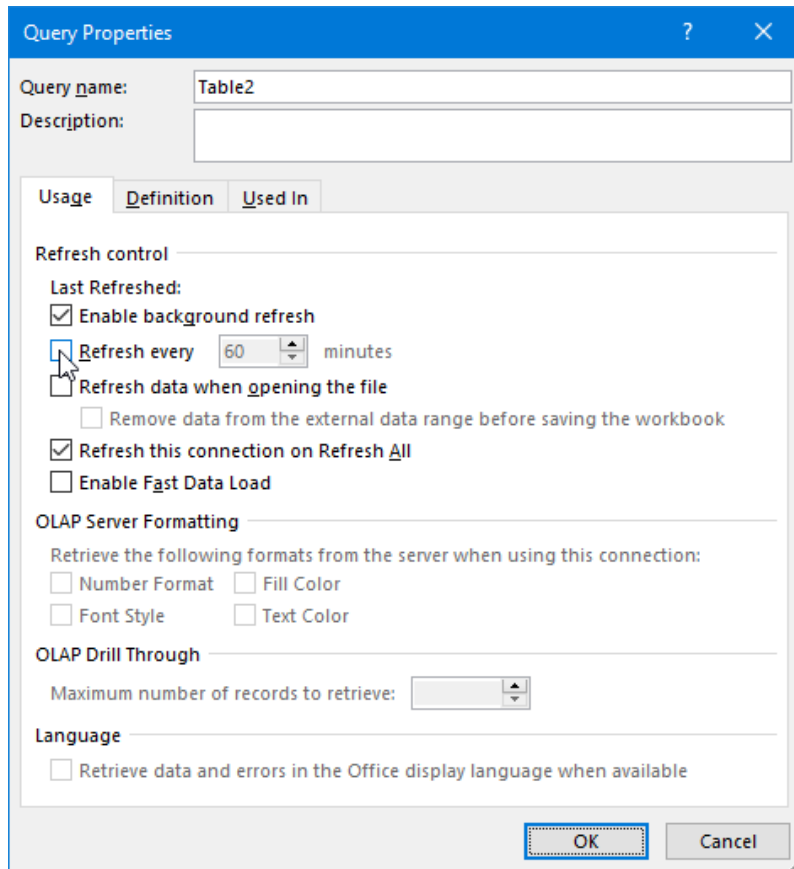


Figure 5.9 – Refresh options in Excel

4. Note that you can choose to refresh data when opening a file, as well as refresh the connection when using the **Refresh All** option.
5. Click on **OK** to confirm the changes you've made and return to the Excel workbook.

In this section, we have learned how data is refreshed and the various options related to refresh within the **Query Properties** dialog box. In the next section, we will concentrate on altering data using column and row tools.

Basic column and row tools

In this section, you will work with many column tools in order to remove columns you no longer require and split columns so that data is separated from one column into separate columns. You will be working the extract, merge, index, and conditional columns.

You will learn how to reshape table data so that the data is cleaned and ready for analysis in other applications, such as Power BI and Excel Power Pivot, by removing unwanted columns; removing top or bottom rows, should there be additional information you don't require; using the index column to aid analysis; creating conditional columns based on criteria you provide and applying column filtering using AND/OR conditions; using the single and multiple criteria filter; removing any duplicating rows or null values; working with the header row in a query; and splitting column values into separate delimiters. These options help to shape data so that it is presented in a format that aids reporting.

Removing columns

Removing columns in Power Query is extremely easy, and the process is very similar to removing a column in Excel. If a column adds no analysis purpose in a query, then remove it. The reason for this is that unnecessary data adds to the load when refreshing or working with the data. The following methods can be achieved using Power Query in Excel or Power BI:

1. We will use `WeatherData.xlsx` for this query.
2. There are two options to remove columns. The first is **Remove Columns**, while the second is **Remove Other Columns**. We can use the right-click method or the ribbon to perform these actions. It depends on what you select as to which option you wish to use. Use the **Remove Columns** option to remove the selected columns and use the **Remove Other Columns** option to remove all columns that are not selected.
3. Select the column or columns to remove. For this example, we will remove the column labeled **Out**.

4. Right-click and select **Remove**:

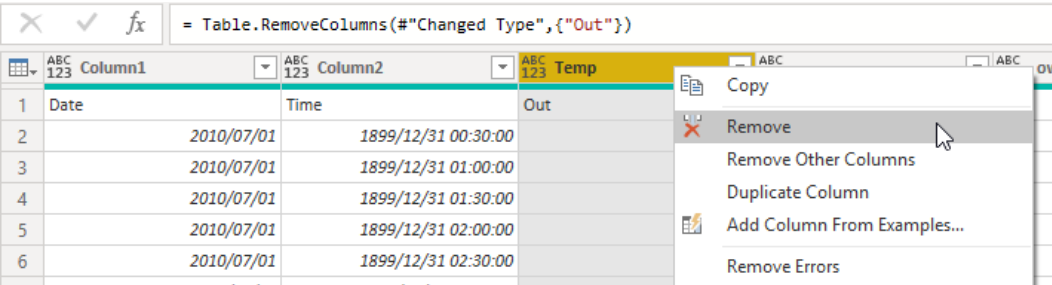


Figure 5.10 – Removing columns

5. The columns disappear from the query. Do not forget to save the query to update it.

Next, we will learn how to remove a top or bottom unwanted row to make the data appear crisp.

Removing top or bottom rows

One of the most common things to do when cleaning data is to remove unwanted top or bottom rows from table data. Let's continue with the WeatherData query from the previous example. There are a number of remove row options located on the **Home | Remove Rows** drop-down list in Power Query. Click to select **Remove Top Rows**. Specify the number of rows to remove (in this case, we will remove only the first row) from the top of the data table. Click on **OK** to confirm and save the query:

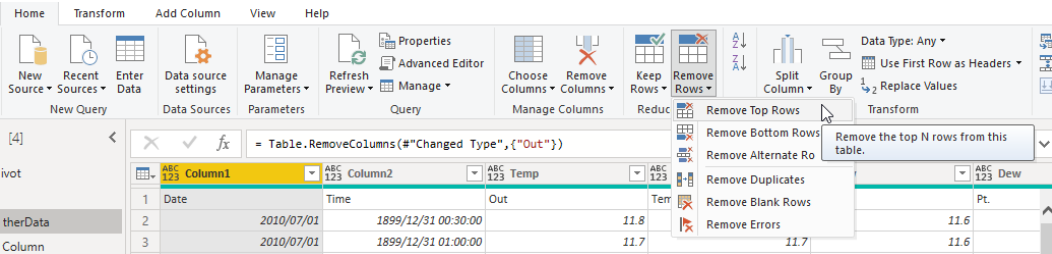


Figure 5.11 – Removing the top rows from a data table

Notice that you are also able to specify a pattern to remove alternative rows. You would specify the starting row, then the number of rows to remove, and lastly the number of rows to keep. Power Query will work its magic according to the pattern you provide.

Using the index column

We will create an index column to act as a row counter or temporary column so that it can aid data analysis. The index column will be a set of numbers that is used to count rows. For this example, we will use the `SSGProductsAll` query to create the index column:

1. Click to select the **SSGProductsAll** query (this query was created as part of the *Turning data with the unpivot and pivot tool* section in this chapter).
2. Navigate to **Add Column | Index Column | Custom...** to create an index column with a specified increment:

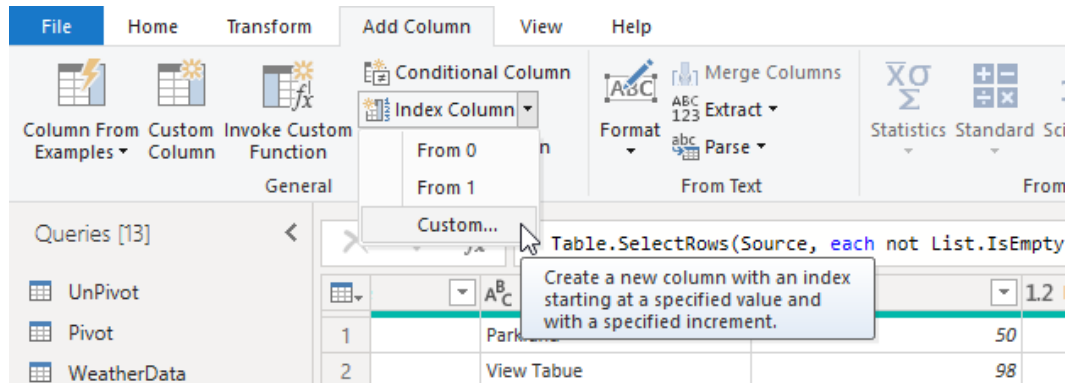


Figure 5.12 – Index column navigation

3. The **Index column** dialog box will populate, where you will enter the values to increment by choosing the starting point and the increment value for each subsequent number. For this example, we will start with 1 and increment by 1. Enter these values into the placeholders provided:

	Units	Price/Unit	Sales	Due Date	Index
1	50	11.6	580	2019/05/03	1
2	98	9.99	979.02	2019/05/03	2
3	74	10	740	2019/06/01	3
4	210	9.99	2097.9	2019/05/03	4
5	22	9.99	219.78	2019/05/01	5
6	89	9.99	889.11	2019/06/01	6
7	32	11	352	2019/06/01	7
8	15	16	240	2019/05/03	8
9	65	13.6	884	2019/05/01	9
10	75	12.55	941.25	2019/05/03	10
11	45	12.55	564.75	2019/05/01	11

Figure 5.13 – Index column created at the end of the table

4. Once the index column has been created, it will be visible at the end of the table. We would normally require the index to be at the beginning of the table. To move the index column so that it takes the role of the first column in the table, click on **Transform | Move | To Beginning**:
5. Save the query to update it.

With that, you have learned how to create an index to serve as a row counter as the first column of query data. Now, we will focus on creating conditional columns using the `if/then/else` statement, asking questions of our data so that we achieve the result we want.

Creating a conditional column with the `if...then...else` statement

The conditional column is a tool within Power Query that returns an automatic result based on meeting a certain set of conditions. Specifically, this is useful when requiring the `if...then...else` conditions and can be a complex tool that allows `else...if` clauses to be executed. We will run through the steps for creating a conditional column to display the sales team for each region matching each of the salespersons in column A of the source data. Let's get started:

1. Import the `SalesTeams.xlsx` data into Power Query using either Excel or Power BI:

	A
1	Sales Team West
2	Arthur Payne
3	David Header
4	Brian Donkin
5	James Bartha
6	Carol Harrinson
7	Sales Team North
8	Robert Wagner
9	Hillary Hunker
10	Kimberly Hepburn
11	George Houderson
12	Sales Team East
13	Ariel Saywers
14	Monica Essous
15	Sales Team South
16	Gerald Lafasne
17	Angie D'Abrose
18	Andrew Sales
19	Katherine Henderson
20	Gregory Duncan

Figure 5.14 – The source workbook data structure for `SalesTeams.xlsx`

2. Rename the query `SalesTeams`.
3. What we would like to do here is create a conditional column that lists the relevant sales team for each of the salespeople in column 1. At present, the data source lists the Sales Team region, then directly beneath that, it lists the salespeople in that region and continues in that fashion down the column. It depends on what you intend to do with the data, but in this case, we would like to end up with the Sales Team region in one column and the salesperson in the other. Let's run through the steps to complete this.
4. Make sure you have selected the **SalesTeams** query. Then, click on **Add Column | Add Conditional Column**.
5. In the dialog box that appears, set **New column name** to `Sales Teams`. We will now compose the `if... then... else` statement. If **Column1** begins with `Sales`, then the output must be in `Column1`; otherwise, place `Null` into the column. This is the first step and will create a new column with the headings for each of the sales teams and a null value under each. Click on **OK** to confirm this:

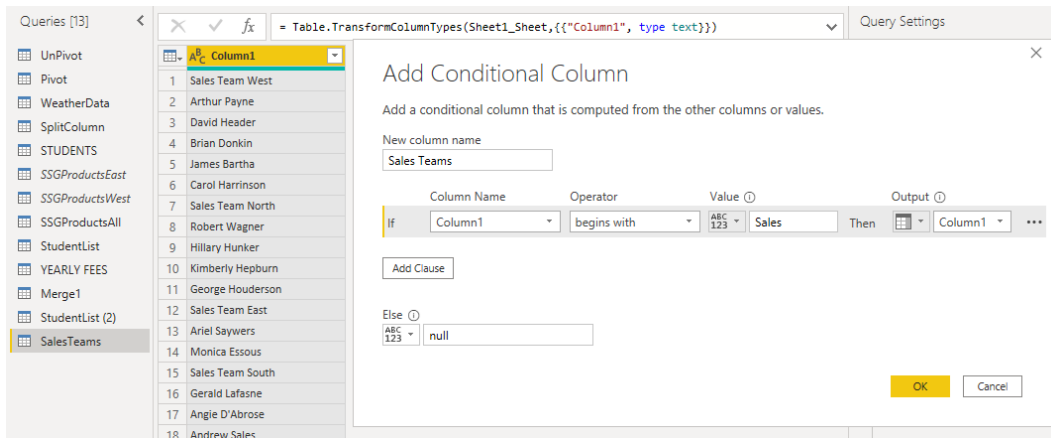


Figure 5.15 – If, then, else, conditional column criteria

6. Your result will look identical to the following:

	Column1	Sales Teams
1	Sales Team West	Sales Team West
2	Arthur Payne	null
3	David Header	null
4	Brian Donkin	null
5	James Bartha	null
6	Carol Harrinson	null
7	Sales Team North	Sales Team North
8	Robert Wagner	null
9	Hillary Hunker	null
10	Kimberly Hepburn	null
11	George Houderson	null
12	Sales Team East	Sales Team East
13	Ariel Saywers	null
14	Monica Essous	null
15	Sales Team South	Sales Team South
16	Gerald Lafasne	null
17	Angie D'Abrose	null
18	Andrew Sales	null
19	Katherine Henderson	null
20	Gregory Duncan	null

Figure 5.16 – If...then...else result

- The next step is to fill down the sales teams in the new conditional column so that each salesperson is placed in the correct team. Right-click on the **Sales Teams** column and choose **Fill | Down**.
- Lastly, we need to filter the rows in **Column1** to remove the words beginning with **Sales** so that only the salesperson's names appear in that column.
- Click on **Filter icon**, which is located to the right of the **Column1** header. The filter options will populate. Here, you will need to select **Text Filters | Does Not Begin With....**
- Locate the text **Keep Rows where 'Column1' does not begin with** and then enter the text **Sales** into the placeholder that reads **Enter or select a value**.

11. Click on the **OK** icon to confirm and see the result:

Queries [13] < = Table.SelectRows(#"Filled Down", each not Text.StartsWith([Column1])

	Column1	Sales Teams
1	Arthur Payne	Sales Team West
2	David Header	Sales Team West
3	Brian Donkin	Sales Team West
4	James Bartha	Sales Team West
5	Carol Harrinson	Sales Team West
6	Robert Wagner	Sales Team North
7	Hillary Hunker	Sales Team North
8	Kimberly Hepburn	Sales Team North
9	George Houderson	Sales Team North
10	Ariel Saywers	Sales Team East
11	Monica Essous	Sales Team East
12	Gerald Lafasne	Sales Team South
13	Angie D'Abrose	Sales Team South
14	Andrew Sales	Sales Team South
15	Katherine Henderson	Sales Team South
16	Gregory Duncan	Sales Team South

Figure 5.17 – The Conditional Column result

12. Rename **Column1** so that it says Salesperson, if required.

You are now able to create a conditional query using the **if...then...else** statement, thus achieving an automatic output based on certain criteria. The next section will focus on using **And/Or** conditions when filtering data.

Filtering data using the And/Or conditions

As we are already experienced with filtering and sorting data within Excel, we know that there are many methods to achieve certain results. The same applies to filtering data using Power Query. We will look at two types of filtering situations, namely, **And/Or**. Power Query will only load 1,000 distinct records at a time into the filter list. You will see a notification regarding this, stating that the list of values in the filter list is incomplete. Always click the **Load more** link to load the next 1,000 values, and so on. To try this filter, perform the following steps:

1. Import the source data from `SSGFilter.xlsx` into Power Query.
2. Each column header within the query has an active filter icon. This filter icon is displayed as a drop-down arrow list that will populate when you click the icon. This is exactly how you would use this feature in Excel.

3. Click on the filter icon alongside the **Division** column heading. From the filter list provided, choose **Text Filters | Equals | Advanced**.
4. We will now use the **Advanced** filter for the rows that are from the **Soningdale** division **And** have a **GROSS PAY** greater than **213**. Select **DIVISION** from the **Column** category, set **equals** to **Operator**, and then click the drop-down arrow to set **Value** to **Soningdale**. Make sure to check that the **And** condition is selected and then select **GROSS PAY** for **Column**. **Operator** should reflect **is greater than** and **Value** should be **213**:

Filter Rows

Apply one or more filter conditions to the rows in this table.

☐ Basic ☒ Advanced

Keep rows where

And/Or	Column	Operator	Value	
	DEPT	equals	Shewe	
Or	DEPT	equals	Mankay Falls	...

Add Clause

OK Cancel

Figure 5.18 – Filtering rows using the And condition

5. Click on **OK** to see the results of the filter. The filter icon will now appear to the right of the **DIVISION** column, indicating that a filter has been applied.
6. Let's add the **Or** condition to this filter to display the rows where **DEPT** either equals **Shewe OR Mankay Falls**.

7. Click on **OK** to view the results:

✕

Filter Rows

Apply one or more filter conditions to the rows in this table.

☐ Basic ☒ Advanced

Keep rows where

And/Or	Column	Operator	Value
	DIVISION	equals	Soningdale
And	GROSS PAY	is greater th...	213

Add Clause

OK Cancel

Figure 5.19 – Advanced OR filter

8. Your filter should now display 11 rows that meet the criteria entered:

Queries [1] ✕ ✓ fx = Table.SelectRows("#Filtered Rows1", each [DIVISION] = "Soningdale" and [GROSS PAY] > 213)

	123 CODE	A ⁰ C FIRST	A ⁰ C SURNAME	A ⁰ C EMP NO	A ⁰ C DIVISION	A ⁰ C DEPT	DATE of HIRE	1.2 HRS	1.2
1	4	Chris	Hume	SUN59	Soningdale	Shewe	1988/05/12 00:00...	40	
2	8	Kristen	DeVinney	SUN45	Soningdale	Shewe	1987/06/05 00:00...	35	
3	9	Robert	Murray	SUN47	Soningdale	Mankay Falls	1987/06/10 00:00...	40	
4	11	Sean	Willis	SUN09	Soningdale	Mankay Falls	1985/07/05 00:00...	35.5	
5	18	Paul	Hoffman	SUN57	Soningdale	Shewe	1987/12/19 00:00...	40	
6	33	Laura	ReaMILn	SUN77	Soningdale	Mankay Falls	1990/08/12 00:00...	35	
7	45	Kyle	Earnhart	SUN16	Soningdale	Shewe	1984/10/08 00:00...	40	
8	57	Bill	Wheeler	SUN05	Soningdale	Mankay Falls	1981/08/14 00:00...	38	
9	58	Todd	MPKLters	SUN69	Soningdale	Shewe	1989/11/23 00:00...	40	
10	68	William	Abel	SUN66	Soningdale	Mankay Falls	1989/10/14 00:00...	40	
11	73	Bradley	Howard	SUN12	Soningdale	Mankay Falls	1984/02/14 00:00...	40	

Figure 5.20: The 11 rows with criteria satisfied

9. **Close & Load** your query result back into Excel.

You have now mastered filtering data using **AND/OR** conditions. The next section will concentrate on extracting data from columns using single criteria.

Creating single-criteria filters

In this section, we will learn how to create a single-criterion filter to extract data from a column using Power Query. Let's get started:

1. Continue with the Excel worksheet from the previous exercise. Make sure you have selected the Filter worksheet as the source data.
2. In cell **L4**, enter the column heading **Division**. Then, directly below the heading, enter the text **Soningdale**. These two cells will form the filter for our example:

	J	K	L
	GROSS PAY		Division
50	£ 860.00		Soningdale

Figure 5.21 – Table criteria for filtering from the worksheet

3. Firstly, load the source data into Power Query and name the table **SSGFilter**. Click on **Close & Load**. Name the worksheet SSGFilter.
4. Now, load the filter criteria and heading from the worksheet in cells **L4:L5** into Power Query.
5. To filter only for the Soningdale division at this point, we will need to drill down to convert the table into a singular value to ensure that it only filters Soningdale. Right-click on the text **Soningdale** and select **Drill Down**:

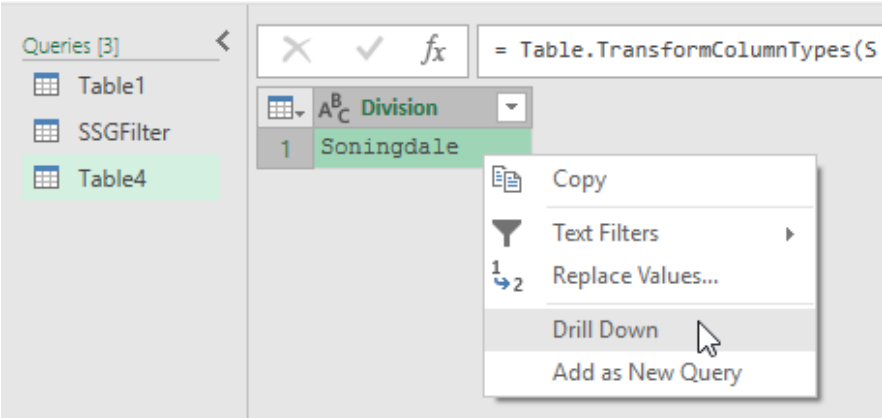


Figure 5.22 – Drilling down to a singular value for the filter

6. The first row of the query has now been converted into the singular value for Soningdale. Rename the **Table4** query Filter:

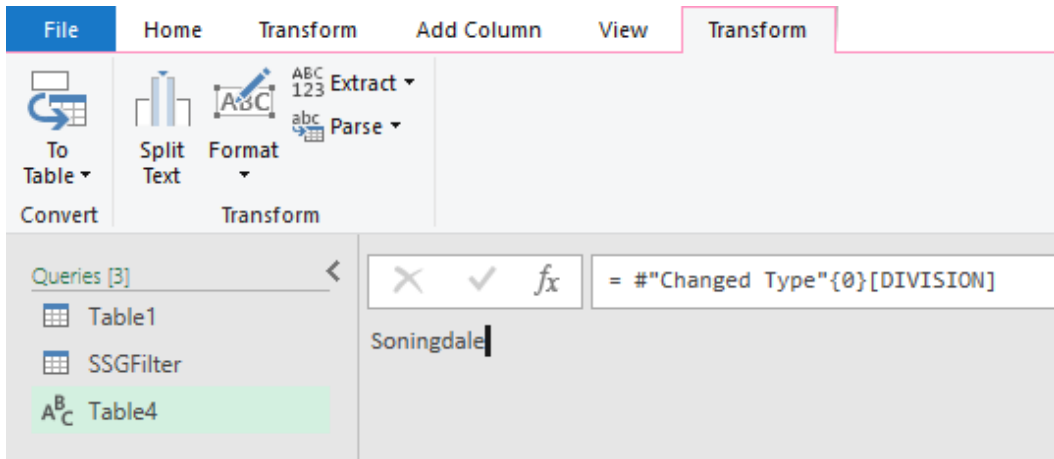


Figure 5.23 – Drill Down feature

7. To connect the source data query called **SSGFilter** and the filter criteria called **Filter**, we need to perform two steps. The first is to navigate back to the **SSGFilter** query and create a filter on the **Division** column. Let's filter by **Munerton**. The query has been updated to reflect only **Munerton**:

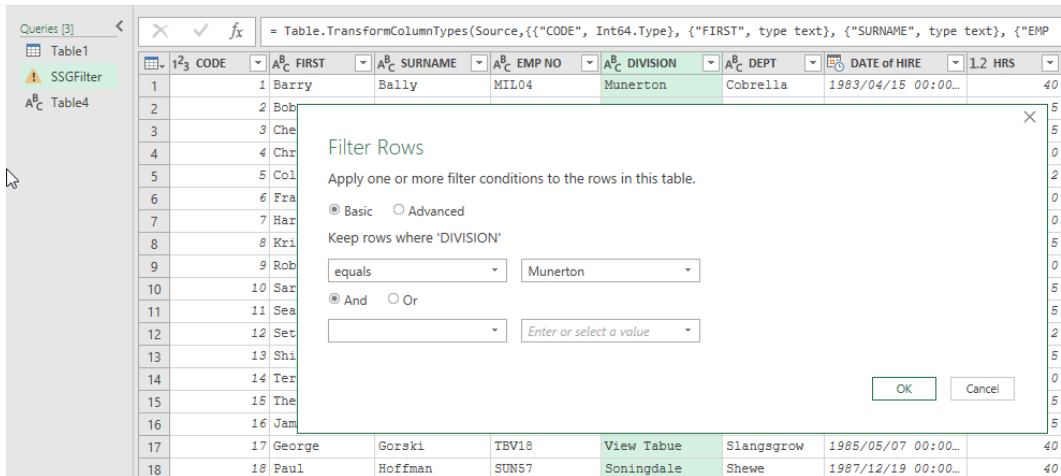


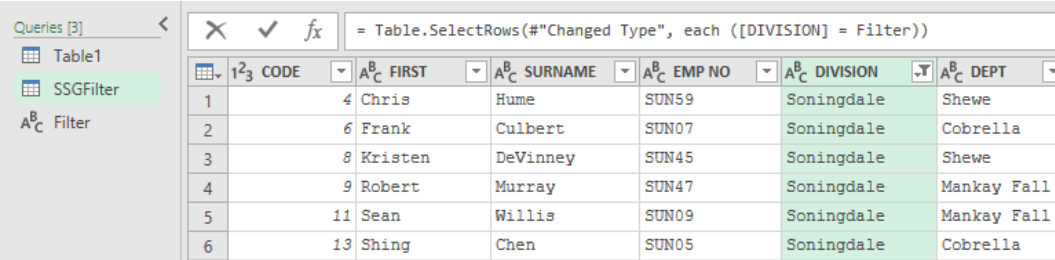
Figure 5.24 – Filter by division

8. Notice the results in the **Division** column.

- Secondly, look at the formula bar, where you will notice that the last part of the M code refers to the filter "**each ([DIVISION] = "Munerton")**". Replace the text **Munerton** at the end of the formula bar to reflect the filter criteria query you previously created. Type **Filter** in place of **Munerton**. Now, you will see that the **Division** column has updated to reflect our filter query criteria of **Soningdale**. The formula bar will now read as follows:

= Table.SelectRows("#Changed Type", each ([DIVISION] = Filter))

Note that we removed the quotation marks around the text as we are now referring to a query:



	1 ² 3	CODE	A ^B _C FIRST	A ^B _C SURNAME	A ^B _C EMP NO	A ^B _C DIVISION	A ^B _C DEPT
1		4	Chris	Hume	SUN59	Soningdale	Shewe
2		6	Frank	Culbert	SUN07	Soningdale	Cobrella
3		8	Kristen	DeVinney	SUN45	Soningdale	Shewe
4		9	Robert	Murray	SUN47	Soningdale	Mankay Fall
5		11	Sean	Willis	SUN09	Soningdale	Mankay Fall
6		13	Shing	Chen	SUN05	Soningdale	Cobrella

Figure 5.25 – Formula bar showing the DIVISION column linked to the Filter query criterion

- Close & Load** the queries in Excel, placing both the **Filter** query and the **SSGFilter** query next to each other on a worksheet named **SSGFilter**. The **Filter** query should be in cells **L4:L5** and the **SSGFilter** table should be positioned in cell **A4**.
- To test the filter, type **View Tabue** into cell L5, then right-click over the **SSGFilter** table and choose **Refresh** from the drop-down list provided. The data table's **Division** column should update to reflect only the rows that contain **View Tabue**.

You are now confident in creating a single-criteria filter. This will allow any parameter to be entered as a search criterion. In the next section, we will go one step further and look at how to set up multiple-criterion searches.

Creating dynamic multiple-criterion filters

In this section, we will learn how to create a dynamic filter to extract data from multiple columns using more than one filter condition within it using Power Query. Let's get started:


- Continue with the Excel worksheet from the previous exercise. Make sure you have selected the **Filter** worksheet as the source data.
- In cell **L9:L11**, type in **DEPT** as the heading, with **Mankay Falls** and **Cobrella** as the filter criteria directly below the heading.

3. Click on **Mankay Falls** and use **Get & Transform** to import the data table into Power Query.
4. The next step is to duplicate the **FilterData** query as we need to use this as the source data for our multiple-criterion filter. We use the **Duplicate** option when we want to use data from another query to create a new query with different code. Right-click on the **FilterData** query and select **Duplicate** from the drop-down list provided. Remove the last step, **Filtered Rows**, from the **APPLIED STEPS** pane.
5. To set the query so that it matches either **Mankay Falls** or **Corbella**, we will need to join the two queries together using the **Merge Queries** tool using an outer join. The result will return only those two departments and remove the rest of the data. Click on the **FilterData(2)** query, and then visit **Home | Merge Queries**.
6. In the **Merge** dialog box, select the **DEPT** column from **FilterData(2)** and then, from the drop-down list, select the **Table5** query. We're doing this as we would like to show only the matching rows from both tables:

✕


Merge

Select a table and matching columns to create a merged table.

FilterData (2) 

CODE	FIRST	SURNAME	EMP NO	DIVISION	DEPT	DATE of HIRE	HRS	HOURLY RATE	GF
1	Barry	Bally	MIL04	Munerton	Cobrella	30421	40	21.5	
2	Bob	Ambrose	MIL14	Munerton	Mankay Falls	31072	35.5	12.5	
7	Harry	Swayne	MIL25	Munerton	Cobrella	33237	40	21.5	
10	Sara	Kling	MIL29	Munerton	Mankay Falls	31770	35.5	12.5	
20	TBVrol	Hill	MIL18	Munerton	Mankay Falls	31614	35.5	12.5	

< >

Table5 

DEPT
Mankay Falls
Cobrella

Join Kind

Inner (only matching rows) ▾

✓ The selection matches 18 of 27 rows from the first table, and 2 of 2 rows f...

OK Cancel

Figure 5.26 – Inner join matching rows between two tables

- Click on **OK** to see the result. As indicated at the bottom of the **Merge** dialog box, there are 18 of 27 rows matching from the first table and 2 of 2 rows from **Table5**:

	EMP NO	DIVISION	DEPT	DATE of HIRE	HRS	HOURLY RATE	GROSS PAY	Table5
1	MIL04	Munerton	Cobrella	30421	40	21.5	860	Table
2	MIL25	Munerton	Cobrella	33237	40	21.5	860	Table
3	MIL14	Munerton	Mankay Falls	31072	35.5	12.5	443.75	Table
4	MIL29	Munerton	Mankay Falls	31770	35.5	12.5	443.75	Table
5	MIL18	Munerton	Mankay Falls	31614	35.5	12.5	443.75	Table
6	MIL04	Munerton	Mankay Falls	29653	40	19.5	780	Table
7	MIL15	Munerton	Mankay Falls	31359	40	19.5	780	Table
8	MIL32	Munerton	Mankay Falls	32106	35.5	12.5	443.75	Table
9	MIL12	Munerton	Cobrella	32029	29.5	6.5	191.75	Table

Figure 5.27 – Inner join result

- The **APPLIED STEPS** pane displays the last step as **Merged Queries**. Delete the **Table5** column, which was added to the data after the two tables were merged. Select the column, and then press *Delete* on the keyboard or **right-click | Delete** with the mouse.
- To test the multiple query, change the **DEPT** criterion to **Shewe** and **Slangsgrow**. Right-click over the table data and select **Refresh** to see the result:

	CODE	FIRST	SURNAME	EMP NO	DIVISION	DEPT	DATE of HIRE	HRS	HOURLY RATE	GROSS PAY	DEPT
2	25	Sung	Kim	MIL49	Munerton	Slangsgrow	32827	40	15.5	620	Slangsgrow
3	34	Brian	Smith	MIL40	Munerton	Shewe	32452	40	19.5	780	Shewe
4	42	Bill	Simpson	MIL07	Munerton	Shewe	29963	40	19.5	780	
5	63	Jerry	McDonald	MIL08	Munerton	Slangsgrow	30139	40	15.5	620	
6	67	Paula	Robinson	MIL23	Munerton	Slangsgrow	32118	29.5	6.5	191.75	
7	70	Kim	Smith	MIL54	Munerton	Shewe	32839	42	24	1008	
8	72	John	JPKLobs	MIL27	Munerton	Slangsgrow	31689	40	6.5	260	
9	85	Helen	Stewart	MIL57	Munerton	Slangsgrow	32996	42	15.5	651	
10	92	Sam	Whitney	MIL09	Munerton	Shewe	30648	40	7.22	288.8	

Figure 5.28 – Multiple query result for Slangsgrow and Shewe

You should now have the knowledge to create a multiple-criterion filter to manipulate data output. In the next section, we will look at how to remove duplicate rows of data from a query.

Removing duplicate rows

When we try to remove duplicate data from a table, we could possibly make a mistake without realizing that we may be deleting the incorrect rows. This is especially relevant when removing duplicate rows manually using the keyboard. Power Query includes the **Remove Duplicates** option as one of its transform tools. Let's take a look:

1. Load the `SalesData.xlsx` worksheet into the query editor using Power BI. If the **Date** column is imported as text, use the locale to convert it into date format (right-click on the column, and then choose **Change Type | Using Locale**). Locale is perfect when dealing with text files that contain dates and number formats from different countries. You learned about this in *Chapter 3, Introduction to the Power Query Interface*:

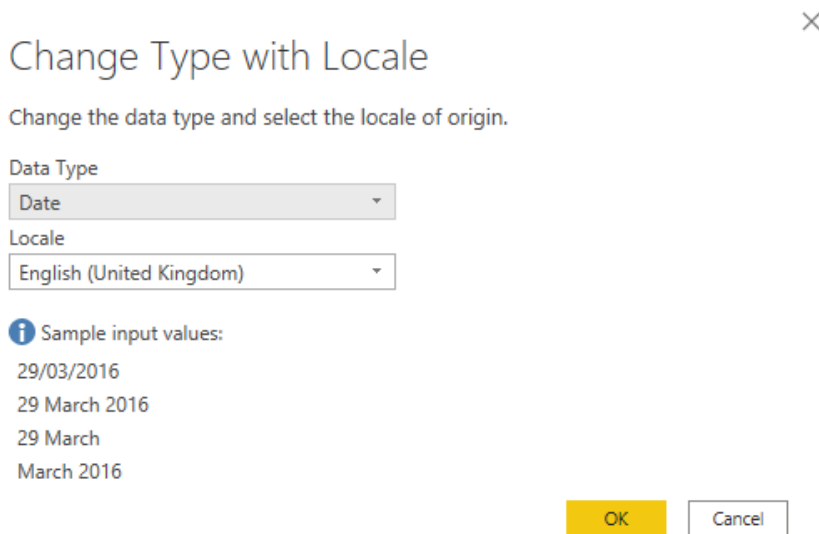


Figure 5.29 – Changing Locale

- We have noticed that there are duplicate sets of data in this sales report. For instance, rows three and four of the table are identical. To remove duplicates, click on **Home | Remove Rows | Remove Duplicates**:

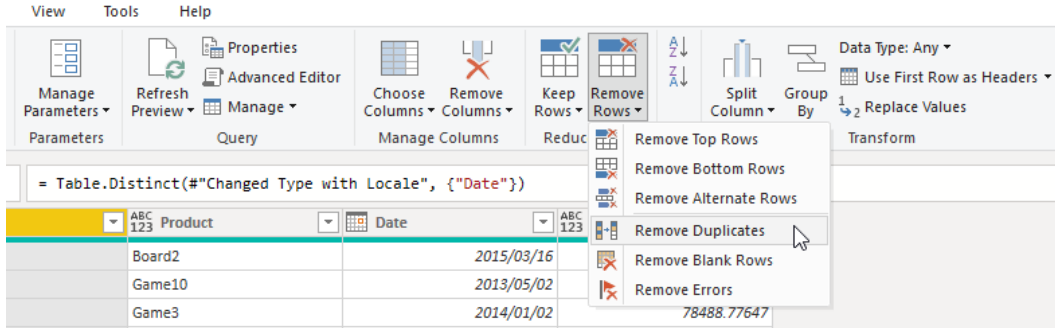


Figure 5.30 – Removing duplicate rows

- We started off with 57 rows of data. Now that the duplicates have been removed, there are 54 rows left. Note that we did not select any columns to perform this action. You need to be very careful as to which columns you have selected prior to removing duplicates. For instance, if we selected the **Region** and **Product** columns, and then selected **Remove Duplicates**, the result would be different. Experiment with this now on the **SalesData** dataset.

Now that you know how to remove duplicate sets of data from a query, let's look at how to replace null values in a query.

Replacing null values

To replace values in Power Query Editor, you will need to follow the same procedure that you followed when replacing values using Excel:

- Duplicate the **SalesTeams** query and rename it **SalesTeamsNull1**. Remove the **APPLIED STEPS** window until you see the raw data with null values. Click on a null value to select it.

2. Click on **Home** | **Replace Values**:

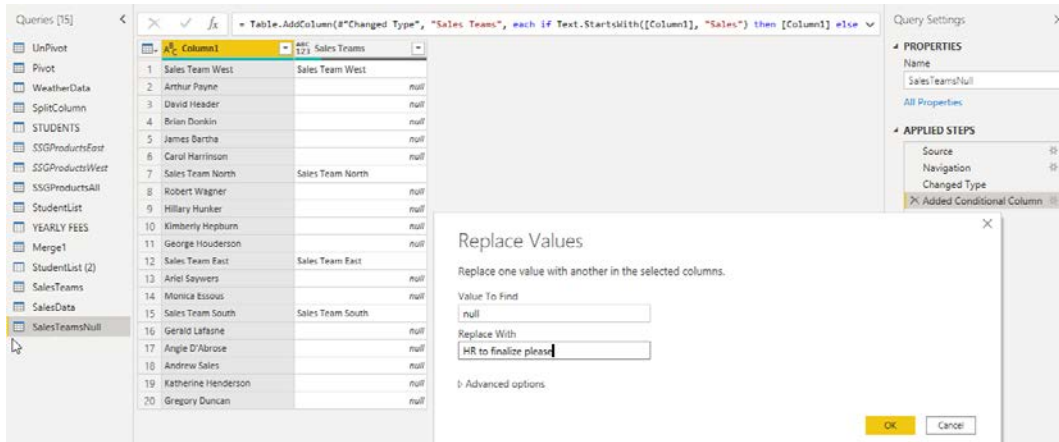


Figure 5.31 – Replacing null values

Null is already evident in the **Value To Find** placeholder as we selected the first null value prior to clicking on the **Home** tab. Enter HR to finalize it in the **Replace With** placeholder. Notice that **Advanced options** are also available at the bottom of the dialog box so that you can select matching options and replace them with special characters.

3. Click on **OK** to perform the replace action and view the results.

Here, we used the replace command to find values in a query and then replaced these values with other values. You will now work with the header row options available in Power Query.

Working with the header row

In Power Query, you have the option to **Use the First Rows as Headers** or **Use Headers as First Row**. This should not happen often as you will typically import the data identifying the first row as column headers from the start. However, some data sources will require this step when you enter Power Query. Let's take a look:

1. Let's import `SalesData1.csv` into Power Query for this example.
2. After connecting the data source, you will see that the headers that are automatically assigned are the defaults, namely, **Column1**, **Column2**, **Column3**, and **Column4**, and that the actual headers are situated in **Row1** of the table.

3. To move the contents of **Row1** so that you can display it as the header row of the table, click **Home | Use First Row as Headers**.
4. The contents of **Row1** are now displayed as the table headers.

In the next section, we will look at how to split columns in order to separate data from a single column into multiple columns to aid the querying of data.

Splitting columns

You may ask yourself why we would want to use split columns in Power Query when we have tools such as **Flash Fill**, **Text to Columns**, and formulas available in Excel. When splitting columns in Power Query, as opposed to using the tools within Excel itself, you are able to repeat the process and refresh it. Yes, you can refresh data when using the formula options in Excel, but the complexity of dealing with varying widths of, for example, text such as cities in a column is going to cause further problems.

The **Split Column** feature is located on the **Home** tab. Alternatively, you can reach it by right-clicking over a column:

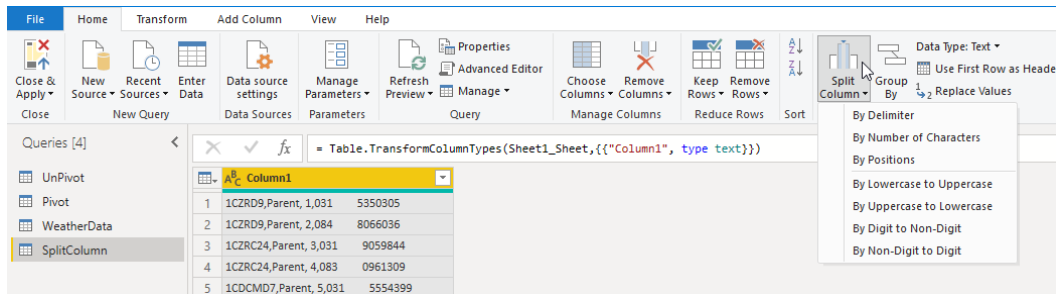


Figure 5.32 – Power Query split columns

Now that you have had an introduction to splitting columns, let's put this into practice:

1. Load the data from the workbook named `SplitColumns.xlsx` into Power Query.
2. Select **Column1** and go to **Home | Split Column**.
3. There are a number of choices to make from the drop-down list provided. For this example, we will use the **By Delimiter** option, and then choose to split the column by **Comma**:

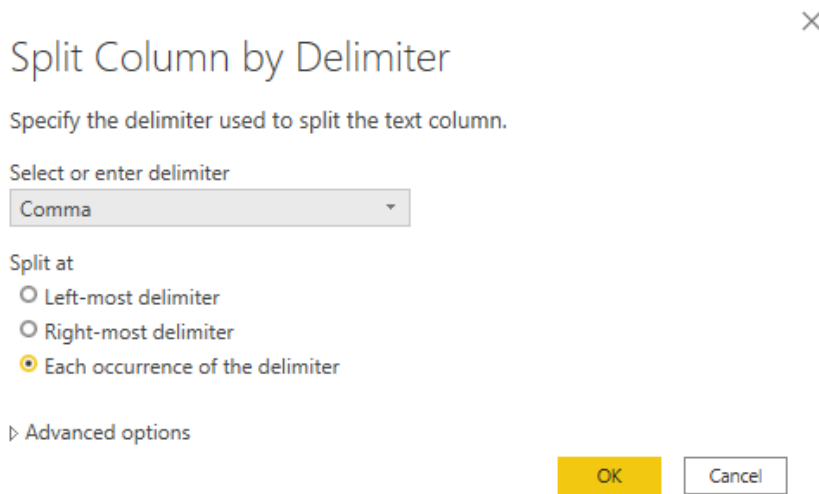


Figure 5.33 – Splitting columns by the comma delimiter

4. Make sure that the **Split at** option is set to **Each occurrence of the delimiter**. If we do not choose this option, then only the first part of the data, before the comma, will be split into a separate column. Because the data has three sets of commas separating data, we need to make sure that each occurrence of the comma is identified.
5. Click on **OK** to view the results:

	A ^B _C Column1.1	A ^B _C Column1.2	1 ² ₃ Column1.3	A ^B _C Column1.4
1	1CZRD9	Parent		1 031 5350305
2	1CZRD9	Parent		2 084 8066036
3	1CZRC24	Parent		3 031 9059844
4	1CZRC24	Parent		4 083 0961309
5	1CDCMD7	Parent		5 031 5554399
6	1CDCMD7	Parent		6 083 8954814

Figure 5.34 – Result of the comma split

6. Now, we can rename the columns to their appropriate headings and delete **Column1.2** as we do not need the parent field anymore; it will become the heading for **Column1.3**. There is one more thing we need to do: **Column1.4** needs to be split as well as it contains a code and a number. There are spaces in between the code and number that we will remove by splitting the columns by position. Click to select the last column and then choose **Split Column | By Positions**.

- Power Query will look at the column and assume the positions according to the data in the column. In this case, the assumption is correct. Spend some time investigating positions as this tool is very useful:

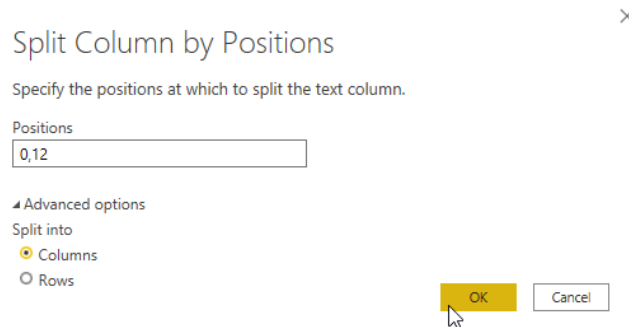


Figure 5.35 – Split Column by Positions

- Click on **OK** to see the changes. There are now two columns, one for the code and the other for the number. You will notice that the 0s were dropped from the first column data. This is because the column data type was altered automatically from text to number. If you prefer to display the 0s at this point, simply remove the last step, **Changed Type1**, from **APPLIED STEPS**.

When using **Split Column by Delimiter**, you need to be aware that when refreshing the query, the new data may not automatically appear due to a setting in **Split Column by Delimiter | Advanced options**, as shown here:

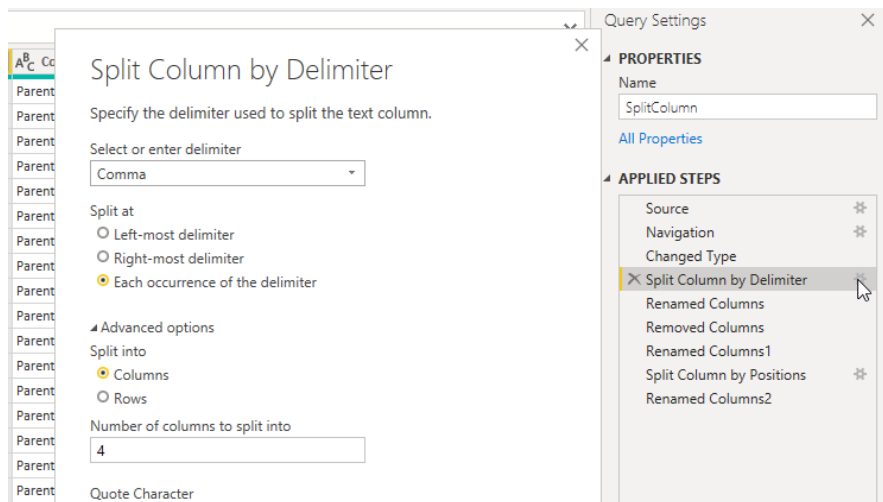


Figure 5.36 – Clicking the gear icon to change the Advanced options column setting

Once you have created the query, click on the gear icon for the **Split Column by Delimiter** step in the **APPLIED STEPS** pane to view the criteria that Power Query automatically assumed on creation. If we look at the underlying table from which we created the split column query, the data was separated into four parts using the comma delimiter, and therefore split according to four columns. When the source data is updated via the **Refresh** option, it may not include the now new fifth piece of data. To correct this, locate **Advanced options** and set the **Number of columns to split into** setting to the new number of columns according to the source data. It will then refresh the data automatically.

Now that we've learned how to split columns, we will look at the merge and append tools available in Power Query.

Merging and appending tools

In this section, we will understand how to combine, merge, and extract data from sources within Power Query. These are extremely powerful and useful tools that can be used to transform data.

Merging columns using combine

We can merge data sources or data tables whose structures are totally different from each other. The only thing we require is a relationship between the sources. This is normally a common field that relates one field in the first data source to another field in the second data source. This concept is exactly the same as working with primary keys, relationships, and joins in Access 2019.

If we have a list of student data in one table and data related to fees that have been paid per student in another table, we can merge them to flatten the data into one table. The **Merge** tool offers you the ability to select a common field and create a relationship between the two tables so that data can be combined into a single flattened data table. Let's get started:

1. Create two queries from the **STUDENTS** and **YEARLY FEES** data tables, which are located in the `SSGSchoolAdmin.accdb` data source.
2. Rename the `Students2` query `StudentList`. Both tables include a common, related field called **Code**. We will use this field to create a link between the two tables.

Click on the **StudentList** query and then go to **Home | Merge Queries | Merge Queries as New**:

1. In the **Merge** dialog box that populates, select **Code** from the **STUDENTS** table, which will be the column you will use as the joining field. From the second drop-down list, select the **YEARLY FEES** table, and then click on the **Code** column to identify it as the matching join. The joining fields do not have to have the same column heading. Should you have more than one matching column, you simply select them in the order you wish to perform the merge by using the *Ctrl* + click method. A value in sequence is then placed next to each matching column. You are also able to select a **Join Kind** from the drop-down list provided. The most common is **Left Outer (all from first, matching from second)**, where it will match all the fields from the first table and match them from the second table:

Merge

Select tables and matching columns to create a merged table.

StudentList

Code	Surname	Name	HouseType	Gender	Grade	Class	DOB	EnrolDate
1005	MARTHA	Mary	Ruby	Female	11	r	2004/08/08 00:00:00	1999/04/26 00:00:00
1341	WESSELS	Lee	Ruby	Female	8	P	2008/12/25 00:00:00	2000/01/01 00:00:00
164	VOSLOO	Natalie	Sapphire	Female	9	r	2009/04/12 00:00:00	2000/01/01 00:00:00
412	CARNEY	Wesley	Emerald	Male	12	e	2003/11/27 00:00:00	2000/01/01 00:00:00
427	MEAKER	Gregory	Sapphire	Male	12	e	2003/10/20 00:00:00	2001/02/10 00:00:00

YEARLY FEES

CODE	YEARLY FEE TOTAL	AMT_RECEIVED	DATE
1005	13200	6000	2004/06/28 00:00:00
895	13200	12000	2004/06/30 00:00:00
427	13200	7000	2004/06/25 00:00:00
429	13200	12000	2004/06/30 00:00:00
412	13200	1000	2004/06/30 00:00:00

Join Kind

Left Outer (all from first, matching from second)

Left Outer (all from first, matching from second)

Right Outer (all from second, matching from first)

Full Outer (all rows from both)

Inner (only matching rows)

Left Anti (rows only in first)

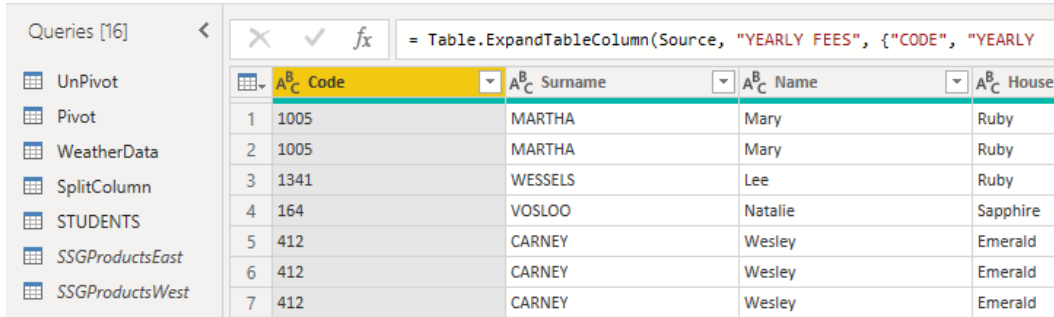
Right Anti (rows only in second)

OK

Cancel

Figure 5.37 – Merge by matching columns

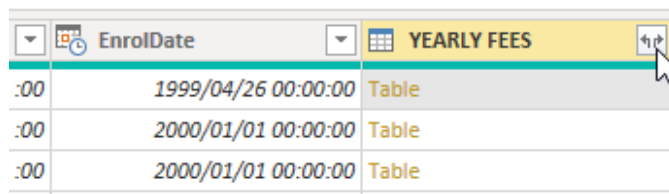
2. Click on **OK** to confirm the merge.
3. A new query called **Merge1** will be created, along with the result. If you scroll to the very right of the table, you will notice that the **YEARLY FEES** column has been included. This column is called a **structured column** as it has a structure (table) in every single cell that, when we click on the blank area (not the word *Table*) of the cell, will display a further table. So, this shows that **Student 1005** has made two payments toward **YEARLY FEE TOTAL**:



	A ^B _C Code	A ^B _C Surname	A ^B _C Name	A ^B _C House
1	1005	MARTHA	Mary	Ruby
2	1005	MARTHA	Mary	Ruby
3	1341	WESSELS	Lee	Ruby
4	164	VOSLOO	Natalie	Sapphire
5	412	CARNEY	Wesley	Emerald
6	412	CARNEY	Wesley	Emerald
7	412	CARNEY	Wesley	Emerald

Figure 5.38 – Student 1005 displaying two payment entries

4. We need to go one step further by combining the data to flatten the structure, as Power BI will not understand the concept of a table within other tables. Click on the **Combine Files** icon located to the right of the **YEARLY FEES** column heading. The **Combine Files** option is also available from the **Combine** group, which is located on the **Home** tab ribbon:



	EnrolDate	YEARLY FEES
:00	1999/04/26 00:00:00	Table
:00	2000/01/01 00:00:00	Table
:00	2000/01/01 00:00:00	Table

Figure 5.39 – Combine icon to flatten the structure of a table within tables

- Click to select which fields you would like to include in the combine process:

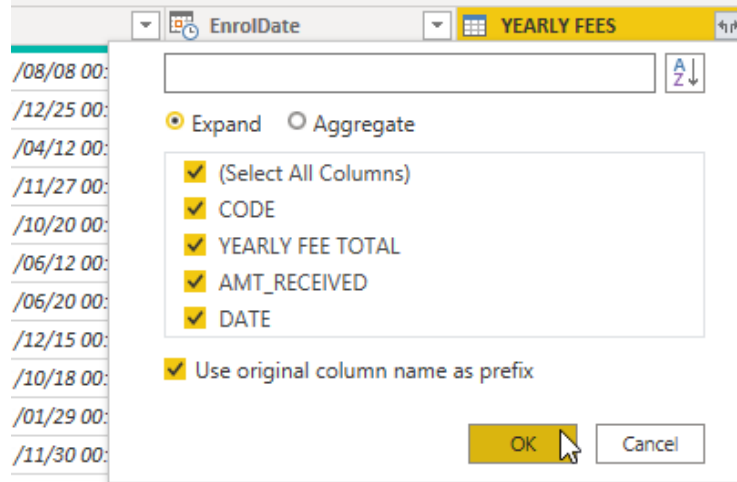


Figure 5.40 – Fields to be added for combining

For this example, we will use all the fields. Click on **OK** to confirm this.

- The tables are combined into a single table, ready to be analyzed in Excel or Power BI.

With that, you have learned how to use the **Combine** tool to merge tables together using a primary key. In the next section, we will learn how to merge multiple columns into one column.

Merging text and values into one column

When using this tool, it is important to note that the order in which you select your columns is important. For instance, if your mouse's focus is on a particular column in the query, that column will appear first in the merge result. Merging multiple columns into one column, separated by a delimiter, will cause the source columns to be lost. Always use the *Shift* + click method to highlight columns from one point to another as they will then be merged in that order, unless you are purposefully selecting random columns to define the order. Let's see an example of this:

- Use a copy of the **StudentList** query for this example. Once you have created a copy of **StudentList**, you will see that the query name has changed to **StudentList2**.
- To combine certain columns into one single column, we need to select the columns to merge. We will merge the **Grade** and **Class** columns. If we select the class column before the grade column, the display order will be incorrect.

3. Select the **Grade** columns, and then the **Class** columns. Click to select **Transform | Merge Columns**:

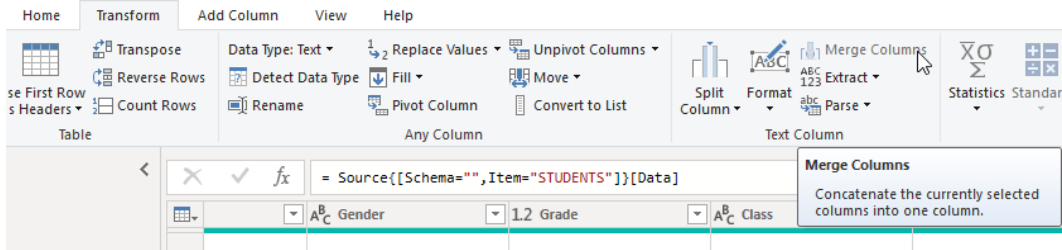


Figure 5.41 – Using the Merge Columns tool

4. In the dialog box provided, choose the **Space** separator and give the new column a name if you wish. We can change this after the merge if we need to. The default name will be present in the dialog box, namely **Class.1**:

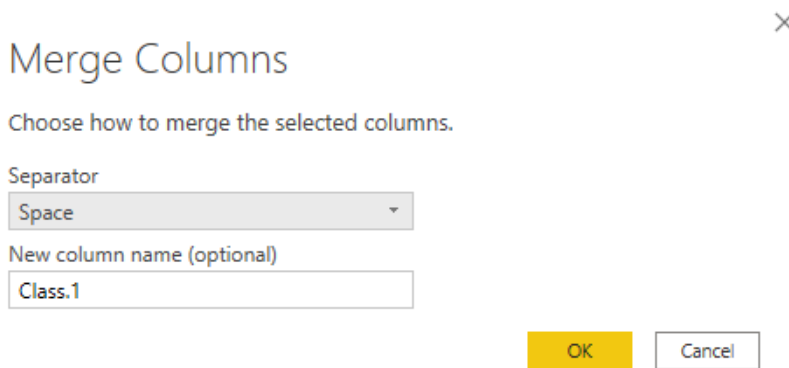


Figure 5.42 – Merge Columns dialog box showing the Space separator

5. The columns are merged together with a space separator in between the grade and class data.

In this section, we learned how to merge selected columns in a query and how to use the Space separator to produce a new column within a query. In the next section, we will learn how to combine two sets of data sources in a single query.

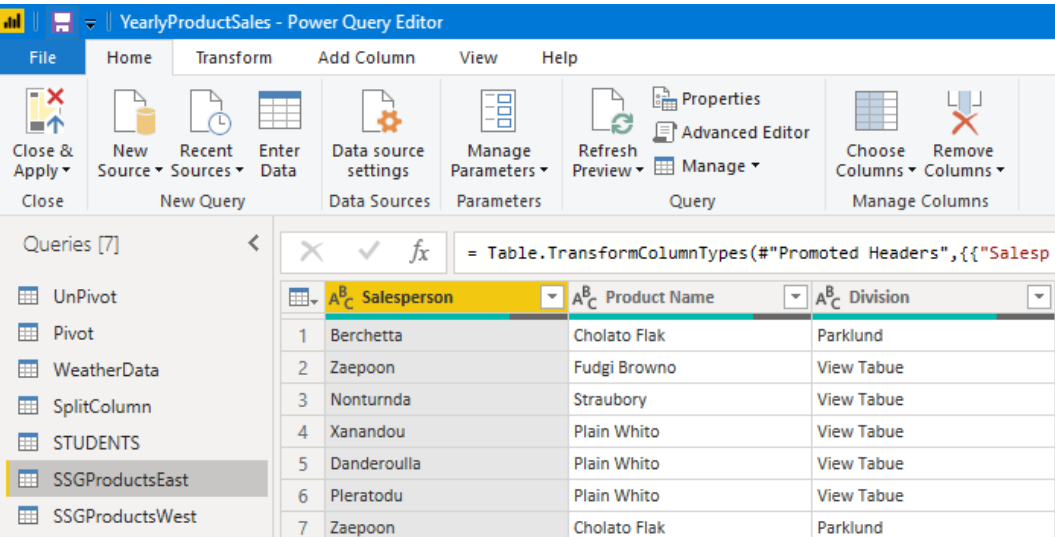
Appending (combining) tables

In this section, we will learn how to combine two data tables together. This can be applied to two different data sources or different data tables. It is also referred to as a union and is just like how we would combine using relational databases or SQL. This tool is great for combining, for instance, multiple student course lists into one dataset. The data will need to meet the following guidelines:

- The structure should be identical, as well as the column headings.
- You need to have at least one column in a table that matches another column in a separate table/worksheet.
- There can be foreign columns in either table, but the best practice is to make sure that the majority of the columns are the same.

To see how this works, we have to do a bit of setup.

Create two separate queries in Power Query from the worksheets in the `ChcolatoFlakSalesNW.xlsx` workbook. The following screenshot displays the two queries, **SSGProductsEast** and **SSGProductsWest**, that have been imported into Power Query. These queries contain data for our company in the East and West. The tables are similar with regard to column headings, except the East has an additional column called **Division**:



	Salesperson	Product Name	Division
1	Berchetta	Cholato Flak	Parklund
2	Zaepoon	Fudgi Browno	View Tabue
3	Nonturnda	Straubory	View Tabue
4	Xanandou	Plain White	View Tabue
5	Danderoulla	Plain White	View Tabue
6	Pieratodu	Plain White	View Tabue
7	Zaepoon	Cholato Flak	Parklund

Figure 5.43 – Query results in Power Query

Notice that both tables have two extra columns at the end. We can remove these two tables to tidy our dataset. We would like to combine the queries into one table. Let's get started:

1. Select one of the queries. Then, go to **Home | Append**, where you will select to either **Append Queries** or **Append Queries as New**. If you choose **Append Queries**, the existing queries for East and West will no longer exist and the result query will be created by combining the two queries together. **Append Queries as New** will give you the opportunity to keep the West and East queries as-is and create a brand-new combined query.
2. For this example, we will **Append Queries as New**. The **Append** dialog box will be populated, where you will see the description to concatenate rows from two tables into a single table. If you are combining three or more tables, this option is available. The **Primary** table has already been inserted for us. Select **SSGProductsWest** from **Table to append to the primary table** from the drop-down list provided:

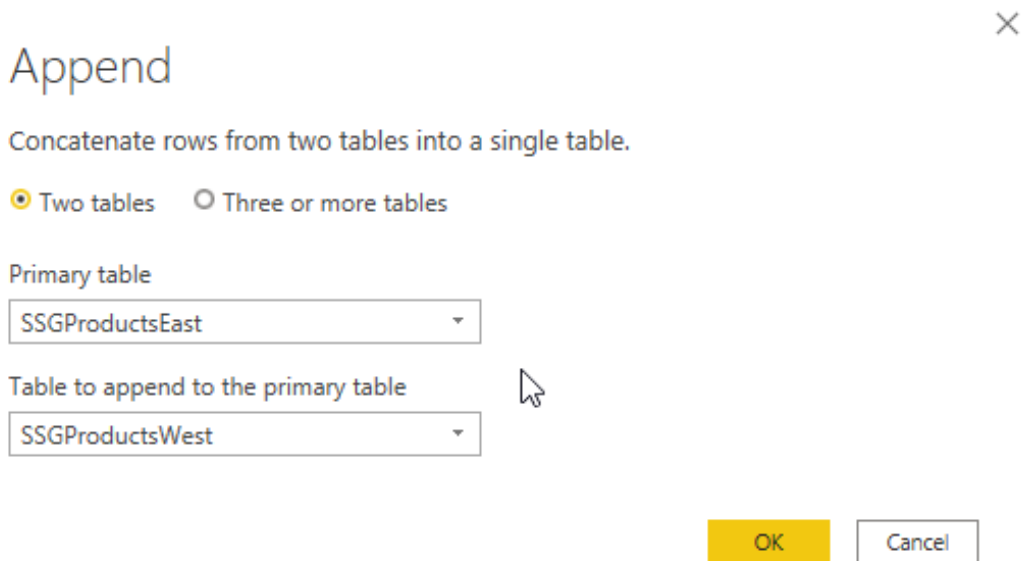


Figure 5.44 – Append as new dialog box

3. Click on **OK** to see the result.
4. A new query is evident in the **Queries** pane. It should be named **Append1**.

- Since we had one query with an additional column, there will be some `Null` values in our result. Simply clean the data by removing any blank rows. Make sure you have selected the **Append1** query, and then click on **Home | Remove Rows | Blank Rows**. You should now have 84 rows in the result (42 rows from the West and 42 rows from the East). Do not forget to save and apply the changes.

Note

The **Division** column has included data for the East salespeople but nothing for the West as the column did not exist in the underlying West query. A note to remember is that **Append** will not remove duplicate rows – if you have exactly the same row (with the same data) in both tables you are combining, it will not remove the duplicate. So, check your data after appending and remove any duplicate records.

- Make sure that, after appending the underlying queries, **SSGProductsEast** and **SSGProductsWest** are set not to load. You will gain greater performance from the model and save memory. You do not need all three queries to load into Power BI, only the resulting query. Right-click on the **SSGProductsEast** query and select **Enable load** to deselect it:

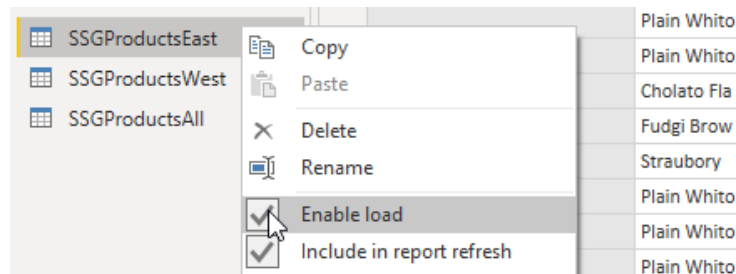


Figure 5.45 – Disabling loading the query into Power BI

- When removing the **Load to Power BI** option from a query, a notification window will appear, indicating that you could lose data during this process:

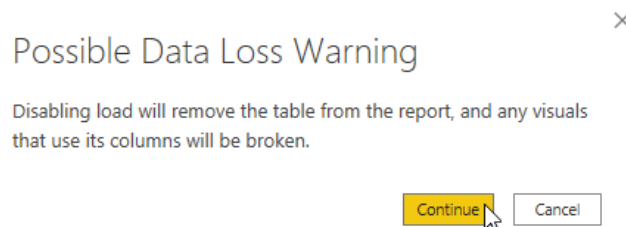


Figure 5.46 – Notification of data loss when disabling Load to Power BI

- Click on **Continue**. The queries now have an **Italics** attribute assigned, which indicates that these queries will not load to Power BI, but they will still continue to be refreshed.

Now that you know how to append tables from different sources into one query, we will look at grouping data according to different criteria.

Grouping data

In this section, we will learn how to group data from a table within Excel Power Query by specifying columns to group by and select the operation to perform as a new column. **Group By** in Power Query is very much like the **Group By** option in SQL.

We will use the **SalesData1** query to group the region and find the sum aggregate of sales for each region:

- Select the **SalesData** query in Power Query and duplicate it. Rename the query SalesGroup.
- Click on **Home | Group By**.
- We will first complete a **Basic** example. Click on the drop-down list located directly under **Basic** and make sure **Region** is selected. Set **New column name** to RegionSales and choose an operation to perform. The column to select to obtain the total sales for each region would be **Sales** in this case:

✕

Group By

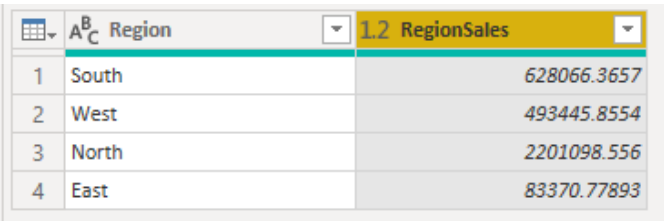
Specify the column to group by and the desired output.

☒ Basic
 ☐ Advanced

Region		
New column name	Operation	Column
RegionSales	Sum	Sales

Figure 5.47 – Group By dialog box

4. Click on **OK** to view the result:

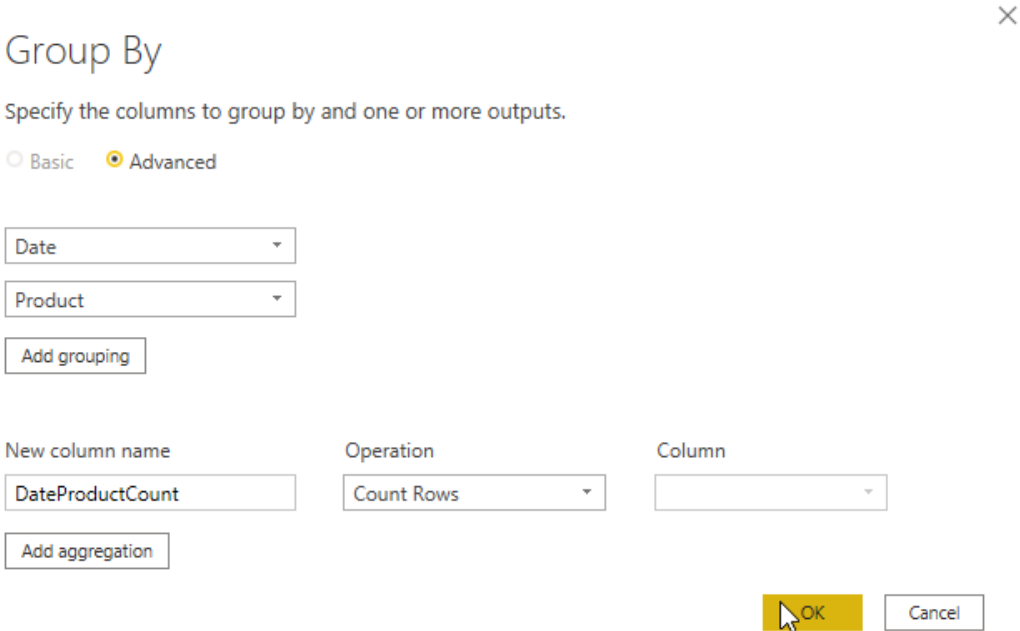


The screenshot shows a Power Query result table with two columns: 'Region' and 'RegionSales'. The 'Region' column has four rows: South, West, North, and East. The 'RegionSales' column shows the sum of sales for each region. The table is displayed in a grid with a yellow header row and a grey footer row.

	Region	RegionSales
1	South	628066.3657
2	West	493445.8554
3	North	2201098.556
4	East	83370.77893

Figure 5.48 – Result of sum of sales per region

5. Let's see how the **Advanced Group By** feature works. This time, import the SalesDataAdv.xlsx data source into Power Query.
6. Click on **Home | Group By**.
7. This time, select the **Advanced** option.
8. Use **Date** and **Product** as the columns to group by. Enter a **New column name** called **DateProductCount**, and then choose an **Operation** (for this example, we will count the rows as we would like to see how many products were sold on each date):



The screenshot shows the 'Group By' dialog box in Power Query. The 'Advanced' option is selected. The 'Date' and 'Product' columns are selected for grouping. The 'New column name' is 'DateProductCount', the 'Operation' is 'Count Rows', and the 'Column' is empty. The 'Add aggregation' button is visible at the bottom right.

Group By

Specify the columns to group by and one or more outputs.

☐ Basic ☒ Advanced

Date

Product

Add grouping

New column name: DateProductCount

Operation: Count Rows

Column:

Add aggregation

OK Cancel

Figure 5.49 – Entering the Advanced Group By criteria

9. Click on **OK** to view the results:

	A ^B _C Date	A ^B _C Product	1.2 DateProductCount
1	16/03/2015	Board2	1
2	02/05/2013	Game10	2
3	02/01/2014	Game3	3
4	02/10/2011	Dyno2	1
5	03/04/2011	Game8	2
6	16/09/2010	Dyno4	1
7	17/12/2015	Board2	1
8	16/03/2015	Game4	1
9	02/01/2014	Dyno5	2

Figure 5.50 – Advanced Group By result

The number of products sold on each date is added as a new column to the query. If you would like to edit the criteria, simply click on the gear icon to the right of the last applied step.

In this section, you learned how to group data in Power Query. We will now work with the various extraction tools available in the interface.

Working with extraction tools

At some point, you may have a very large table and need to create a number of separate Power Query tables out of the data source. The extract tool is perfect for this purpose. I need to mention here that there are many methods to achieve this, such as creating a duplicate query, and then deleting the columns you no longer require. We will also use the extract tool to extract the age of a student from a date in the query.

Extracting an age from a date

In this section, you will use Power Query to extract the age of employees from a date column, transform the time format into total year format, and round down the age column.

The first example we will look at is to extract the age of students from an enrolment list:

1. Import the student data table from `SSGSchoolAdmin.accdb` into Power Query.
2. Since the **DOB** column was converted into the Date/Time format type on import, format the **DOB** column to the **Date** data type.

3. Select the **DOB** column. Then, click on **Add Column | Date | Age**:

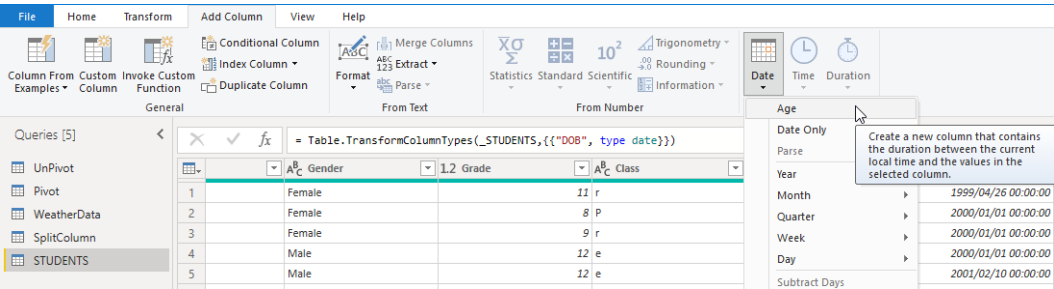


Figure 5.51 – Date to Age

4. The age column is presented to you as a time data format.
5. You now need to transform this into the year format. Click on **Transform | Duration | Total Years**.
6. The last step is to round the years down to the age. Locate the **Number Column** group on the **Transform** tab. Click **Rounding | Round Down** to complete this process.
7. Save the query.

You will now be confident in extracting an age from a date column in Power Query using the **Transform** option. In the following section, we will learn how to extract a column from a range.

Extracting columns

Let's use the extract tool to remove a column from a range in Excel using Power Query:

1. Open the Excel workbook named **SafestSolutions-Law.xlsx**.
2. You will notice that **Sheet2** contains a query that's already been set up using Power Query. Visit the **Queries & Connections** pane to view the query. Right-click on the query and choose **Edit**, after which you will return to the Power Query editor.
3. We will now extract the **CASE NO** column from the query and turn it into a list (which consists of one column). If we take a look at **Advanced Editor**, we will see that the **Source** statement refers to the current workbook and that **Output** refers back to the source. We will now change **Source** so that it reflects only the column we require. Do this by extracting the **CASE NO** column.

4. Click into the formula bar, and then edit the syntax so that it reads `= Source [CASE NO]`, meaning that it no longer refers to the entire table but only to the **CASE NO** column. You are also able to edit **Source** by editing the code directly in **Advanced Editor**.
5. Click on the enter tick to the left of the formula bar to accept the changes:

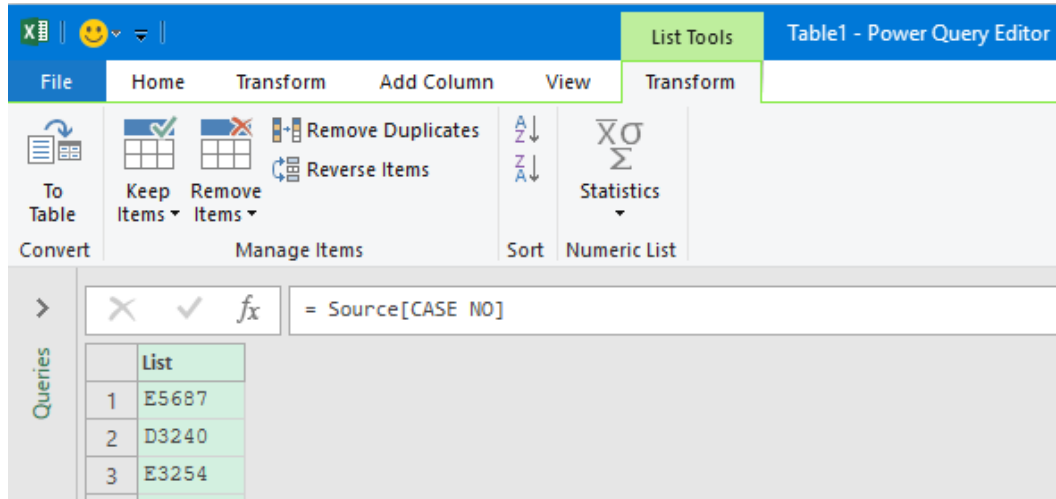


Figure 5.52 – Extracting a column in Power Query

6. You have now created your first column extract! The column is presented as a list.
7. Now, you can perform another step by converting the list into a table using the **To Table** option on the **Transform** tab. **Close & Load** the results to the Excel workbook when you are done.

Now that you've learned how to transform data within a query to extract columns, we will concentrate on the various options we can use to extract values using delimiters.

Using the extract column features

The **Extract** feature in Power Query has a multitude of options we can use to manipulate parts of a string in a data source to produce an output. We will learn where to locate this feature and how to use it to extract data from a column. Let's get started:

1. The scenario is that I have been sent a list of serial numbers from which I need to extract a certain part of the serial number to use in another query.
2. Open the workbook named `SerialNo.xlsx` and import the table into Power Query.

3. Click on **Transform | Extract** to view the options available:

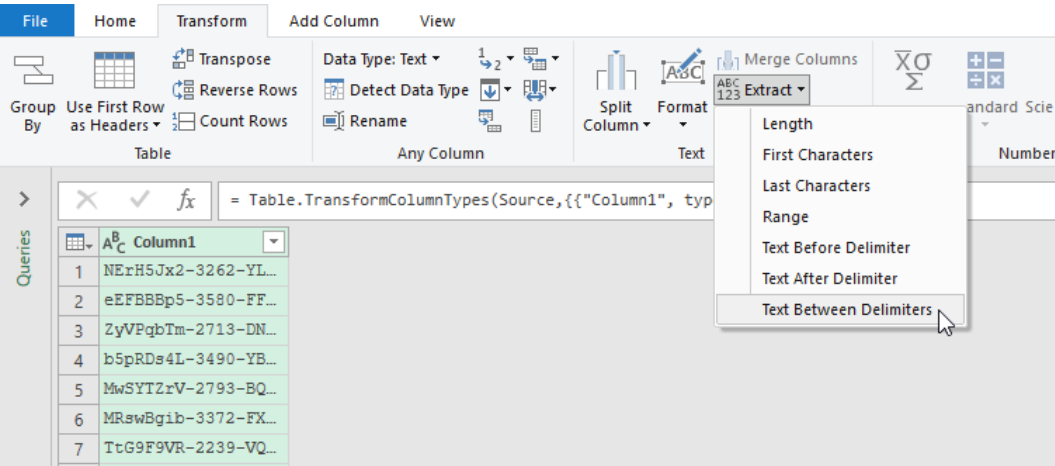


Figure 5.53: Transform data using Extract tool

4. We would like to extract the values in the middle of the serial number. Choose **Text Between Delimiters** from the list provided. This option will allow us to extract the numbers after and before the two dashes in the sequence.
5. Enter a dash as **Start delimiter** and a dash for **End delimiter**, and then click on **OK** to view the result:

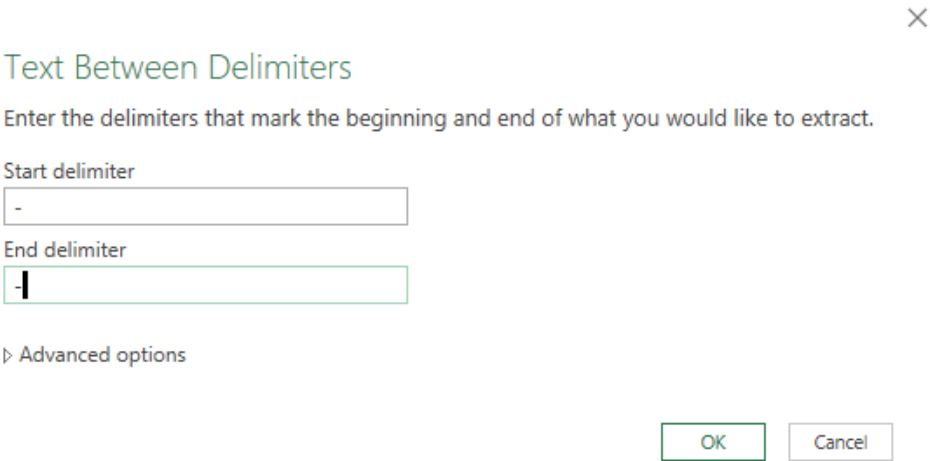


Figure 5.54 – Text Between Delimiters extraction

6. Click **Close & Load** to send the results back to Excel:

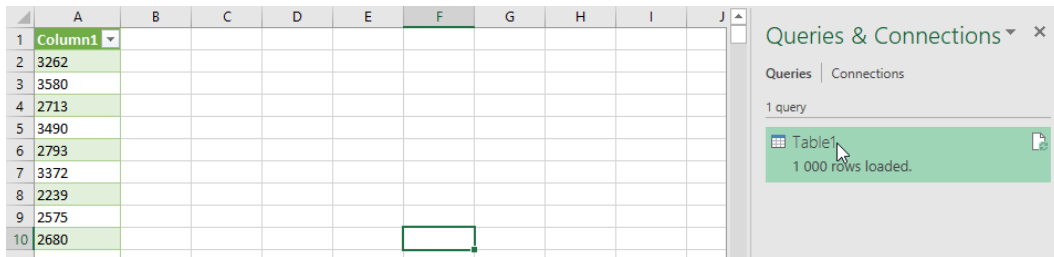


Figure 5.55 – Result of Close & Load to Excel

Let's try another option. Click on **Sheet1** of the workbook and import the range to create a second query in Power Query.

7. This time, we will use the **Range** option to select a certain start and end point of the data. As **Starting Index**, enter 4 to get rid of the first four characters in the sequence. For **Number of Characters**, enter 9, to keep the nine characters that appear after the first four characters, and remove anything after 9:

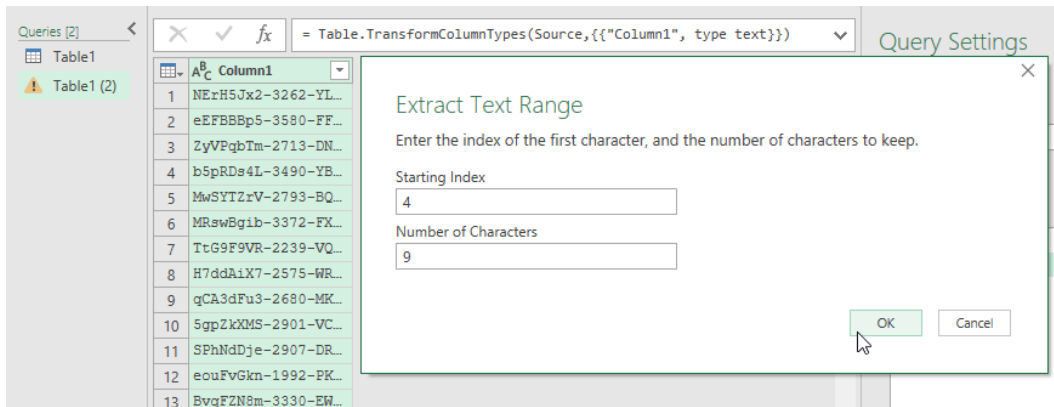


Figure 5.56 – Extract Text Range options

8. Click on **OK** to view the results, and then choose **Close & Load** to send the query back to Excel.

You have now learned how to use two different extract text options. Experiment with the other options in the list as they are all very useful when dealing with simple and complex data extractions.

Summary

In this chapter, you have been taught how to use the pivot tools to structure data correctly for analysis by being shown how to transform rows in order to display them across columns and vice versa. We had a look at the theory behind the **Refresh** option to understand the sequence of refresh, how to refresh, and its pros and cons. In the column and row tools section, you learned how to clean data by removing top or bottom rows from a table, removing duplicate rows, and replacing null values in a table and adding a header row. You also learned how to remove certain columns if you do not require them for analysis purposes and how to add an index column to act as a row counter that aids in data analysis. You are now able to meet a set of query conditions by creating a conditional column using the `if...then...else` statement. You consolidated your knowledge of filtering in Excel by learning how to sort and filter using the `AND/OR` statements, as well as how to create a dynamic multiple-criteria filter to extract data. With that, you can now split data based on delimiters.

We then spent some time mastering the extraction tools in order to combine column data or merge text and values into the same column and then moved on to understand how to group data within Power Query using the basic and advanced options. The chapter concluded by learning how to extract an age from a date by adding an age column and extracting a part of a serial number from an existing column.

In the next chapter, we will be working with advanced queries and functions, learn how to write `If` functions, and learn how to create parameter tables. We will discover a number of functions, such as `Modulo` and `Index`, append multiple files and tabs by creating new columns from filenames, and also do the same with sheet tabs using multiple data files.

6

Advanced Power Queries and Functions

This chapter concentrates on the more advanced queries and functions in Power Query such as the `IF`, `Index`, and `Modulo` functions. With these, you will be able to transform a range in Excel with no formatting into a highly customized table with only the relevant information that you require for your query.

You will learn to create parameters to alter query paths and append multiple files and sheet tabs. When using this, it will allow you to easily change the query paths without leaving Excel, which will then change the data source paths so every month you can add in your new data in a monthly folder and run the query instantaneously.

In this chapter, we're going to cover the following main topics:

- Writing an `IF` function in Power Query
- Creating a parameter table for queries
- Creating `Index` and `Modulo` functions
- Appending multiple files
- Appending multiple tabs

Technical requirements

You need an internet connection to download the relevant files from GitHub. Also, the code files for this chapter can be found at <https://github.com/PacktPublishing/Learn-Power-Query/>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=QSTjKk1MeKY&list=PLLeLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=7&t=0s.

This chapter assumes that you already know how to write basic IF, OR, and AND formulae in Excel. It is also important that you know how to load data into Power Query and how to add custom columns.

Writing an IF function in Power Query

Although writing an IF function might appear very different from the way that you normally write a formula in Excel, once you master it, it does get pretty easy. You might even find it easier than writing the formula in Excel.

Refer to the following screenshot of the spreadsheet:

	A	B	C	D	E	F	G	H	I	J
1	Order No	Order Date	Order Type	Delivery Date	Salesman	Region	Product	No. Customers	Net Sales	Profit / Loss
2	13658	01/01/2019	Online	null	Jordan	West	Dyno1	4	984	625
3	13663	01/01/2019	Pickup		John	North	Dyno3	8	2,253	653
4	13667	01/01/2019	Online	null	Jordan	West	Dyno2	4	920	569
5	13669	01/01/2019	Pickup		Lucie	Midlands	Dyno1	4	951	756
6	13670	01/01/2019	In Store	05/01/2019	Jordan	North	Dyno3	9	1,233	486
7	13673	01/01/2019	Pickup		Clive	North	Dyno3	7	1,064	436
8	13676	02/01/2019	Online	null	John	Midlands	Dyno2	8	1,736	690
9	13679	02/01/2019	Pickup		Clive	Midlands	Dyno3	10	2,540	835
10	13682	02/01/2019	Pickup		John	North	Dyno3	8	1,528	574
11	13684	02/01/2019	In Store	08/01/2019	Clive	Midlands	Dyno3	9	2,592	857
12	13688	03/01/2019	Online	null	Clive	Midlands	Dyno4	1	1,866	1,011
13	13689	03/01/2019	Pickup		Lucie	Midlands	Dyno2	10	1,430	534
14	13694	03/01/2019	Pickup		John	West	Dyno3	6	684	225
15	13695	03/01/2019	Online	null	Jordan	Midlands	Dyno3	9	1,512	503
16	13698	03/01/2019	Pickup		Clive	Midlands	Dyno2	10	2,240	722
17	13700	04/01/2019	Pickup		John	North	Dyno3	6	1,983	735

Figure 6.1 – Basic spreadsheet before editing

In this example, we are going to pretend that we have a spreadsheet and we would like to work out whether the placed order has been delivered. When looking at the spreadsheet, we currently do not have a place for this column and although we could add it in Excel straight away, I am going to do this in Power Query. One of the reasons for doing this is that this sheet might be a template from the company and we would have to add the column every time we wanted to do the query. By adding it in Power Query, it will automatically do this to any additional query with the same template.

We want to look at the delivery date and see whether the order has been delivered. If the order has not been delivered, we will want this flagged. We will need to add a new column called `Delivery Status`.

To load the Excel range into Power Query, we will convert it into a table and then select **From Table/Range** in the **Data** Tab, as shown in *figure 6.1*. Now, refer to the following screenshot:

Table3 - Power Query Editor

File Home Transform Add Column View

Column From Custom Examples Custom Column Invoke Custom Function

General

Conditional Column Index Column Duplicate Column

Format Merge Columns Extract Parse

From Text

Statistics Standard Scientific

From Number

Trigonometry Rounding Information

Date Time Duration

From Date & Time

Queries

	123	Order No	Order Date	Order Type	Delivery Date	Salesman	Region
1		13658	01/01/2019 00:00:00	Online		Jordan	West
2		13663	01/01/2019 00:00:00	Pickup		John	North
3		13667	01/01/2019 00:00:00	Online		Jordan	West
4		13669	01/01/2019 00:00:00	Pickup		Lucie	Midlands
5		13670	01/01/2019 00:00:00	In Store	05/01/2019 00:00:00	Jordan	North
6		13673	01/01/2019 00:00:00	Pickup		Clive	North
7		13676	02/01/2019 00:00:00	Online		John	Midlands
8		13679	02/01/2019 00:00:00	Pickup		Clive	Midlands
9		13682	02/01/2019 00:00:00	Pickup		John	North
10		13684	02/01/2019 00:00:00	In Store	08/01/2019 00:00:00	Clive	Midlands

Query Settings

PROPERTIES

Name

Table3

All Properties

APPLIED STEPS

Source

Changed Type

Figure 6.2 – Power Query Editor

Note

The loading of Excel ranges has been covered in *Chapter 2, Power Pivot Basics, Inadequacies, and Data Management*, if you are not sure how to do this.

In the Power Query Editor, select **Conditional Column** from the **Add Column** tab. This will bring up the **Add Conditional Column** dialog box, which will allow us to add in our custom column, shown as follows:

✕

Add Conditional Column

Add a conditional column that is computed from the other columns or values.

New column name

	Column Name	Operator	Value ①		Output ①
If	Delivery Date	does not equal	ABC 123 null	Then	ABC 123 Completed ...

Else ①

Figure 6.3 – Completed conditional column

As shown in figure 6.3, when looking at the completed dialog box, you can see how our **If** statement works. **If the Delivery Date does not equal null, Then** say it is **Completed**. If it is **null**, then say **null**. Once this has been completed, select **OK**.

We can see that the new **Delivery Status** column now has either **null** or **Completed** as the two options in this field:

Table3 - Power Query Editor

File Home Transform Add Column View

Column From Examples Custom Column Invoke Custom Function General

Conditional Column Index Column Duplicate Column

Format Merge Columns Extract Parse From Text

Statistics Standard Scientific Trigonometry Rounding Information From Number

Date From

	123 Order No	Order Date	ABC Order Type	ABC Delivery Date	ABC Delivery Status	ABC Salesman
1	13658	01/01/2019 00:00:00	Online	null	null	Jordan
2	13663	01/01/2019 00:00:00	Pickup	null	null	John
3	13667	01/01/2019 00:00:00	Online	null	null	Jordan
4	13669	01/01/2019 00:00:00	Pickup	null	null	Lucie
5	13670	01/01/2019 00:00:00	In Store	05/01/2019 00:00:00	Completed	Jordan
6	13673	01/01/2019 00:00:00	Pickup	null	null	Clive
7	13676	02/01/2019 00:00:00	Online	null	null	John
8	13679	02/01/2019 00:00:00	Pickup	null	null	Clive
9	13682	02/01/2019 00:00:00	Pickup	null	null	John

Figure 6.4 – Delivery Status column added

Tip

If you wanted to move the column to a different place, you can select the column and move it in the same way that you move a column in Excel.

When we look at the M code in the **Advanced Editor**, you can see the formula that has been written is similar to Excel but has a few other rules. Refer to the following screenshot:

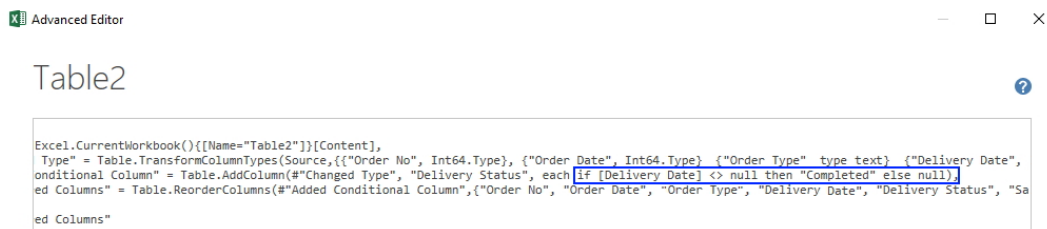


Figure 6.5 – The M code in the Advanced Editor

When you are using the GUI Power Query Editor, it is really easy to create additional columns. Sometimes, however, it is not as easy as this and we will then have to write the formula ourselves.

Note

The difference between writing an Excel formula and a Power Query formula is as follows:

Excel=**IF(test, ValueIfTrue, ValueIfFalse)**

Power Query =**if test then ValueIfTrue else ValueIfFalse**

You will notice that there are no brackets or parentheses in Power Query, but you do need to use the words **then** and **else**.

If in Power Query is in lowercase and **null** means blank in Power Query.

When looking at *figure 6.5*, you might notice that if a person comes to pick up their order in the store, it does not show this as being delivered. As a result, we will need to change our **if** statement to an **if or** statement to show that the order has indeed already been delivered.

To make this change, we will need to do the same thing as before and add a custom column. The only difference with this is we will have to write the M code and we will not be able to use the GUI interface. Follow the same steps as we did earlier in this section and when you get to the Power Query Editor, click **Custom column**, shown as follows:

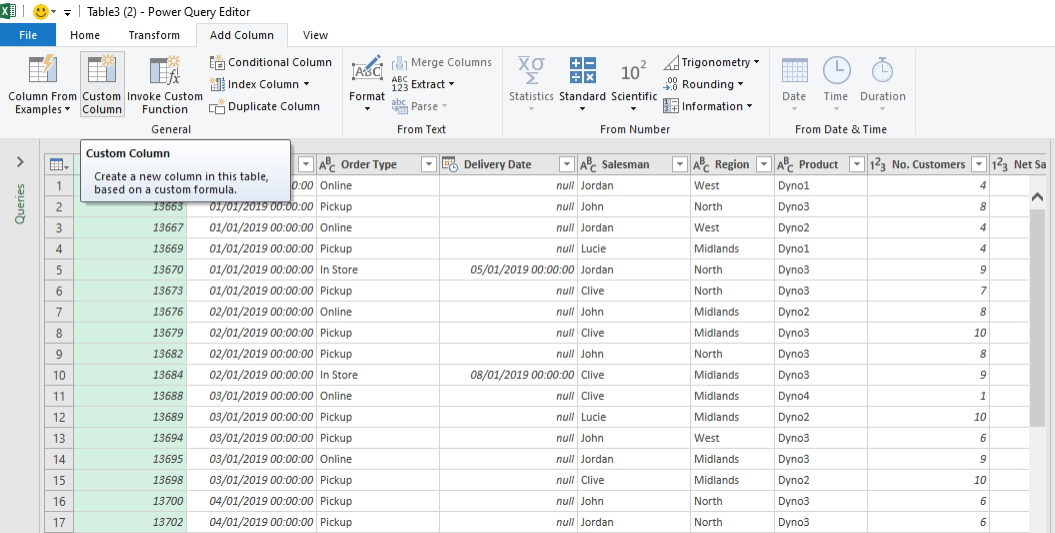


Figure 6.6 – Custom Column

You can now start typing the following in the window that has opened:

```
if [Delivery Date] <> null or [Order Type] = "In Store
Pickup" then "Completed" else null
```

I find that this is easier than writing everything; you can double-click on the columns that are available on the right, which will then insert that column name into the formula. Refer to the following screenshot:

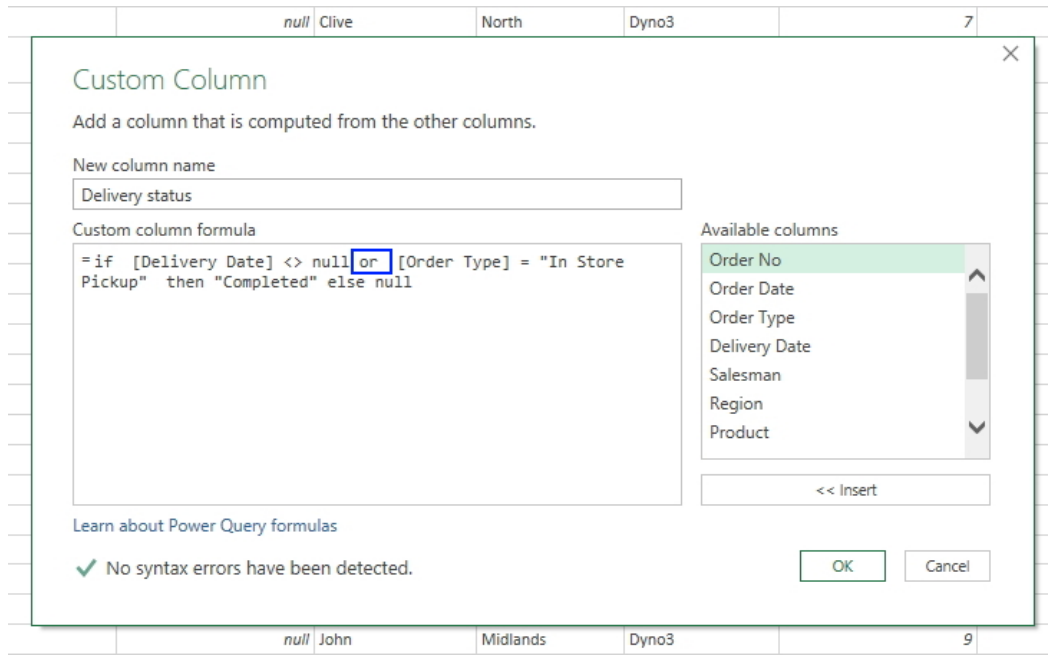


Figure 6.7 – Available columns

When we look at the formula we have typed in, it says that if the **Delivery Date** is not blank or the order type is **In Store Pickup**, then it needs to say **Completed**; otherwise, it needs to leave the cell blank. Refer to the following screenshot:

The screenshot shows the Power Query Editor interface with a table containing 23 rows of data. The columns are: Order No, Order Date, Order Type, Delivery Date, Delivery Status, Salesman, Region, and Product. The 'Delivery Status' column is calculated using an AND IF function: `if [Delivery Date] <> null or [Order Type] = "In Store Pickup" then "Completed" else null`. The 'Delivery Date' column is highlighted in green for rows where the status is 'Completed'.

	Order No	Order Date	Order Type	Delivery Date	Delivery Status	Salesman	Region	Product
1	13658	43466	Online	null	null	Jordan	West	Dyno1
2	13663	43466	Pickup	06/01/2019...	Completed	John	North	Dyno3
3	13667	43466	Online	null	null	Jordan	West	Dyno2
4	13669	43466	Pickup	04/01/2019...	Completed	Lucie	Midlands	Dyno1
5	13670	43466	In Store	05/01/2019...	Completed	Jordan	North	Dyno3
6	13673	43466	Pickup	08/01/2019...	Completed	Clive	North	Dyno3
7	13676	43467	Online	null	null	John	Midlands	Dyno2
8	13679	43467	Pickup	07/01/2019...	Completed	Clive	Midlands	Dyno3
9	13682	43467	Pickup	05/01/2019...	Completed	John	North	Dyno3
10	13684	43467	In Store	08/01/2019...	Completed	Clive	Midlands	Dyno3
11	13688	43468	Online	null	null	Clive	Midlands	Dyno4
12	13689	43468	Pickup	06/01/2019...	Completed	Lucie	Midlands	Dyno2
13	13694	43468	Pickup	08/01/2019...	Completed	John	West	Dyno3
14	13695	43468	Online	null	null	Jordan	Midlands	Dyno3
15	13698	43468	Pickup	07/01/2019...	Completed	Clive	Midlands	Dyno2
16	13700	43469	Pickup	08/01/2019...	Completed	John	North	Dyno3
17	13702	43469	Pickup	07/01/2019...	Completed	Jordan	North	Dyno3
18	13705	43469	Online	null	null	Lucie	North	Dyno3
19	13709	43469	Online	null	null	Clive	North	Dyno1
20	13712	43472	Pickup	10/01/2019...	Completed	Clive	West	Dyno3
21	13715	43472	Pickup	13/01/2019...	Completed	Clive	West	Dyno1
22	13720	43472	Pickup	11/01/2019...	Completed	Jordan	North	Dyno2
23	13725	43472	Pickup	14/01/2019...	Completed	Lucie	West	Dyno3

Figure 6.8 – The completed AND IF function

The steps are separated by the word **or**, making this easier to read (see *figure 6.8*).

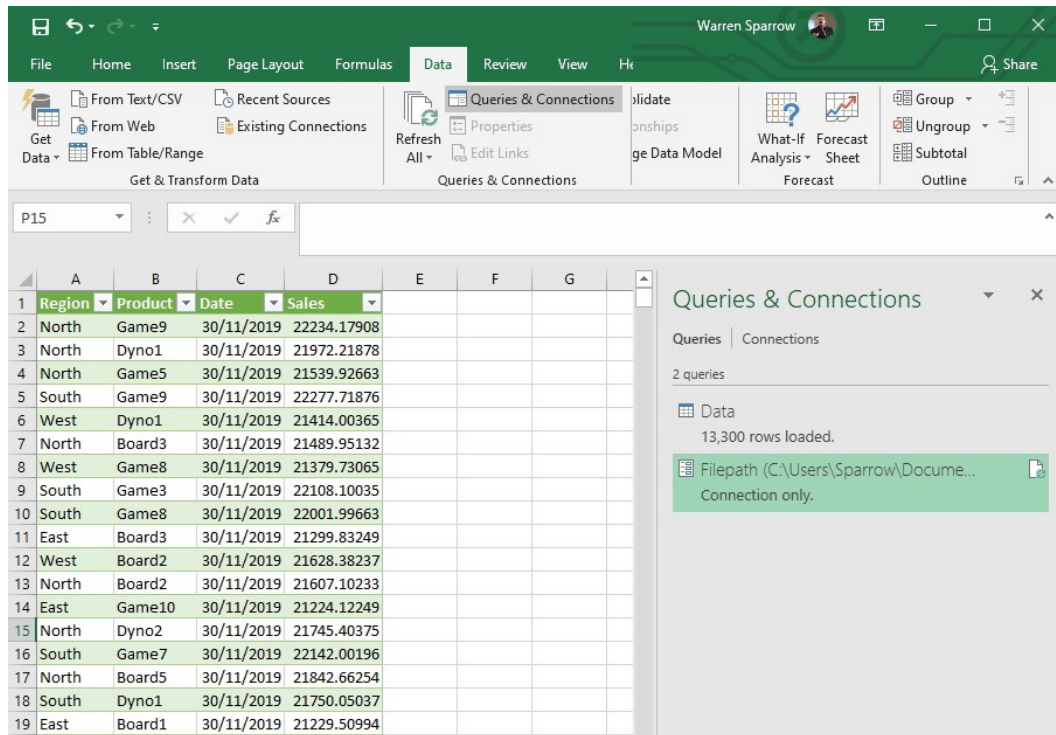
The `if` statement in the Query Editor allows you to create basic functions through the GUI, and it allows you to make more powerful functions when you write your own code. Although there is some similarity to Excel formulas, there is a slight difference, but when reading the M programming language, it does become apparent what the formula is doing.

In this section, we added an additional column that we needed. The next section will show how we can easily create a custom or updated path for our data source so we can update our query without leaving the Excel table.

Creating a parameter table for queries

When we design a query for something specific, for example, a monthly sales query, we would generally create it for that month. The following month, we would want to use the same query but have a different source document. Naturally, we would not want to redo the entire query but just change the query for that specific month. So, in this way, we can make the necessary additions to the query to make our lives easier and create a parameter to the query.

I am using the same Excel files from *Chapter 4, Connecting to Various Data Sources using Get & Transform*. The following is a CSV file that I have opened and I have connected it to a query:



	A	B	C	D	E	F	G
1	Region	Product	Date	Sales			
2	North	Game9	30/11/2019	22234.17908			
3	North	Dyno1	30/11/2019	21972.21878			
4	North	Game5	30/11/2019	21539.92663			
5	South	Game9	30/11/2019	22277.71876			
6	West	Dyno1	30/11/2019	21414.00365			
7	North	Board3	30/11/2019	21489.95132			
8	West	Game8	30/11/2019	21379.73065			
9	South	Game3	30/11/2019	22108.10035			
10	South	Game8	30/11/2019	22001.99663			
11	East	Board3	30/11/2019	21299.83249			
12	West	Board2	30/11/2019	21628.38237			
13	North	Board2	30/11/2019	21607.10233			
14	East	Game10	30/11/2019	21224.12249			
15	North	Dyno2	30/11/2019	21745.40375			
16	South	Game7	30/11/2019	22142.00196			
17	North	Board5	30/11/2019	21842.66254			
18	South	Dyno1	30/11/2019	21750.05037			
19	East	Board1	30/11/2019	21229.50994			

Figure 6.9 – Imported CSV file in the query

When you edit the query, you will see that it is a simple one with mostly the default steps. The only thing I have added is that I have changed the **Sales** column to the currency type.

If you click on the gear icon on the right-hand side of the source in the **APPLIED STEPS**, it will open the **PROPERTIES** window. Refer to the following screenshot:

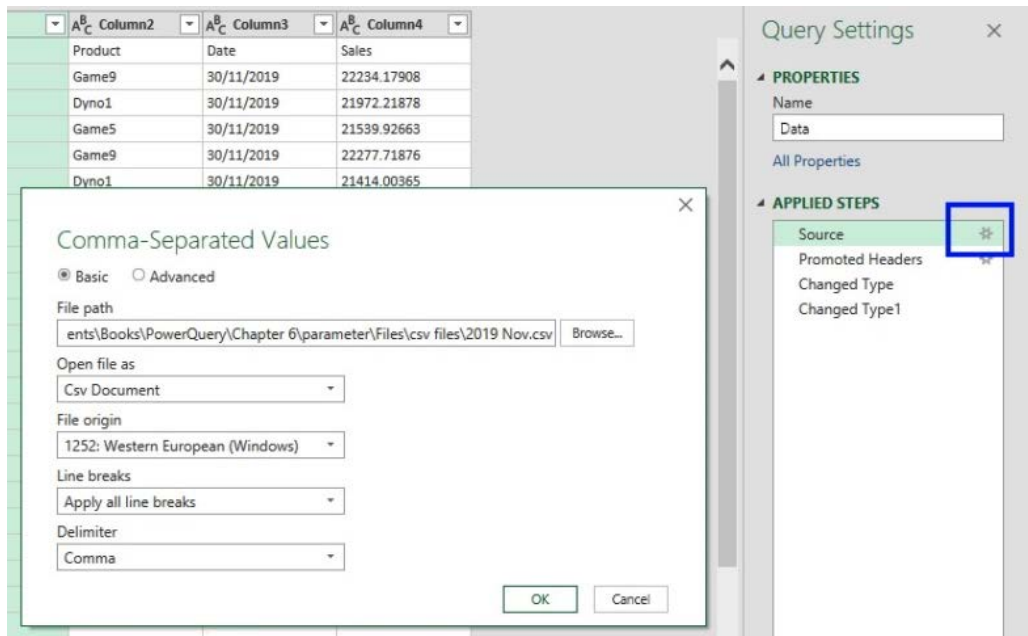


Figure 6.10 – The properties of the source file

In this window, you are able to see the **File path** of the source document and the type of file as well as what the delimiter is.

At this time, the file or value is hardcoded into our query. This means that when we open this query, it will always attempt to open this file in the current location.

We will create a new parameter called `FilePath`, which will allow us to change this parameter every month. Click on **Manage Parameters** and then select **New Parameter**, as shown in the following screenshot:

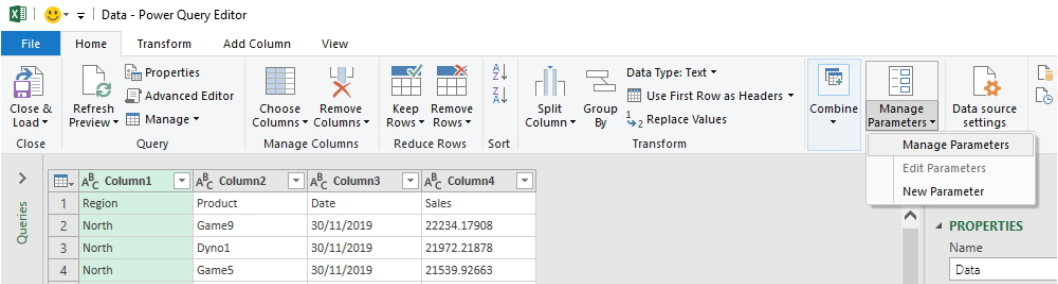


Figure 6.11 – Manage Parameters

This will open the **Parameters** window, as follows:

Parameters

New

FilePath

Name
FilePath

Description
This is the parameter that will change the file path

☒ Required

Type
Text

Suggested Values
Any value

Current Value
erQuery\Chapter 6\parameter\Files\csv files\2019 Nov.csv

OK Cancel

Figure 6.12 – Parameters window

In the **Parameters** window, type in the name of the parameter, (FilePath); select **Type** as **Text**, as this is a text-based parameter; and lastly, in the **Current Value**, type in the current file path and the name of the file.

Once completed, click on **OK** and you will now see the new query on the query pane on the left, as shown here:

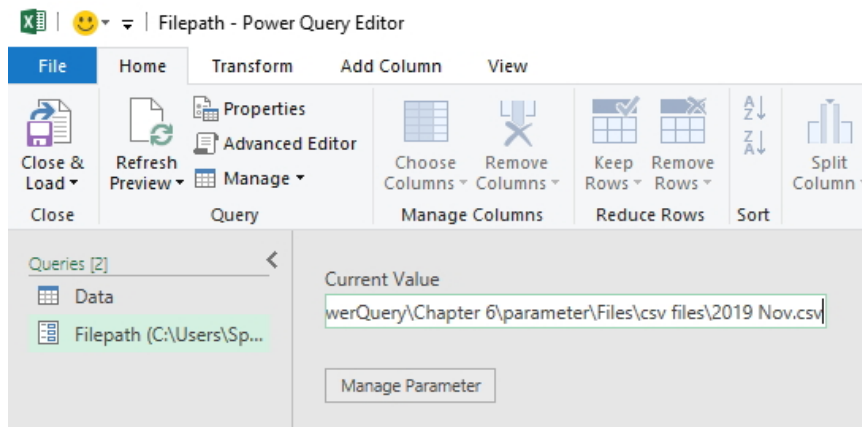


Figure 6.13 – New FilePath query

When you click on the **Data** query in the query pane, it will take you to the same screen as *figure 6.10*. When we click again on the gear icon on the right-hand side of the source in the **APPLIED STEPS**, it will open the **PROPERTIES** window again. What we are going to do is change the path from the existing file path (C:\Users\...\parameter\Files\csv files\2019 Nov.csv) to the **Parameter Filepath**, as shown here:

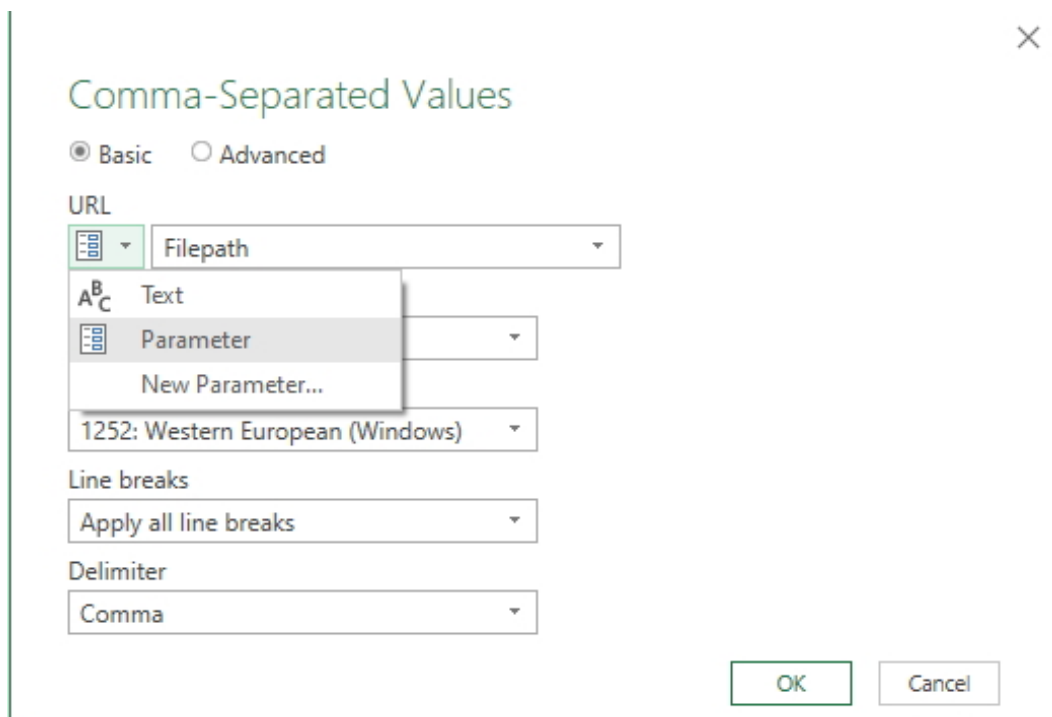


Figure 6.14 – New Parameter filepath

One of the things you will need to do is change the **URL Text** to **Parameter**. I find it easier to select **Parameter** and then select the **Filepath** parameter I need. Once you have completed this, you can click **Close and Load**, which will load the query into Excel. Refer to the following screenshot:

The screenshot shows the Microsoft Excel interface with the 'Data' tab selected. The 'Queries & Connections' pane on the right displays two queries. The first query, 'Data', is highlighted and shows '13,300 rows loaded.' The second query, 'Filepath (C:\Users\Sparrow\Docume...', is listed below it with the note 'Connection only.' The main worksheet area shows a table with the following data:

	A	B	C	D	E	F
1	Region	Product	Date	Sales		
2	North	Game9	30/11/2019	22234.1791		
3	North	Dyno1	30/11/2019	21972.2188		
4	North	Game5	30/11/2019	21539.9266		
5	South	Game9	30/11/2019	22277.7188		
6	West	Dyno1	30/11/2019	21414.0036		
7	North	Board3	30/11/2019	21489.9513		
8	West	Game8	30/11/2019	21379.7306		
9	South	Game3	30/11/2019	22108.1004		
10	South	Game8	30/11/2019	22001.9966		
11	East	Board3	30/11/2019	21299.8325		
12	West	Board2	30/11/2019	21628.3824		
13	North	Board2	30/11/2019	21607.1023		
14	East	Game10	30/11/2019	21224.1225		
15	North	Dyno2	30/11/2019	21745.4038		
16	South	Game7	30/11/2019	22142.002		
17	North	Board5	30/11/2019	21842.6625		
18	South	Dyno1	30/11/2019	21750.0504		
19	East	Board1	30/11/2019	21229.5099		
20	North	Game9	30/11/2019	21946.1079		

Figure 6.15 – The query table connected via the Filepath parameter

You will now notice that the query is still connected to the same file, but it is now connected via a parameter instead of a direct link.

Changing the monthly data source

We are now going to pretend that this is the next month and we would like to use the existing Power Query template, but we now need the new month's data. So, we now need to make some changes; please refer to the following screenshot:

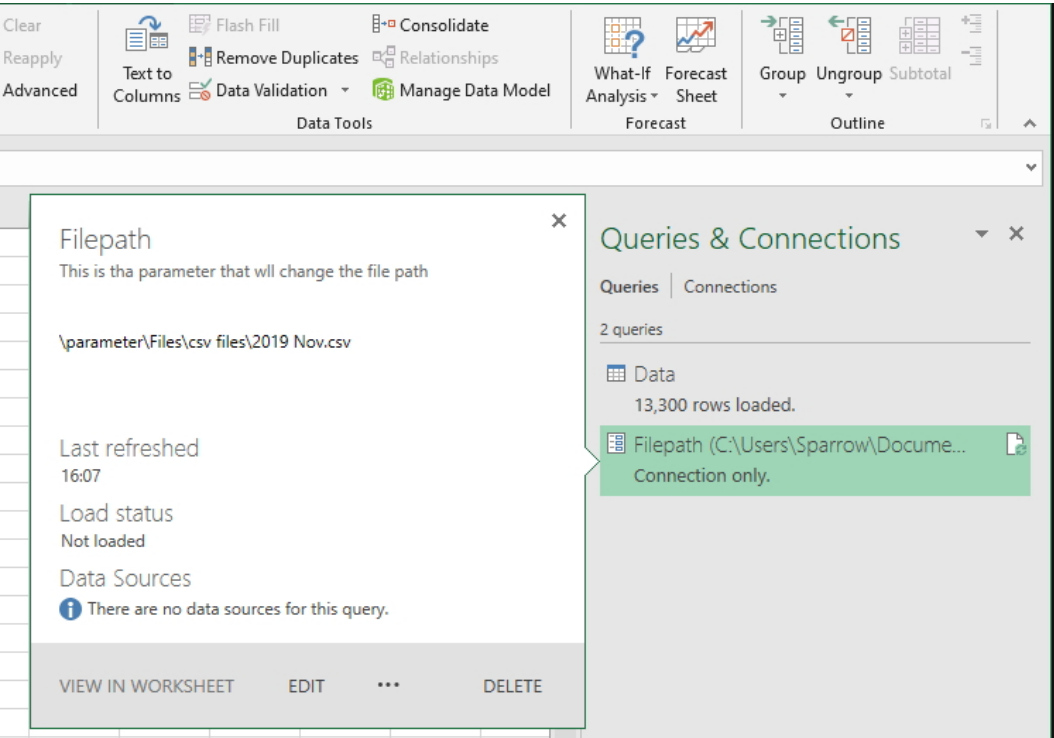


Figure 6.16 – Edit FilePath

We first click on **EDIT** from the Filepath parameter when you hover over the **Queries & Connections** window. This will open a window that will allow you to change the **Filepath** parameter by selecting the **Manage Parameter** icon, shown as follows:

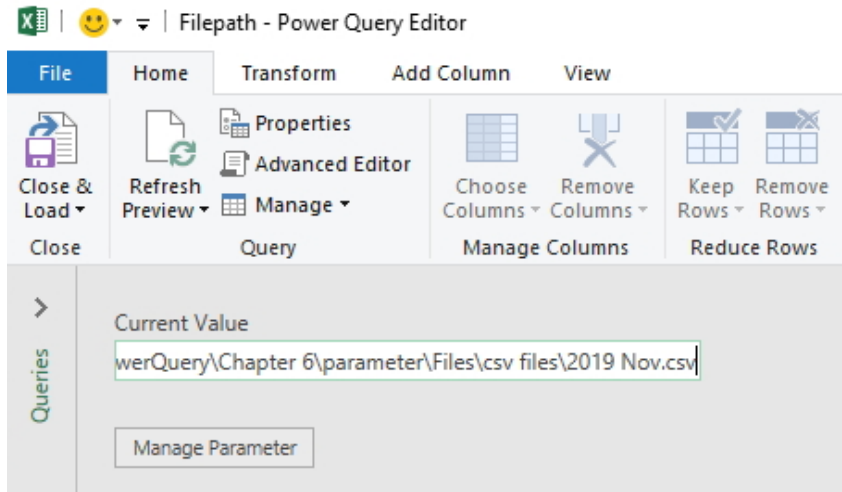


Figure 6.17 – Manage Parameter

Once you have changed this to the new file, simply select **Close and Load** when you are finished and this will then display your new data.

Although this is relatively easy every month to change your Filepath parameter, it is a bit time consuming to open the Query Editor and then change the parameter. It is possible to create a table of parameters, which will shorten the process. Let's see how this is done:

Parameter	Value
Folder	C:\Users\Sparrow\Documents\Books\PowerQuery\Chapter 6\parameter\Files\csv files\
File name	2019 Dec.csv

Region	Product	Date	Sales
North	Game9	30/11/2019	22234.1791
North	Dyno1	30/11/2019	21972.2188
North	Game5	30/11/2019	21539.9266
South	Game9	30/11/2019	22277.7188
West	Dyno1	30/11/2019	21414.0036
North	Board3	30/11/2019	21489.9513
West	Game8	30/11/2019	21379.7306
South	Game3	30/11/2019	22108.1004

Figure 6.18 – Table of parameters

I have created a simple table called **Parameter**, which has a **Parameter** column and the **Value** of the folder, which has the value of the filename. We are going to link our existing query to the parameter table. This will allow us to change the values in the parameter table and then will automatically update the query. Let's see how:

1. We will need to create a new query by selecting **Get Data | From Other Sources | Blank Query**.

I am going to rename my new query `fParameter`. This is going to be the function name that I am going to use to call the values from a parameter table. (I am keeping the same format as I have done in the previous chapters.)

2. Then, click on **Advanced Editor** in the **View** tab and delete all of the default information that comes up. You will need to replace it with the following code snippet:

```
let Parameter=(TableName, ParameterLabel) =>
let
    Source = Excel.CurrentWorkbook() { [Name=TableName] }
    [Content] ,
    value = Source{ [Parameter= ParameterLabel] } [Value]
in
    value
in Parameter
```

This is a query function and when you look at the code, it takes two arguments, namely, `TableName` and `ParameterLabel`, and connects them to our table. In the **Parameter** column, we are going to use a lookup and then return that value in the **Value** column. Once completed, click on **Done**:

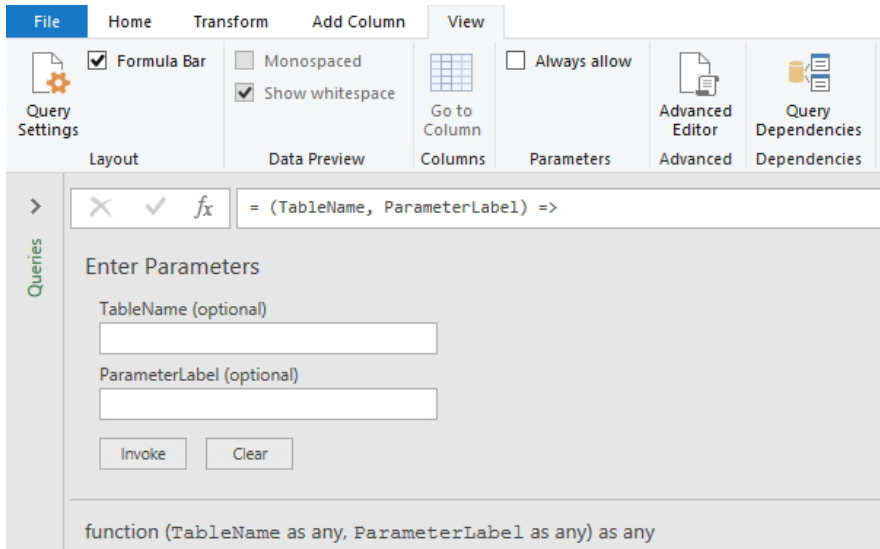


Figure 6.19 – Creating the parameters

We will need to alter the **FilePath** that we created before and insert additional code so that we are able to change the table in Excel and it will then automatically update.

From the current window, select the **Data** query and then select the source step. We are going to delete the **FilePath** parameter and insert additional code:

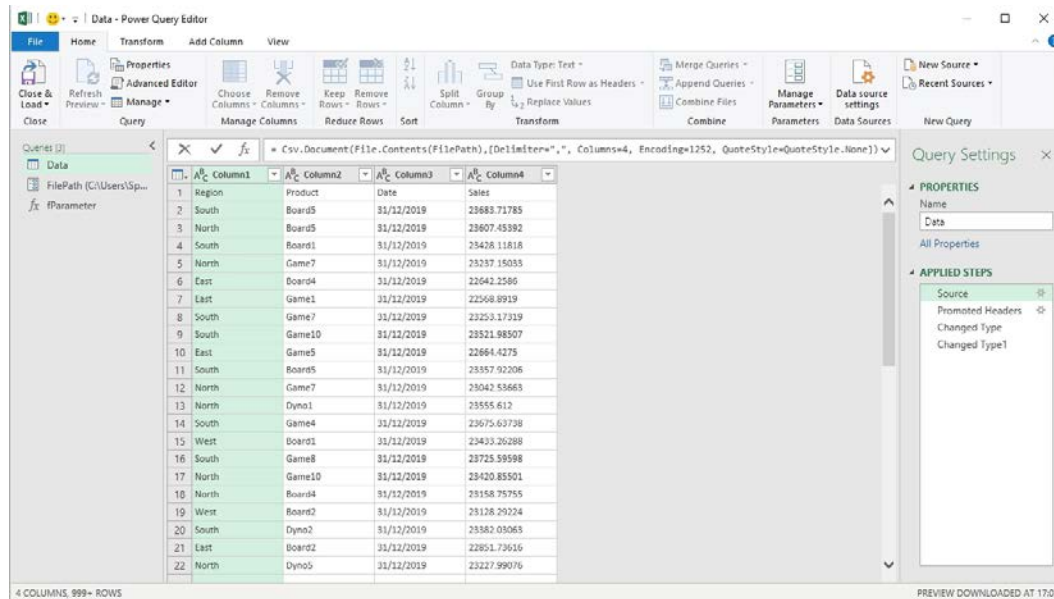


Figure 6.12 – Changing the FilePath parameter

Delete the **FilePath** parameter from the code in the toolbar, but do not delete the parentheses, (). Copy the following code and insert it in place of **FilePath**:

```
fParameter("Parameter", "Folder") & "\" &
fParameter(Parameter, FileName)
```

From the preceding code, we are calling the ParameterTable parameter and our folder path and then we are going to append this to FileName from ParameterTable.

Once you have finished, click **Close and Load**, which will bring you back to the table in Excel. You can now change the **Folder** path and the **FileName** of the various files and after you refresh, the data it will then load that file, as shown here:

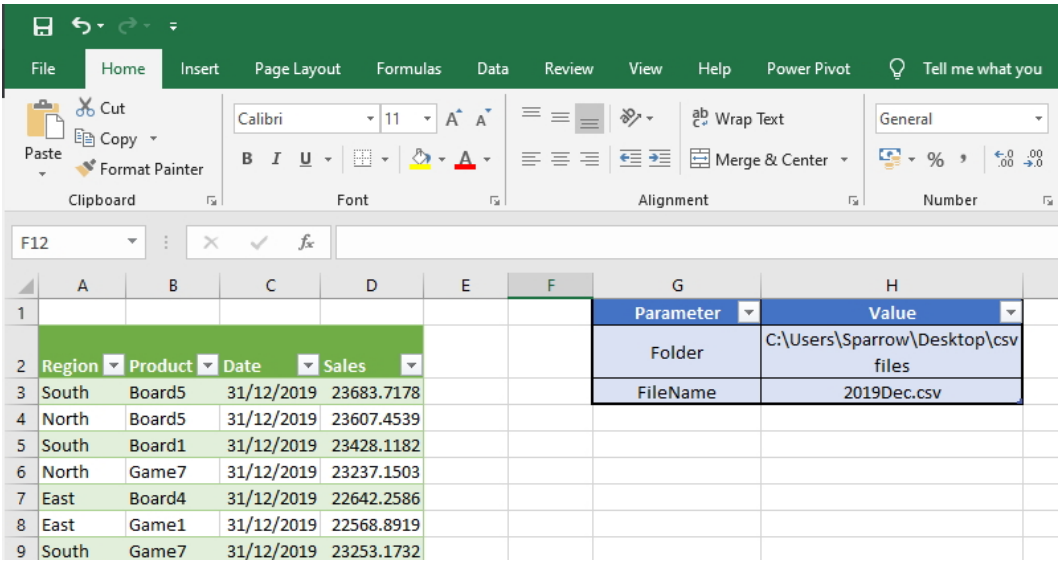


Figure 6.13 – Change the data in the Parameter table

Using the parameter tables allows you to quickly change the file from the current one without having to go and edit the data source every time.

Using the parameter tables works really well if your table is formatted correctly, but many times, you will have to format the table in Power Query before you can use parameter tables.

Using the Index and Modulo functions, we are able to remove the information we do not need. So, we will study these in the next section.

Understanding the Index and Modulo functions

In this section, we will be looking at the index and modular functions. These tools allow us to remove rows and create additional rows that we can order, which will assist in filtering the relevant data that we need. We will also explore how to rename and delete steps, which will make it easier to understand which step is associated with what it is doing.

Beginning with the modulo function

Before we start looking at the *MOD* function, it might be beneficial to explain what it is and how it can be used. *Modulo* is a term used in computing and mathematics that works out the remainder after you have divided a number by a different number. For my example, let's say that we are going to divide 11 by 4. This would give us an answer of 2 with a remainder of 3. This could be very practical in a class situation if you have to divide a class of students into groups of a specific number. Using the formula `=mod (A3, B3)` in cell C3, this will give you a result of 1. This means that 1 student is left over and could not fit into a group. We have represented this in the following screenshot:

	A	B	C	D
1				
2	Number of students in classes	Group size	Left Over	
3	35	17	1	
4	27	13	1	
5	22	10	2	
6	29	13	3	
7	26	13	0	
8	24	11	2	
9	23	11	1	
10	19	9	1	

Figure 6.14 – A MOD example

Although this seems a bit basic, we can expand this, as we can use MOD with other formulae, such as Sum and Column.

Let's have a look at the following table of sales. I am going to pretend that we want to find out the sales of every even month:

Cut Copy Paste		Calibri 11 Arial Bold Italic Underline Text Color Background Color		Wrap Text Merge & Center		General Number		Conditional Formatting Table Styles		Cell Styles		Insert Delete Format Cells		AutoSum Fill Clear		Sort & Find Filter Select Editing	
Clipboard		Font		Alignment		Number											
F12 {=SUM(B2:M9*(MOD(COLUMN(B2:M9), 2)=1))}																	
	A	B 2	C 3	D 4	E 5	F 6	G 7	H 8	I 9	J 10	K 11	L 12	M 13				
1	Month	MOD 2	January 0	February 1	March 0	April 1	May 0	June 1	July 0	August 1	September 0	October 1	November 0	December 1			
2			41	45	29	33	36	42	42	38	27	31	38	30			
3			38	44	28	39	38	32	28	31	47	35	29	26			
4			37	25	32	34	47	36	40	38	38	38	31	40			
5			45	41	39	28	48	36	46	31	45	28	35	36			
6			44	34	44	35	28	35	45	25	41	49	25	42			
7			49	42	43	34	50	33	34	35	32	38	37	46			
8			43	29	40	48	39	41	35	42	34	31	37	44			
9			46	44	25	39	35	29	36	33	26	37	50	33			
10																	
11																	
12						Total Sales in even months	1735										

Figure 6.15 – Table of sales

The formula that I am going to use is `=SUM(B2 :G4 * (MOD (COLUMN (B2 :G2) , 2) =1))`. The `COLUMN` returns the column number; these are the numbers in bold that are above the column names of each column. The `MOD` function divides the column number by 2, hence column 4 divided by 2 has the remainder 0 and the answer will be 0. Column 5 divided by 2 will give a remainder of 1, hence 1 will be the answer. If we wanted to change our query to every 3 months, we can change the number 2 to 3. We will either have an answer of 0 or 1 for each column. If there is a remainder, then the formula will become `1 = 1`, which is true and we will then add those columns together.

Note

If you are doing this, then do not press *Enter* when you type in the formula. You will need to press *Ctrl + Shift + Enter* at the same time as this is an array.

Now that we understand how `MOD` works in Excel, let's have a look at how we can use this in Power Query.

Sometimes, we will have files that have not been formatted the way we want them to be. It could take ages to get rid of all of the irrelevant information. The following is an example of a file that is a record of all of the sales for December 2019:

	A	B	C	D	E	F	G
1		Dyno Sales					
2		Monthly End Report					
3		Dec-19					
4							
5		Order Number		Product Sold		Sales Person	
6		13658		Dyno1		Jordan	
7		Number of Customers		Net Sales		Profit	
8		4		984		625.28	
9		Order Number		Product Sold		Sales Person	
10		13663		Dyno3		John	
11		Number of Customers		Net Sales		Profit	
12		8		2253		653.2	
13		Order Number		Product Sold		Sales Person	
14		13682		Dyno3		John	
15		Number of Customers		Net Sales		Profit	
16		8		1528		573.6112	
17		Order Number		Product Sold		Sales Person	
18		13684		Dyno3		Clive	
19		Number of Customers		Net Sales		Profit	
20		9		2592		857.4336	
21		Order Number		Product Sold		Sales Person	

Figure 6.16 – Monthly End Report file

Using Power Query, we will be able to get this in a format that we can use.

Start by selecting everything on the page, including the header at the top, and select **From Table / Range** in the **Data** tab. Create the table from the **Create Table** window and select OK, which will bring up the following screenshot:

ABC123 Column1	ABC123 Column2	ABC123 Column3	ABC123 Column4	ABC123 Column5	ABC123 Column6
1	Dyno Sales	null	null	null	null
2	Monthly End Report	null	null	null	null
3	01/12/2019 00:00:00	null	null	null	null
4	null	null	null	null	null
5	Order Number	Product Sold	Sales Person		
6	13658	Dyno1	Jordan		
7	Number of Customers	Net Sales	Profit		
8	4	984	625.28		
9	Order Number	Product Sold	Sales Person		
10	13663	Dyno3	John		
11	Number of Customers	Net Sales	Profit		
12	8	2253	653.2		
13	Order Number	Product Sold	Sales Person		
14	13682	Dyno3	John		

Figure 6.17 – The data in the Query Editor

Here, among other features, let's see how we can remove the top four rows as we do not need any headings for the table. Click on **Remove Rows** and **Remove Top Rows** and then, in the **Remove Top Rows** window, type 4 and press **OK**, as shown here:

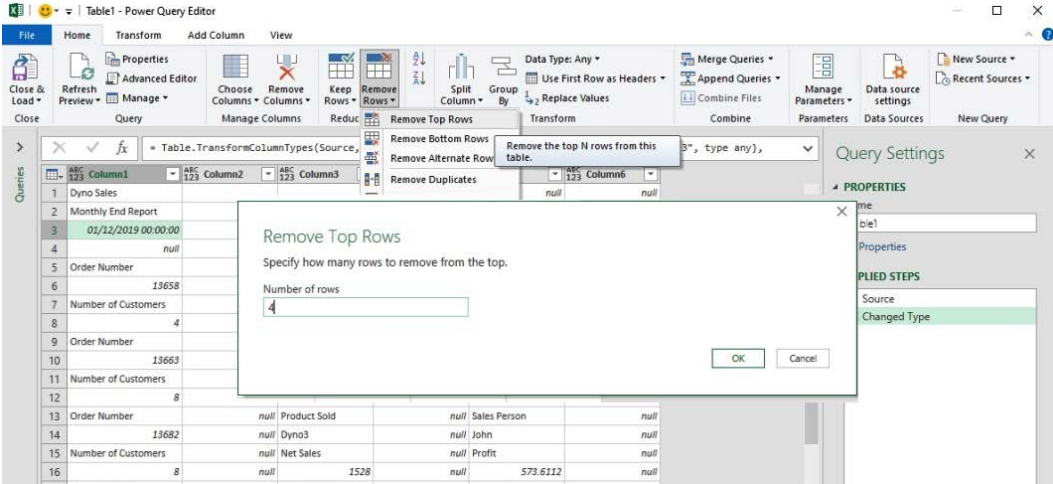


Figure 6.18 – Delete the top four headings

Understanding index functions

A very powerful feature of Power Query is to use an index column, which serves as a row counter for all of our data. We would like to add an index in the Modulo function so that we can work out how many rows there are before the information repeats itself.

To add an index column, select **Index Column** from the **Add Column** tab. You are given options: whether you would like the index column to start **From 0** or **From 1** or whether you would like to create a **Custom** column. For this exercise, I have selected it to start **From 0**, as seen in the following screenshot:

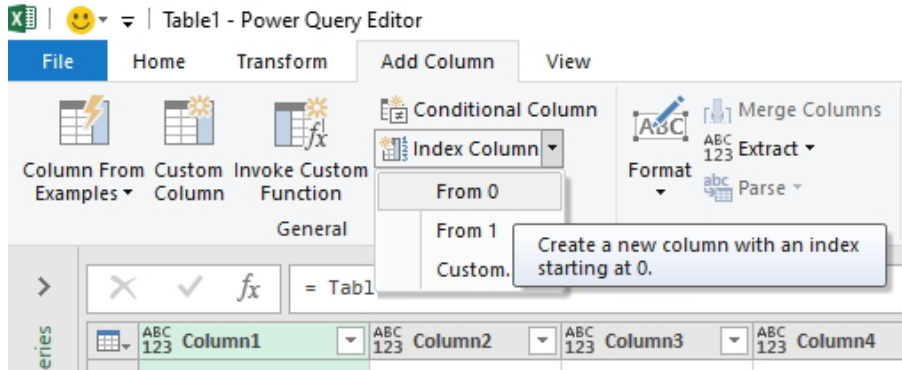


Figure 6.19 – Insert the Index Column

When looking at the following screenshot, you will notice that the first **Order Number** starts at row 1 and then the next record starts at row 5:

Table1 - Power Query Editor

File Home Transform Add Column View

Column From Examples Custom Column Invoke Custom Function General

Conditional Column Index Column Merge Columns Format Extract Parse

Statistics Standard Scientific Rounding Information Date Time

From Text From Number From Date

fx = Table.AddIndexColumn("#Removed Top Rows", "Index", 0, 1)

	Column1	Column2	Column3	Column4	Column5	Column6	1.2 Index
1	Order Number	null	Product Sold	null	Sales Person	null	0
2	13658	null	Dyno1	null	Jordan	null	1
3	Number of Customers	null	Net Sales	null	Profit	null	2
4	4	null	984	null	625.28	null	3
5	Order Number	null	Product Sold	null	Sales Person	null	4
6	13663	null	Dyno3	null	John	null	5
7	Number of Customers	null	Net Sales	null	Profit	null	6
8	8	null	2253	null	653.2	null	7
9	Order Number	null	Product Sold	null	Sales Person	null	8
10	13682	null	Dyno3	null	John	null	9

Figure 6:20 – Working out the number

This is where the modulo comes into effect as we can use this function and tell Power Query how many rows there are before the function repeats itself. In our case, there will be **4** as there are four rows before the data repeats itself:

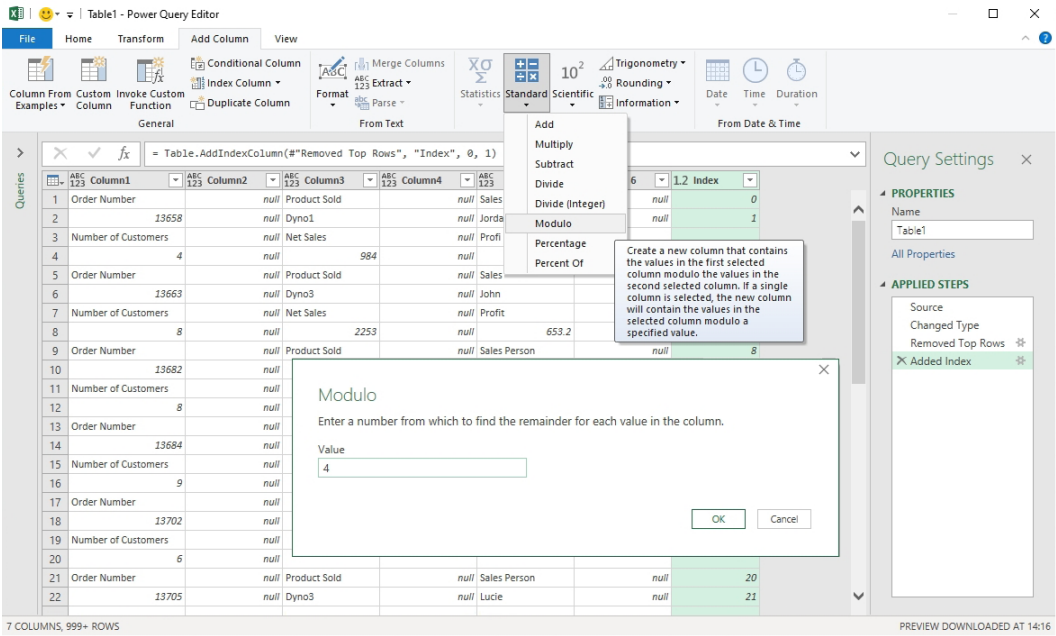


Figure 6:21 – Insert the Modulo column

When looking at the **Modulo** column, you will notice how each new record starts at number **0**. The other important thing is that each number is going to represent a particular item, for example, the **Number of Customers**, as shown in the following screenshot:

Table1 - Power Query Editor

File Home Transform Add Column View

Column From Examples Custom Column Invoke Custom Function General

Conditional Column Index Column Duplicate Column

Format Parse From Text

Merge Columns Extract From Number

Statistics Standard Scientific

Trigonometry Rounding Information

Date Time Duration From Date & Time

Queries

Table.AddColumn("#Added Index", "Modulo", each Number.Mod([Index], 4), type number)

	Column1	Column2	Column3	Column4	Column5	Column6	1.2 Index	1.2 Modulo
1	Order Number	null	Product Sold	null	Sales Person	null	0	0
2	13658	null	Dyno1	null	Jordan	null	1	1
3	Number of Customers	null	Net Sales	null	Profit	null	2	2
4	4	null	984	null	625.28	null	3	3
5	Order Number	null	Product Sold	null	Sales Person	null	4	0
6	13663	null	Dyno3	null	John	null	5	1
7	Number of Customers	null	Net Sales	null	Profit	null	6	2
8	8	null	2253	null	653.2	null	7	3
9	Order Number	null	Product Sold	null	Sales Person	null	8	0
10	13682	null	Dyno3	null	John	null	9	1
11	Number of Customers	null	Net Sales	null	Profit	null	10	2
12	8	null	1528	null	573.6112	null	11	3
13	Order Number	null	Product Sold	null	Sales Person	null	12	0
14	13684	null	Dyno3	null	Clive	null	13	1
15	Number of Customers	null	Net Sales	null	Profit	null	14	2
16	9	null	2592	null	857.4336	null	15	3
17	Order Number	null	Product Sold	null	Sales Person	null	16	0

Figure 6:22 – The modulo added

We are going to use an IF function and the **Modulo** column to create additional columns for each of the fields that we would like, for example, **Order Number**, **Product Sold**, and so on.

We are going to start this by creating a new custom column by selecting **Custom Column** in the **Add Column** tab. The first thing we want to find is **Order number**, so we can use this as the **New column** name. In our **IF** statement, we are saying that if the modulo number is 1, then this is the customer number.

The code that we will write is as follows:

```
= if [Modulo]=1
then [Column1]
else null
```

When we click **OK**, we notice that we now only have a number in this column for the order number. We are now going to do exactly the same process for the other columns, namely, **Product** and **Sales Person**. The **Product Sold** formula would be as follows:

```
if [Modulo]=1
then [Column3]
else null
```

This will give us the following result:

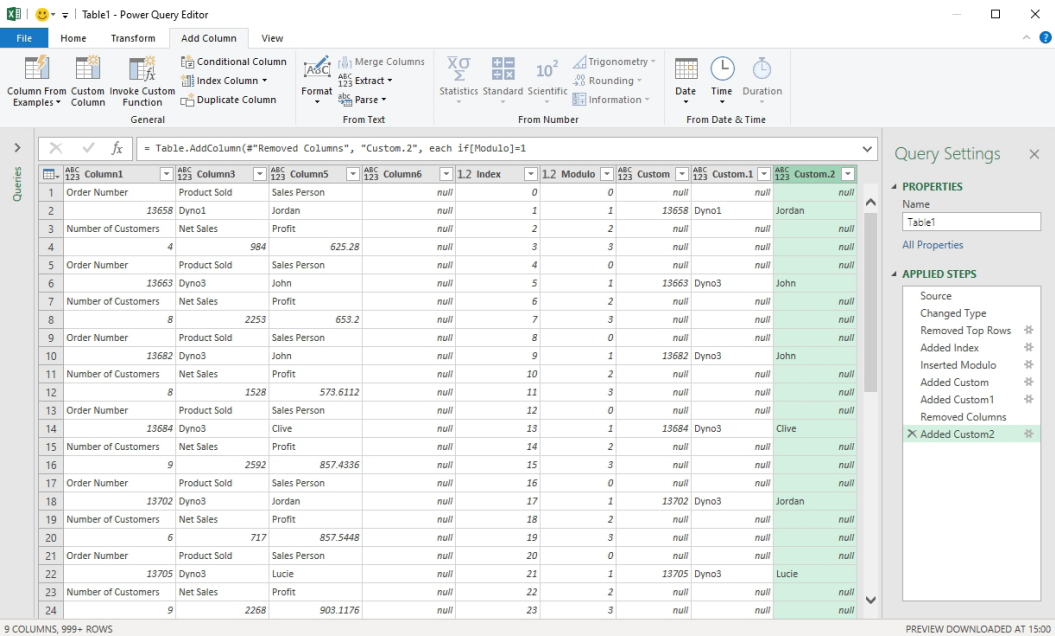


Figure 6:23 – The first three columns

When we continue with this, there will be a slight change for the next three columns; we will change the Modulo number to 3, instead of 1, as we are looking at a different row. Our formula for the **Number of Customers** will be as follows:

```
if [Modulo]=3
then [Column1]
else null
```

If at any time you have made a mistake, you can delete the step and then redo it, but it is easier to click on the gear icon on the right of the applied step and then change the value in the **Add Conditional Column** window:

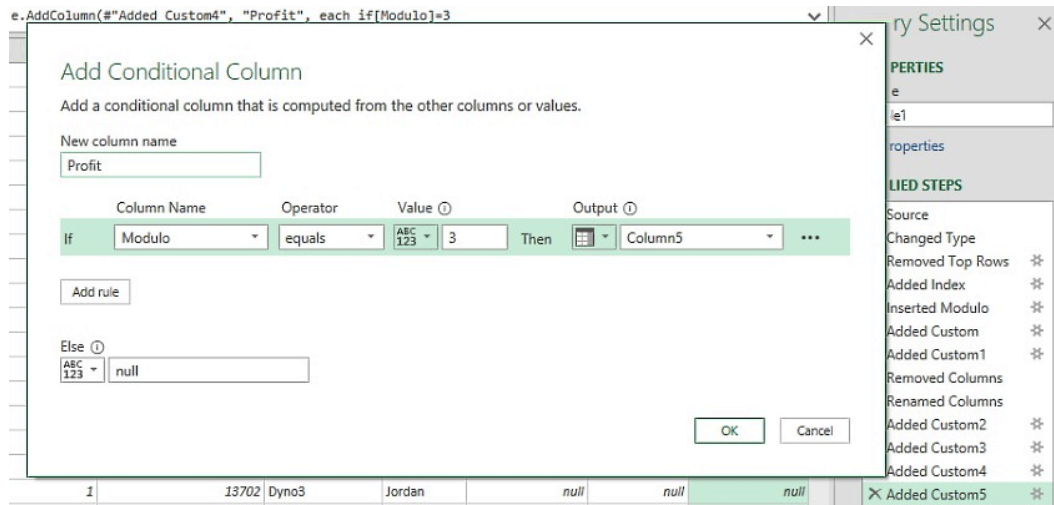


Figure 6:24 – The Add Conditional Column window

Once we have completed everything, we will have all of the additional columns that we now require:

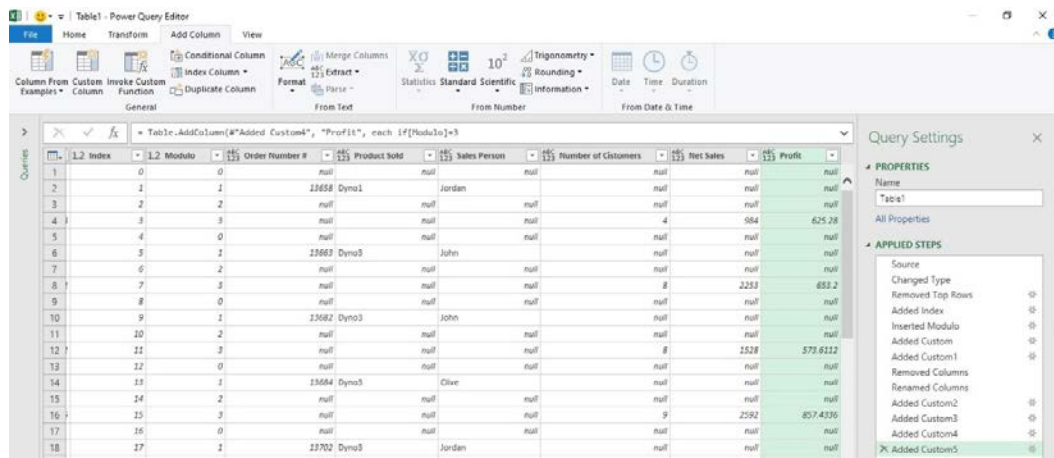


Figure 6:25 – The completed additional columns

At this stage, we have all of the columns we need, but we will need all of the information in a specific row. We are going to try and get all of the information in one of the modulo rows.

To make our lives easier, we can rename the **APPLIED STEPS** by right-clicking on each applied step, for example, **Added Column5**, and then selecting **Rename**:

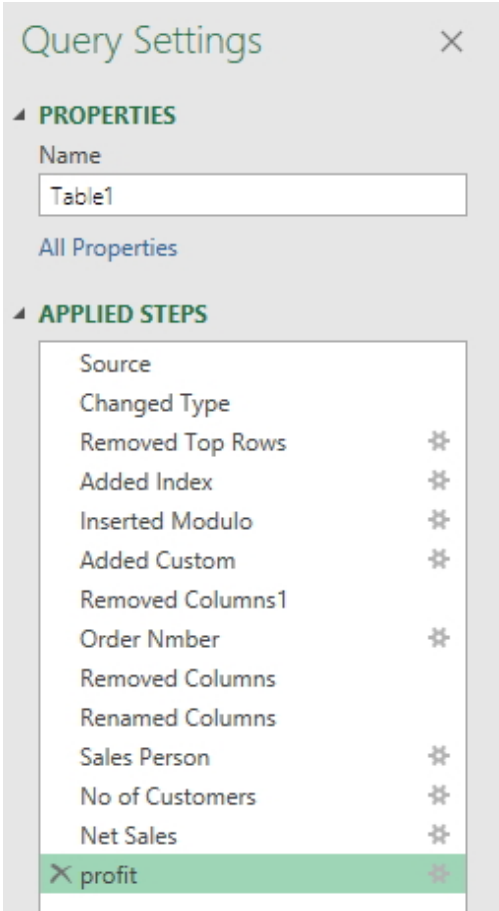


Figure 6:26 – Renaming the different steps

Select all of the new **if** functions columns. (You have to press and hold *Ctrl* to select multiple columns.) Then, right-click and select **Fill | Down**, as shown in the following screenshot:

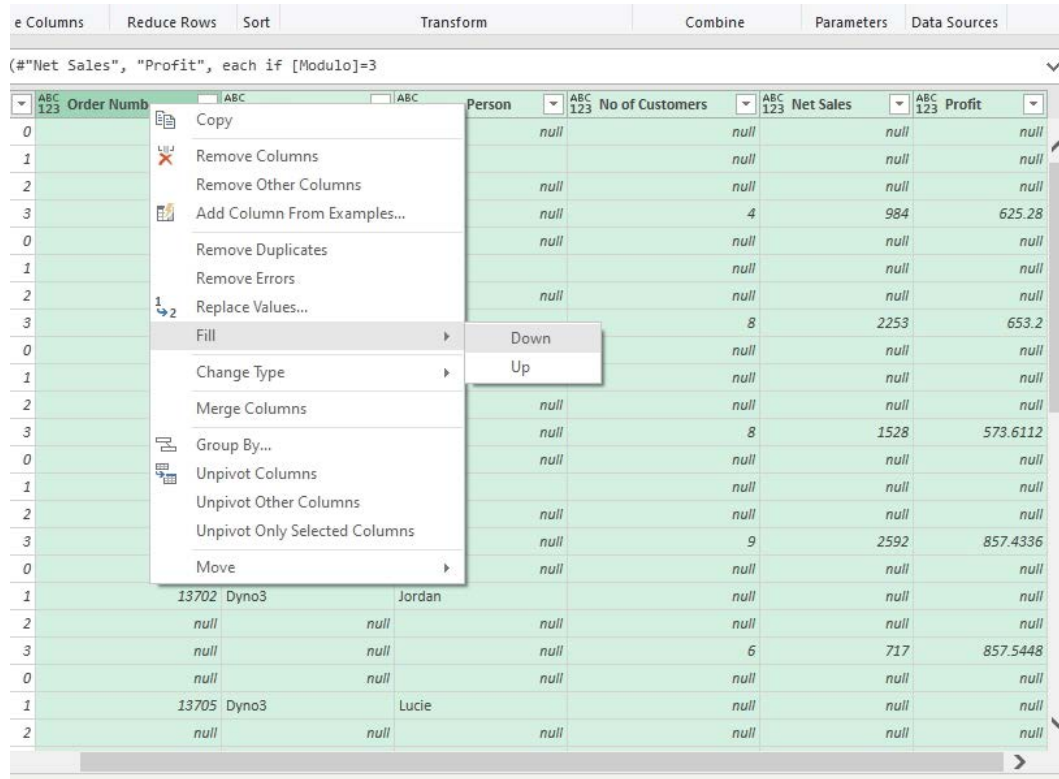


Figure 6:27 – The Fill Down menu

As we only need the information that is in Modulo row 3, we can filter the Modulo row to show just column 3, as follows:

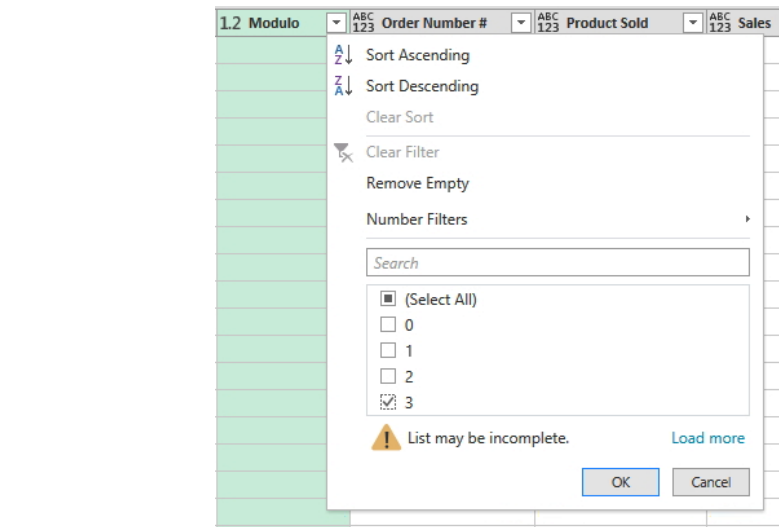


Figure 6.28 – Filtering the Modulo row

We can now select the columns that we want to show by selecting **Choose Columns** on the **Home** tab and selecting the new columns we have created:

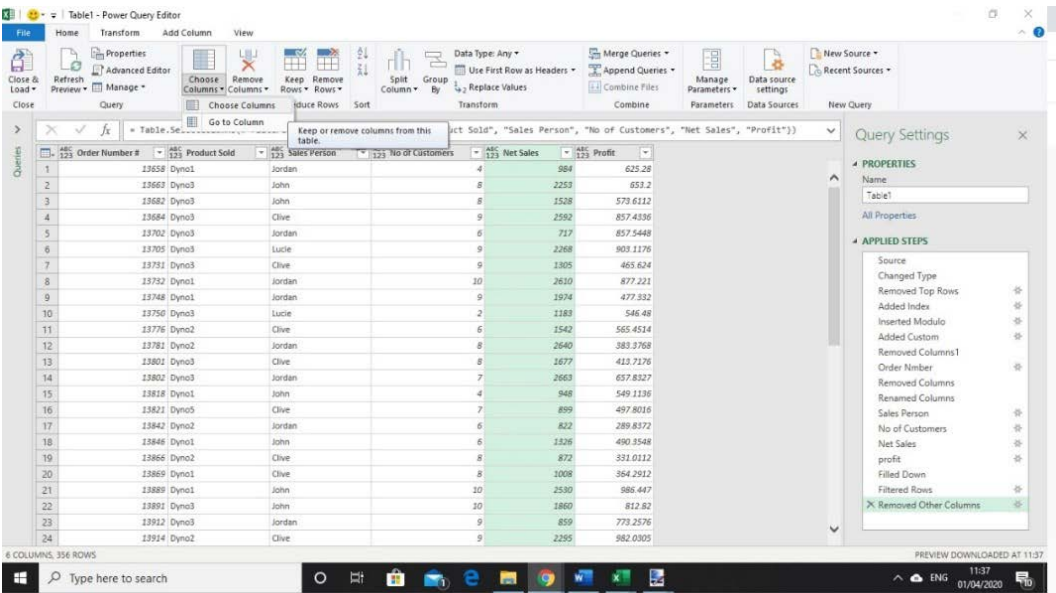
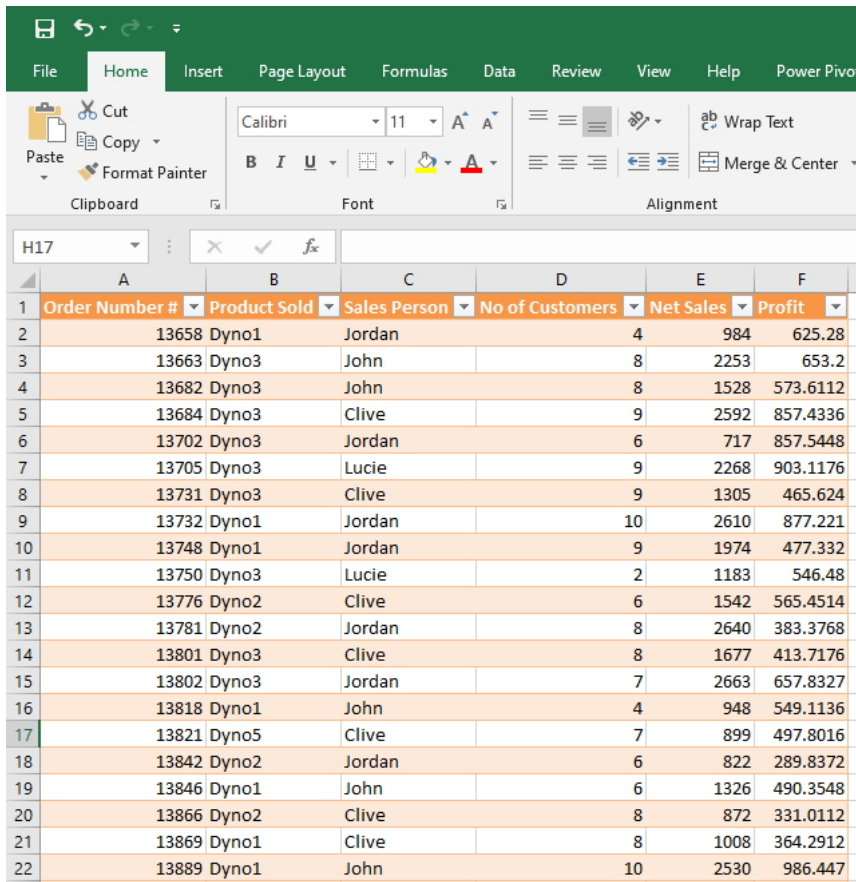


Figure 6.30 – The filtered rows

Once completed, we can click **Select and load** or select and **Load to ...**. When you look at the following completed table, we can see that we can work with it easily and analyze the data without any problems. Of course, for data from the following month, all we have to do is change the source and everything will already be done:



	A	B	C	D	E	F
	Order Number #	Product Sold	Sales Person	No of Customers	Net Sales	Profit
2	13658	Dyno1	Jordan	4	984	625.28
3	13663	Dyno3	John	8	2253	653.2
4	13682	Dyno3	John	8	1528	573.6112
5	13684	Dyno3	Clive	9	2592	857.4336
6	13702	Dyno3	Jordan	6	717	857.5448
7	13705	Dyno3	Lucie	9	2268	903.1176
8	13731	Dyno3	Clive	9	1305	465.624
9	13732	Dyno1	Jordan	10	2610	877.221
10	13748	Dyno1	Jordan	9	1974	477.332
11	13750	Dyno3	Lucie	2	1183	546.48
12	13776	Dyno2	Clive	6	1542	565.4514
13	13781	Dyno2	Jordan	8	2640	383.3768
14	13801	Dyno3	Clive	8	1677	413.7176
15	13802	Dyno3	Jordan	7	2663	657.8327
16	13818	Dyno1	John	4	948	549.1136
17	13821	Dyno5	Clive	7	899	497.8016
18	13842	Dyno2	Jordan	6	822	289.8372
19	13846	Dyno1	John	6	1326	490.3548
20	13866	Dyno2	Clive	8	872	331.0112
21	13869	Dyno1	Clive	8	1008	364.2912
22	13889	Dyno1	John	10	2530	986.447

Figure 6.31 – The completed table

You will agree that this is much easier for us to analyze than the spreadsheet that we started with. Although it took us a little while to set up, it will save us hours every time we have to redo this. This is the beauty behind Power Query: once you have set it up, everything else can be automated.

Lastly, if there are any steps performed incorrectly, we can always delete them.

Up to this point, we have been looking at merging existing files together, but sometimes we will want to append files, which is slightly different from what we have been doing before this. In the next section, we will see how to append them.

Appending multiple files

There are two different ways in which we can join tables in Power Query or Power BI, namely, **merge** or **append**. In computer science, *append* means to add to an existing file and not overwrite anything that came before.

This goes beyond here, as even in our merge queries that we have done before this, we have had a primary or relationship key that has allowed us to create new columns or fields. When we append something, it comes underneath everything else that has gone before.

In Power Query or Power BI, when we append files, we will end up with a query that is made up of two or more queries. As an example, we have the following two different tables in different workbooks. We are going to append these files so that they are underneath each other:

	A	B	C	D	E	F	G	H	
1	Number	Region	Product	Date	Sales	Discount%	Discount	Total	
2	1	North	Game5	43555	41564.56	9.62%	3998.843	37565.71	
3	2	North	Game1	43555	41433.35	0.30%	122.3973	41310.95	
4	3	North	Game4	43555	41426.51	0.29%	121.8273	41304.68	
5	4	North	Gam						
6	5	West	Gam						
7	6	North	Gam						
8	7	West	Gam						
9	8	East	Dyn						
10	9	South	Prod						
11	10	North	Gam						

	A	B	C	D	E	F	G	H	
1	Number	Region	Product	Date	Sales	Discount%	Discount	Total	
2	1	North	Dyno5	43555	9786.894	5.18%	506.7376	9280.157	
3	2	West	Game2	43555	9700.701	9.40%	911.9569	8788.745	
4	3	North	Game6	43555	9953.764	5.38%	535.4767	9418.287	
5	4	North	Game2	43555	9545.53	3.31%	316.2927	9229.237	
6	5	North	Game2	43555	9692.732	9.26%	897.4193	8795.313	
7	6	North	Game5	43555	9616.868	1.77%	170.4682	9446.4	
8	7	North	Board4	43555	9635.518	4.79%	461.1172	9174.401	
9	8	East	Board1	43555	8933.086	4.68%	417.7248	8515.361	
10	9	East	Game9	43555	9101.107	3.50%	318.1835	8782.923	
11	10	South	Dyno4	43555	9847.687	4.73%	465.906	9381.781	

Figure 6.32 – Two different tables to be appended

We are also going to go a step further in this example, as we would like to load the two Excel files, but we will also load a CSV file. Let's see how to do this:

1. Open Power BI and load the queries into it and then select any of the queries to open. It does not make any difference what type of data source you are loading when adding to the data model.
2. On the top-right, click on **Append Queries**, which is below the **Merge Queries** tab:

Completed PowerBI Append files - Power Query E

File

Home

Transform

Add Column

View

Tools

Help

Formula Bar

Monospaced

Column distribution

Show whitespace

Column profile

Column width

Go to Column

Data

Parameters

Advanced Editor

Query Dependencies

Dependencies

Layout

Data Preview

Parameters

Advanced Editor

Query Dependencies

Dependencies

Queries [3]

SA

UK

US March 2020

Table.TransformColumnTypes(*Promoted Headers),{{"Region", type text}, {"Product", type text}, {"Department", type text}, {"Sales", type number}, {"Discount", type number}}

	Region	Product	Department	Date	Sales	Discount
1	Cape	Game5	8	43555	42564.55705	0.02079
2	Cape	Game1	7	43555	42433.35082	0.08989
3	Cape	Game4	3	43555	42426.5121	0.03682
4	Cape	Game3	9	43555	42382.96997	0.06292
5	Jo'burg	Game6	8	43555	42308.45019	0.04685
6	Cape	Game1	4	43555	42180.58512	0.07783
7	Jo'burg	Game3	8	43555	42122.64075	0.05057
8	East	Dyn04	6	43555	42002.03092	0.05627
9	South	Prod T	6	43555	41954.25852	0.04755
10	Cape	Game8	2	43555	41479.47686	0.02703
11	Cape	Dyn05	2	43555	41817.90334	0.06438
12	South	Dyn05	2	43555	41494.81322	0.06483
13	South	Board5	4	43555	42116.11079	0.02420
14	Jo'burg	Game1	2	43555	42194.42764	0.00982
15	Cape	Game7	8	43555	42139.67761	0.02394
16	South	Board4	10	43555	42111.96197	0.0809
17	Cape	Board4	8	43555	42580.89085	0.02656
18	South	Dyn04	2	43555	42478.74161	0.06123
19	Jo'burg	Game5	3	43555	42478.95125	0.03027
20	Cape	Game3	4	43555	42510.21034	0.04510
21	Cape	Game10	7	43555	42716.81589	0.06377
22	South	Game8	10	43555	42686.61824	0.07458
23	South	Game10	5	43555	42905.23031	0.07690
24	South	Game8	4	43555	42721.07824	0.05086
25	Cape	Game1	8	43555	42828.63064	0.01779
26	Cape	Game1	7	43555	42788.20828	0.09
27	Cape	Game5	7	43555	42357.02612	0.09565

Query Settings

PROPERTIES

NameSA

All Properties

APPLIED STEPS

Source

Navigation

Promoted Headers

Changed Type

8 COLUMNS, 909+ ROWS

Column profiling based on top 1000 rows

PREVIEW DOWNLOADED AT

Figure 6.33 – The Merge Queries and Append Queries tabs

The option is to either **Append Queries** or **Append Queries as New**. The difference is **Append Query as New** retains the query result as it is already loaded and allows you to create a new query based on this, while **Append Queries** starts the query with no steps included. As we already want the tables with the steps, we are going to select **Append Query as New**.

- After selecting, the **Append** window will be displayed as follows:

Figure 6.34 shows the Append window. The window title is "Append". It contains the text "Concatenate rows from two tables into a single table." Below this, there are two radio buttons: "Two tables" (selected) and "Three or more tables". There are two dropdown menus: "Primary table" (set to "US March 2020") and "Table to append to the primary table" (empty). At the bottom right are "OK" and "Cancel" buttons.

Figure 6.34 – The Append window

Depending on how many queries you would like to append, there is a difference in how the process works. If you only need to append two queries, then we can select the primary and other tables that we require.

The primary key is normally the table or query that you selected first, but it could also be the main table that you would like to be on top. Refer to the following screenshot:

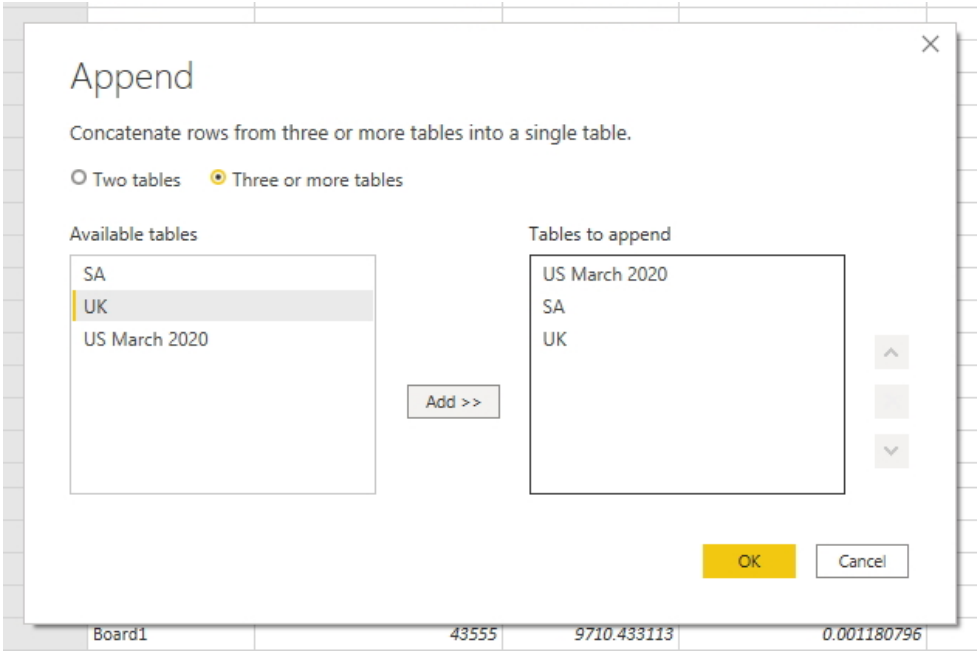


Figure 6.35 – Select the tables to append

As we see in the preceding screenshot, we are going to append three different tables so that we can go through the more advanced options, which is only one step.

Select the different tables on the left and then add them by clicking on the **Add** button, which will move the tables to the right:

We have the option of either deleting this column or creating a merge with the other queries. **Append queries** does not remove duplicate rows. So, we would need to use **Group By...** or **Remove Duplicates** by right-clicking to get rid of duplicates, shown as follows:

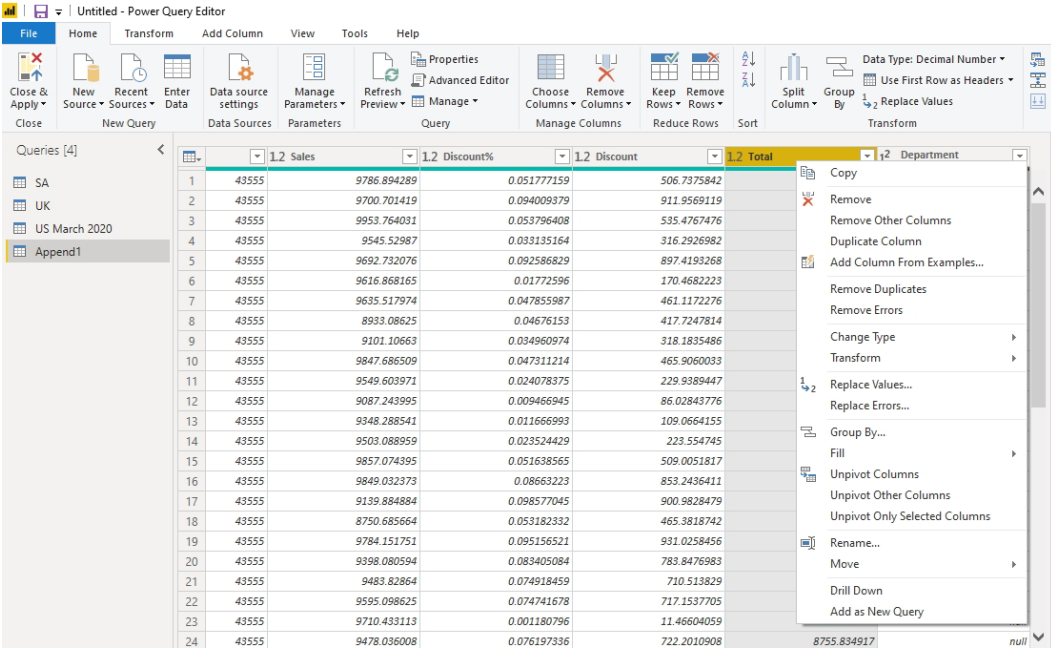


Figure 6.38 – Use of Remove Duplicate and Group By...

Note

Power Query and Power BI use the column names to combine the data from the different tables. Your first table is the primary or the template that is used to combine the other tables. If the other tables have more columns than the template, they will be ignored and if the other tables do not have the same columns as the template, Power Query and Power BI will add null to this row.

The columns do not need to be in the same order as Power Query and Power BI use the column names to merge or append the data.

We can now **Close and Apply** and load this into Power BI for this to be easily analyzed:

Region	Product	Date	Sales	Discount%	Discount	Total	Department
Jo'Burg	Game1	43547	41074.6837588144	0.0694155149866858	2851.22032603336	38223.463432781	5
Jo'Burg	Board5	43547	41050.60425758	0.0832525110517979	3417.56588463716	37633.0383729428	5
Jo'Burg	Board3	43547	41038.5693264093	0.00955123118486528	391.968863132656	40646.6004632766	5
Jo'Burg	Game9	43548	41233.0899304035	0.00598262953523814	246.682301646762	40986.4076287567	5
Jo'Burg	Game2	43548	41466.0291446053	0.0138135021459522	572.791082573123	40893.2380620322	5
Jo'Burg	Game8	43548	40692.447867166	0.0553638185767409	2252.88930116126	38439.5585660047	5
Jo'Burg	Game5	43548	40774.2371869393	0.0712903268684009	2906.80869686661	37867.4284900727	5
Jo'Burg	Game6	43548	41162.0333597948	0.0080756139240276	332.408689741648	40829.6246700531	5
Jo'Burg	Board5	43549	41436.3522849076	0.0742039533872758	3074.74115348802	38361.6111314196	5
Jo'Burg	Game5	43549	40986.117379111	0.00910033733603833	372.987494244373	40613.1298848666	5
Jo'Burg	Dyno5	43549	40984.280290364	0.0531420591364042	2177.98904685349	38806.2912435105	5
Jo'Burg	Dyno1	43549	41307.369579799	0.0677146500559316	2797.11407582712	38510.2555039719	5
Jo'Burg	Board2	43549	40992.5570119977	0.0020181980548036	82.7310988230394	40909.8259131747	5
Jo'Burg	Dyno2	43549	41124.2075731743	0.0557627820930374	2293.20022565176	38831.0073475225	5
Jo'Burg	Game8	43549	41091.4620286749	0.0157360545393125	646.617487583316	40444.8445410916	5
Jo'Burg	Prod T	43549	41167.6890096066	0.0795970044909948	3276.82472693807	37890.8642821225	5
Jo'Burg	Game4	43550	41093.1040686266	0.05304122537144	2179.62859411606	38913.4754745105	5
Jo'Burg	Game8	43550	41094.4703735025	0.0608508889767596	2500.63299953322	38593.8373739693	5
Jo'Burg	Board5	43550	41041.4441454142	0.0327820144180145	1345.4212137111	39696.0229317031	5
Jo'Burg	Dyno5	43550	41323.2769770233	0.0756933893536719	3127.89889359145	38195.3780834318	5
Jo'Burg	Game1	43551	41572.6489124486	0.0831437781629514	3456.50709882289	38116.1418136257	5
Jo'Burg	Prod T	43551	41513.3414015019	0.00916014203318555	380.268109509879	41133.073297992	5
Jo'Burg	Board2	43551	41162.9298100761	0.06605123254478	2718.86224910979	38444.0675609663	5

Figure 6.39 – Completed Append1 data

When we append files, Power BI or Power Query looks at the column names that are the same. If it finds a column name that is the same, then it adds all of the different rows' data into this column. If there is a column that does not match or is different (such as our **Department** column), then Power BI will create a new column for this row.

We need to be careful with columns that might appear to be the same, as a column could have a blank space, which makes the name of the column different. For example, SalesPerson is not the same as Sales Person. The other important thing to remember is that the columns are also case sensitive.

Up to this point, we have used multiple files to get our data, but in the next section, we are going to look at how we can append one file that has multiple tabs of data.

Appending multiple tabs

In this section, we are going to look at how we can append multiple tabs from the same workbook. Although there are different ways in which we can do this, we are going to use M code to append the tables together.

We are going to open a workbook with sample sales data (sample1.xlsx) for different parts of South Africa. Each table is on a separate worksheet in the workbook and each table is the name of the place (Cape Town, Durban, and so on):

1. To append the different sheets, we will create a new **Blank Query** by selecting this from **From Other Sources** in the **Data** tab.
2. In the formula bar, I am going to use the `=Excel.CurrentWorkbook()` formula, which will show me all of the tables in the Excel workbook. Please remember that the formula is case sensitive:

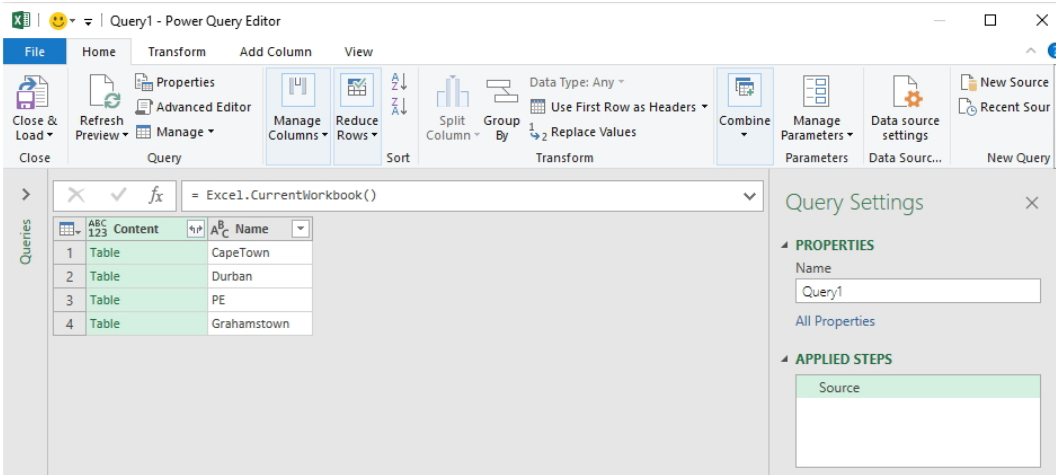


Figure 6.40 – The show tables formula

Looking at the formula, it tells us to look at the current workbook and then it finds the tables, name ranges, and connections, which are displayed. When this is displayed, you will see that there is a table associated with each name.

3. To open the table, we can click on **Table** on the left-hand side of each of the different places, which will then open it:

	Region	Product	Date	Department	Sales	Discount%
1	Cape	Game5	43555	8	43222	0.0
2	Cape	Game1	43555	7	32294	0.0
3	Cape	Game4	43555	3	34419	0.0
4	Cape	Game3	43555	9	40969	0.0
5	Jo'Burg	Game6	43555	6	28876	0.0
6	Cape	Game1	43555	4	40228	0.0
7	Jo'Burg	Game3	43555	8	37678	0.0
8	East	Dyno4	43555	6	27282	0.0
9	South	Prod T	43555	6	31487	0.0
10	Cape	Game8	43555	2	37008	0.0
11	Cape	Dyno5	43555	2	34563	0.0
12	South	Dyno5	43555	2	29772	0.0
13	South	Board5	43555	4	36975	0.0
14	Jo'Burg	Game1	43555	2	35432	0.0
15	Cape	Game7	43555	6	28922	0.0
16						

Figure 6.41 – The expanded table

- For us to go back to our source table, we will have to delete the two **APPLIED STEPS** we created in viewing the table.

Power Query is holding each of these tables in its data model:

	Content	Name
1	Table	CapeTown
2	Table	Durban
3	Table	PE
4	Table	Grahamstown

Figure 6.42 – Append and merge the data

5. As seen in the preceding screenshot, we will need to click on the double-headed arrow in the **Content** column, which will give us an option of which columns we would like to combine:

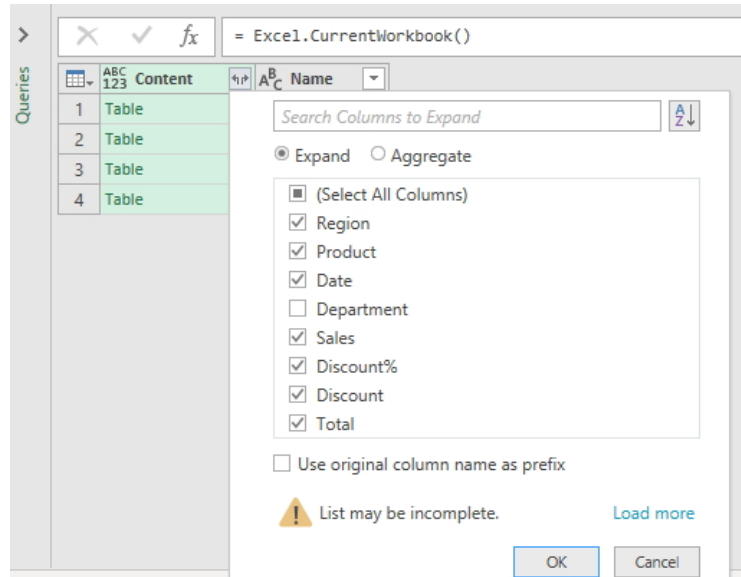


Figure 6.43 – Select the columns we require

6. We are going to unselect **Department**, as only **Cape Town** used this feature, and we are going to also unselect **Use original column name as prefix**.

The reason for this is that Power Query will add the column name **content** to all of the columns. This means that the **Region** column name would become **content.Region**. Once selected, click on **OK**.

7. Go through the sheet and make sure that all of the data types are correct and then click **Close and Load**. This will create a new sheet in our existing workbook.

For the last part of this section, I am going to create another worksheet called **Pretoria** and add this to the workbook. As I refresh the data in my query, everything automatically refreshes and I have the most up-to-date data. In this section, we have appended and merged the data from a data source and we have seen that, when we change or update the data source, Power Query automatically creates new queries based on either new files in a folder or new tabs in an Excel document.

I know that many people work with multiple workbooks, but most of the data I generally work with is summarized into one workbook with multiple tabs. I find that this is the easiest way of keeping my data up to date.

Summary

Throughout this chapter, we have concentrated on more advanced functions, using **IF**; creating **parameter tables** and, my personal favorite, using **Index** and **Modulo** functions; and lastly, appending files and tabs.

We have seen that when using the **IF** statement, it was relatively similar to Excel, but when we used the nested **IF** statement when deciding on whether something was to be picked up in the store or to be delivered the **OR** command became very important. The parameter tables are great if you are using the same template every month and all you want to do is change the data source from one file to another. Changing the table in Excel allows you to update the data source without leaving Excel and going into Power Query Editor, which saves you time. Next, we saw that the **Index** and **Modulo** functions are really useful when you have an Excel document that does not have the correct formatting or perhaps has data that is not even in columns. Using these functions allows you to take that data and create custom indexing columns that allow you to re-order and manipulate the data so that it can be more easily used to be analyzed. In the last part of this chapter, we looked at how it was possible to append different files and tabs as well as different tables in the same sheet so that it was easier every month to update our existing data sources.

Throughout this chapter, we have started looking at the M language and the difference between Excel and M. Two of the main things is that M uses field names, which are in square parentheses, [], while Excel uses formulae mostly with round parentheses, (). This will be discussed in more detail in *Chapter 9, Working with M*.

There are still many more advanced formulae, tips, and tricks, which will be covered in the remainder of this book. The next chapter deals with the automating reports, which will become very useful when you need to make sure that everything is refreshing properly and if there is a problem, we will be looking at the source code to find any underlying problems.

7

Automating Reports in Power Query

Power Query provides options to streamline and automate reports from multiple sources. In this chapter, we will look at the three types of storage modes and discover the pros and cons of each mode. We will also create a report from multiple files that will update every month.

In this chapter, we're going to cover the following main topics:

- Storage modes and dataset types
- Choosing the **Import** storage mode setting
- Power BI refresh types

By the end of this chapter, you will understand how to automate reports and how they prove useful when using Excel and Power Query.

Technical requirements

It is assumed in this chapter that you are familiar with the Windows environment and can view system processes by visiting **Task Manager**. You will also be proficient at importing various data sources into Power BI and you should have prior knowledge of the interface and be able to navigate it with ease. You need an internet connection to download the relevant files from GitHub. Also, the code files for this chapter can be found at <https://github.com/PacktPublishing/Learn-Power-Query>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=KbDnA3WPjQ8&list=PLeLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=8&t=0s.

Understanding the storage modes and dataset types

Different refresh requirements are available for the storage modes and dataset types within Power BI. To view any sorts of changes that have occurred, you can reimport the data into Power BI, or there is the option to query the data directly at the source.

When using the Power BI Desktop version, you may want to set the storage mode so that you can manage how Power BI Desktop caches your table data in memory for any reporting outputs you create. Let's investigate a few options that would make setting the storage mode an advantage:

- Look at reducing the refresh times so that you only cache the data that is essential to meet the requirements of your business.
- The report query performance in Power BI is definitely enhanced by ensuring that data is cached into memory properly. This is especially crucial when users view and access reports when DAX queries are submitted to datasets.
- When working with a large dataset, you can choose which tables you would like to cache and which don't need to be cached. This will ensure that you do not use up memory unnecessarily.
- User input editing can be altered with changes displaying directly within uncached tables.

Now that you understand how setting the storage mode could benefit data refresh, in the next topic, we will look at how to set the storage mode.

Viewing the Power BI Desktop Storage mode setting

Let's have a look at how we would view and set the **Storage mode** property in Power BI. The storage mode is set on individual tables in the Power BI model. We use this setting to control how Power BI caches data in tables within the model:

1. Click on the table for which you would like to set the storage mode in the **Model** view. The **Model** view is the third icon on the left-hand side panel of your Power BI Desktop instance, as in the following screenshot:

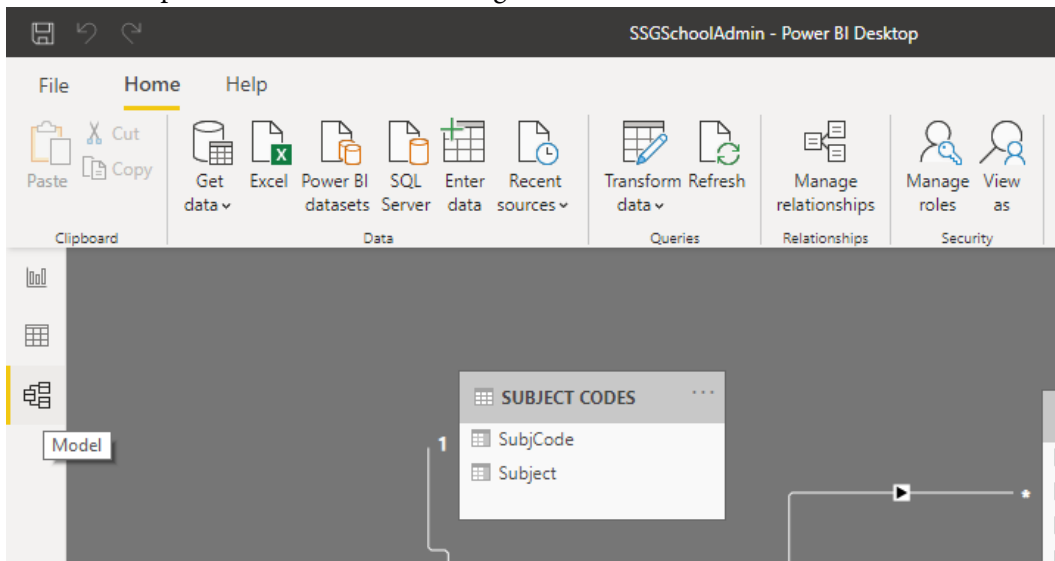


Figure 7.1 – A selected table in the Model view to make changes to the storage mode

2. The **Properties** pane to the right of the window will display the **Advanced** drop-down list heading near the bottom of the pane.

3. Locate the **Storage mode** heading and then click on the arrow alongside the **Import** option to view all the other options in the list:

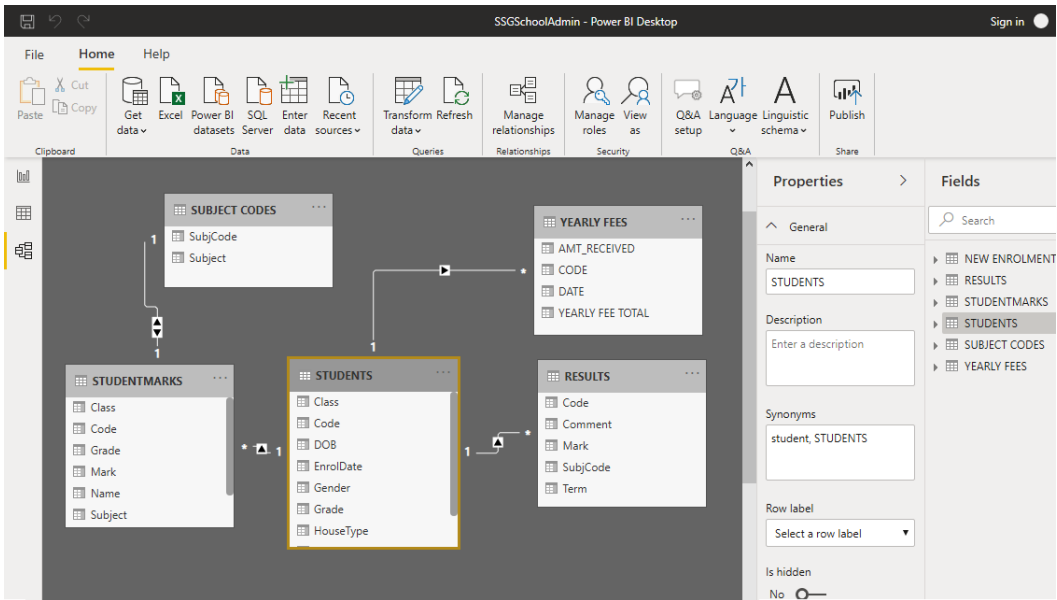


Figure 7.2 – The storage modes at the bottom of the Properties pane

4. You will note that there are three options to choose from under **Storage mode**:

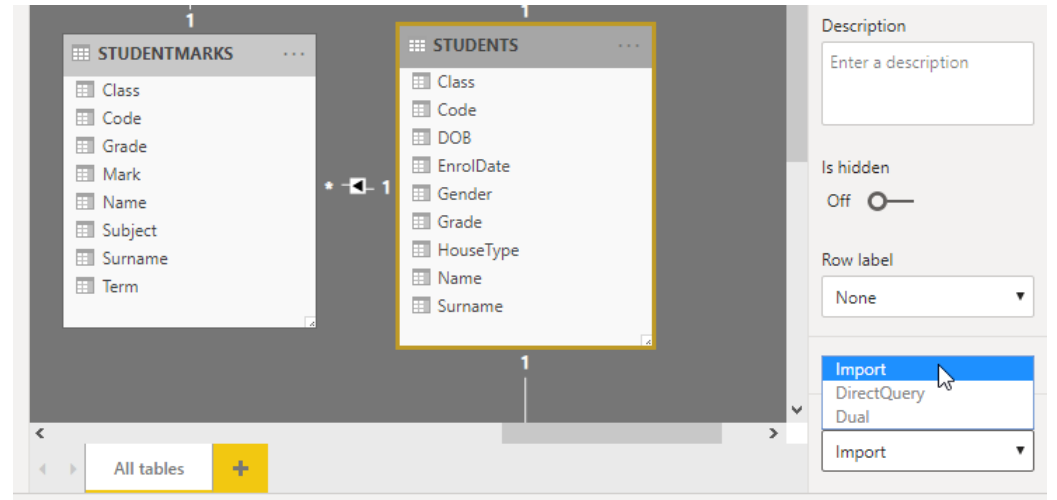


Figure 7.3 – Import, DirectQuery, and Dual Storage modes

We can say the following about the three modes:

- **Import:** When this setting is active, any imported tables are cached and are loaded into the data model. This ensures that data is returned through the imported tables when queried through the Power BI dataset. **Import** is the only type of connection mode that you can set regardless of the type of data source. Please note that when using this option, you cannot revert the action once chosen. By this, we mean that you will not be able to change to another type of **Storage mode** in the drop-down list, such as **DirectQuery**, after setting the mode to **Import**. In the preceding screenshot, you will notice that the **DirectQuery** and **Dual** options are grayed out. This means that they are not available in this instance as the **Import** method pertains to Excel files, and, therefore, will be the only option available in the **Storage mode** drop-down list.
- **DirectQuery:** Tables are not cached in this instance, and any queries submitted to a Power BI dataset will use the query language for that particular data source to send data from the **DirectQuery** tables. This method can be slower as it sends visuals. The plus side is that the single data source does not need to be refreshed. DAX, modeling, and Power Query transformations are limited, however, with this mode.
- **Dual:** This option is flexible and allows tables to be cached or not cached so that they are able to fulfil requirements from either cached data or by actioning an on-demand query using the query language for that particular data source. An example of a query language is SQL. This mode can be used in Power BI for Desktop and the Power BI service. We use this mode to improve performance and limit the number of poor relationships caused by any of the **DirectQuery** tables in a dataset. We refer to models that contain more than one type of connection as composite models. A composite model has a mix of connections—for example, the **DirectQuery** storage mode and the **Import** storage mode. With the **DirectQuery** mode, data is not loaded into Power BI but is queried directly from the table in the SQL server database, whereas the imported tables from Excel will need to be refreshed each time.

I should just mention here that when we refer to the storage mode, we mean the type of connection that we create to a data source. So, we can import many data types into Power BI, and then we can create connections to the source data using either the **Import**, **DirectQuery**, or **Dual** property. Some connections can be set using more than one **Storage mode** option, while others cannot.

Note

Storage mode selection is a very important step, and deciding on which connection type to use must be considered at the start. This will alleviate complications and any hassle down the line of having to repeat the import or connection steps, as well as repeating all the other elements that contribute to the success of your data transformation. We cannot change the **Storage mode** setting during the Power BI cycle process.

In this section, you have learned about the different types of storage modes and how to set each connection type. In the next topic, we will see how to view the connection type applied to each table in a dataset.

Viewing the storage mode at a glance

You can view the storage mode for each imported element using the **Fields** pane on the right-hand side of the Power BI interface once you have imported data into Power BI. Let's see how:

1. Navigate to the **Fields** pane.
2. Place the mouse pointer over any of the imported elements in the list.
3. You will notice a popup to the left of the element that you have hovered the mouse pointer over, giving details about the import and the **Storage mode** type, as follows:

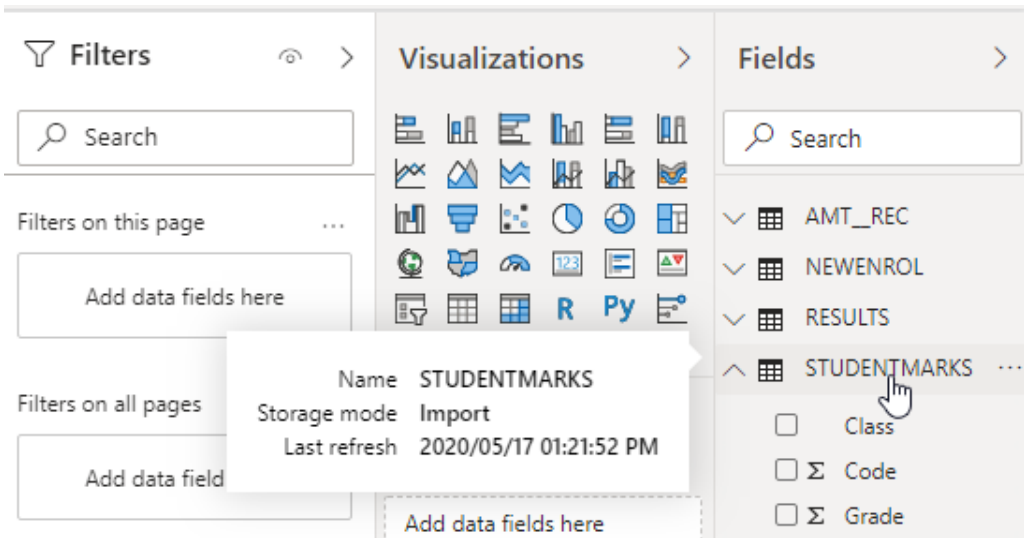


Figure 7.4 – Hovering over a table to view the Storage mode type

From the preceding screenshot, you can see that we went through the steps to visibly see the type of storage mode applied to tables within a dataset. In the next section, we will learn how to set the **Import** storage mode.

Choosing the Import storage mode setting

So far in this book, we have imported a number of datasets, but we have not really broken down the steps to refine the process as we proceeded through the steps. Let's refresh our memory by looking at the various steps again, but this time highlighting the **Import Data** option:

1. Open Power BI Desktop, and then click on **Get Data | Excel**.
2. Select the `SSGSchool.xlsx` file to import.
3. Choose the tables you wish to import. In this case, we will select the tables we want by clicking on the squares next to each one to select them individually:

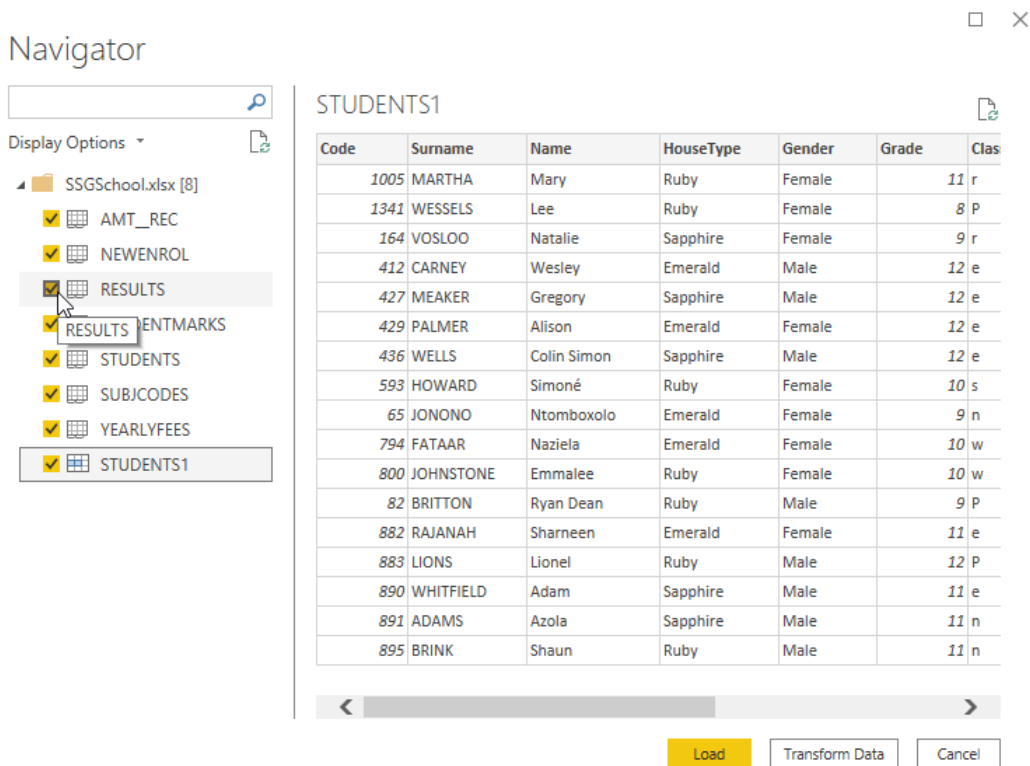


Figure 7.5 – Selecting the tables to import using the checkboxes alongside each element

4. Click on **Load** to import the dataset.
5. You will see the **Load** screen, where the searched relationships are present, before importing:

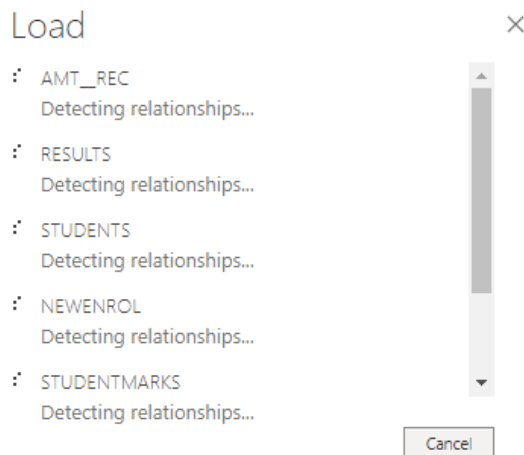


Figure 7.6 – The Load table screen will display once you have clicked on Load

6. If you were importing any SQL connections, the following screen would appear, where the **Import** option for the **Data Connectivity mode** setting will be selected as the default:

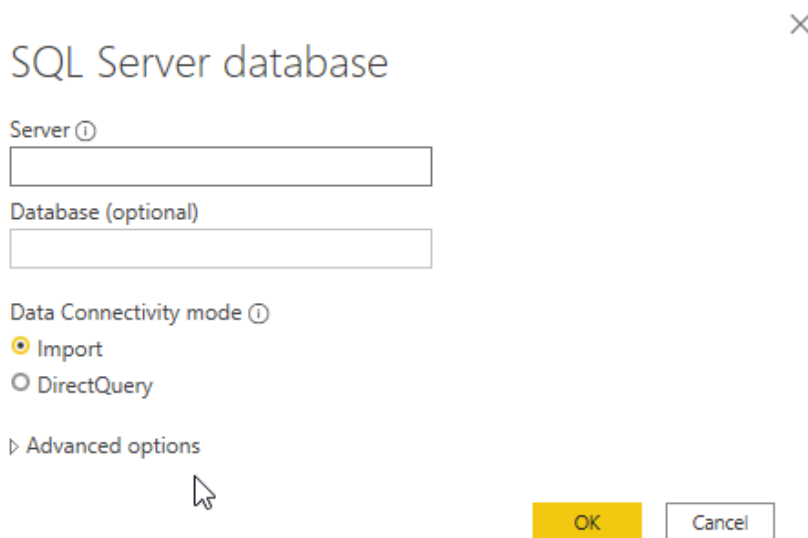


Figure 7.7 – The SQL Server database Import option

If you change this to the **DirectQuery** option, then the tables will not be imported into Power BI. So, the process is the same, except that for the **DirectQuery** option, after logging in to the database, you will see the tables you have access to, but will only have a connection to the data, which is then updated by querying directly to the data source. As we are importing Excel data for this example, the only option here is to choose **Import**.

7. Click on **OK** to import the data.
8. After selecting the **Data Connectivity mode** setting, we normally see the **Report**, **Data**, and **Model** tabs on the left-hand side of the Power BI window. If we create a **DirectQuery** connection, the **Data** tab will not be shown as the data is not stored in the model at all:

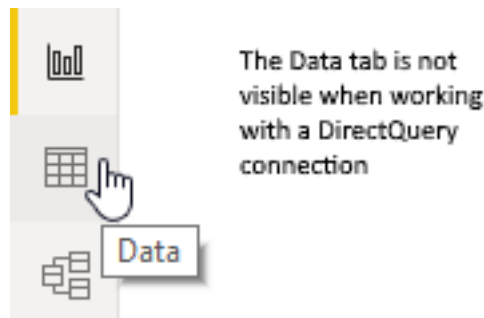


Figure 7.8 – The Data tab visibility in Power BI

Let's talk a little about how Power BI stores imported data and where we can view this in the Windows environment in the next section.

Looking at where Power BI stores data

When a report is opened in Power BI Desktop, the dataset is stored in the memory of the machine that is hosting that Power BI report. After publishing your Power BI report to the Power BI website, it resides in the memory of the computer in the cloud. I am sure you have seen the notifications that pop up when working in Power BI alerting you that the data needs to be refreshed. This is why we also refer to **Import Data** as a scheduled refresh:

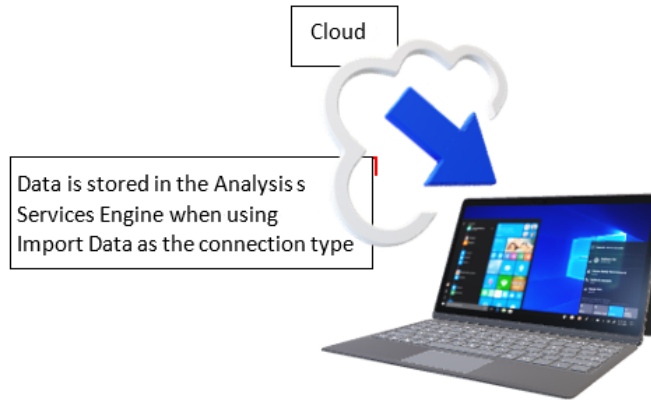


Figure 7.9 – Representation showing how data is stored in Power BI

Let's use **Task Manager** in the next section to identify whether Microsoft SQL Server Analysis Services is running as this is where data is stored in memory when using the **Import Data** connection type.

Investigating whether Microsoft SQL Server Analysis Services is running

To see where Power BI models are stored in memory when using the **Import** data connection **Storage mode**, perform the following steps:

1. Right-click on the Windows taskbar.
2. Choose **Task Manager** from the shortcut menu provided.

3. In the **Task Manager** dialog box that populates, scroll down until you locate the **Power BI Desktop** process. Click on the drop-down arrow to the left of the **Power BI Desktop** heading to expand the options underneath it.
4. Locate the **Microsoft SQL Server Analysis Services** process.
5. Close the dialog box once you have viewed the process:

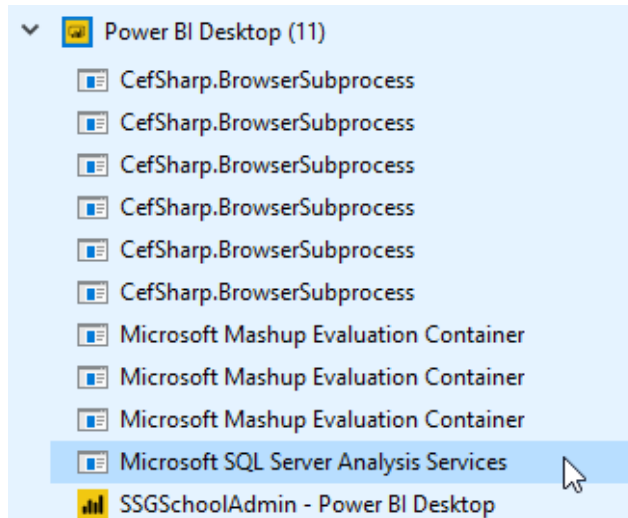


Figure 7.10 – The Power BI dropdown in Task Manager showing Microsoft SQL Server Analysis Services

In the next section, we will look at the various refresh types in Power BI.

Understanding the Power BI refresh types

It is extremely important to make sure that the data in Power BI is the most recent and accurate dataset in your Power BI report visualizations. We want to present relevant and engaging data to an audience through report dashboards. In this topic, we will discuss the refresh types, which will help you understand where Power BI might spend its time during a refresh operation.

Power BI online has a **New look** icon, which, when activated, will change the layout and position of items on the Power BI interface. Depending on the view selected in your online Power BI interface, you may find that the placement of the icons we referenced may be in a slightly different location than explained here. The **New look** icon is located on the Power BI online title bar toward the right-hand side:

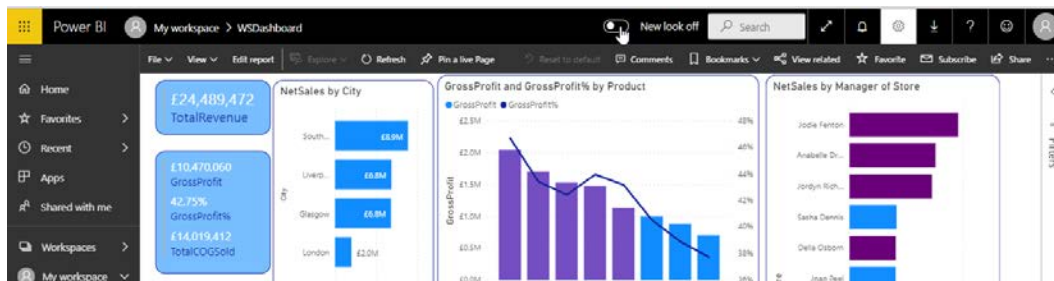


Figure 7.11 – The New look icon in Power BI online

In a Power BI dataset, we have different flows of data from which reports and tiled dashboards will query the dataset. These different modes of storage all have different refresh requirements due to the fact that the data comes from different storage modes and dataset types, as explained in the *Investigating whether Microsoft SQL Server Analysis Services is running* section.

Datasets in the **Import** mode are cached (not imported) and so the size of the dataset is enormous. Refresh issues will arise if the dataset reaches maximum dataset limits. When using the **LiveConnect/DirectQuery** storage mode option, the data is not imported and so is not refreshed, as mentioned before. It does refresh a tile, however, every hour or so by default to allow the tiles to show the most recent results in the report visual.

The dataset settings can be edited to set dashboard tiles and reports to update using the manual **Refresh Now** option. Let's investigate this setting in the next topic.

Learning how to refresh a OneDrive connection

Any Excel or **comma-separated value** (CSV) report or dataset that is compiled using Power BI Desktop and shared online using OneDrive or Sharepoint is refreshed by Power BI using the OneDrive refresh option. For this type of refresh, Power BI normally checks whether a connection to a file in OneDrive needs to be synchronized approximately every 30 minutes.

Note

Power BI does not import data from OneDrive; it **synchronizes** datasets and reports with source file refreshes.

For this example, we will use the report dashboard named `WSDashboard.pbix`, which has been shared to a OneDrive location. We will take you through the steps to set this up before we look at the refresh options:

1. Make sure you have Power BI online open by visiting `https://app.powerbi.com/home`.
2. We will be using the report dashboard created in *Chapter 6, Advanced Power Queries and Functions*, for this example. This report has been imported into Power BI online using the **Get Data** option and linked via the OneDrive online service. The report view looks as follows after it is connected through OneDrive to Power BI online:

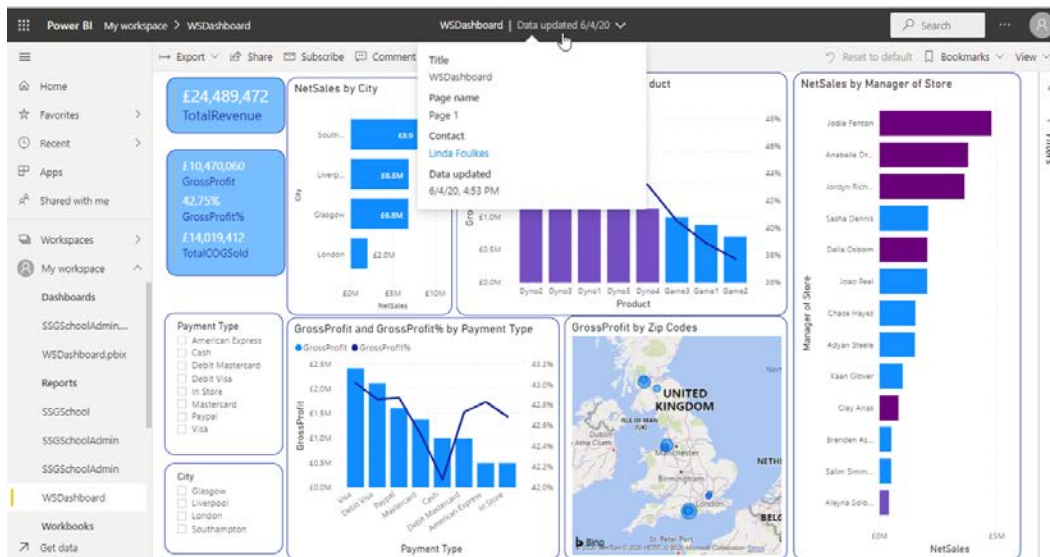


Figure 7.12 – The report dashboard is connected through OneDrive

Now, let's look at the steps to connect your report using OneDrive:

1. Click on the **Get data** item in the left-hand panel in Power BI online.
2. Click on the **Get** icon in the **Files** tile:

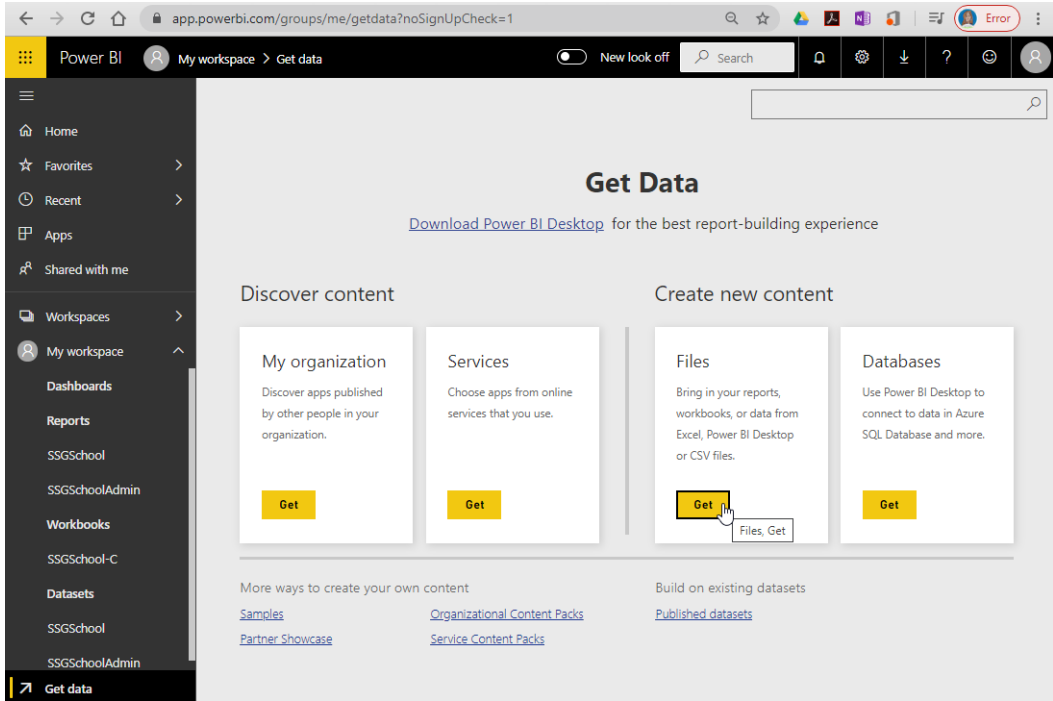


Figure 7.13 – Clicking on Get to create a connection to new content

3. Choose the **OneDrive - Personal** tile:

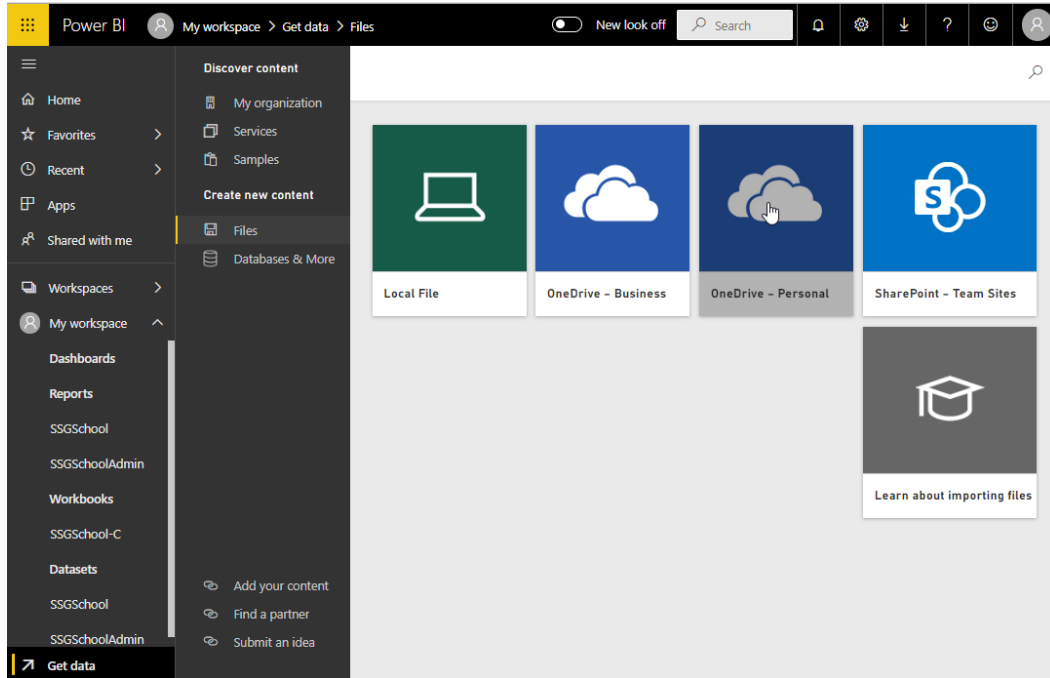


Figure 7.14 – The OneDrive - Personal connection tile

4. You will receive a notification asking for permission for Power BI to access the data you are connecting to. Select **Yes** to continue:

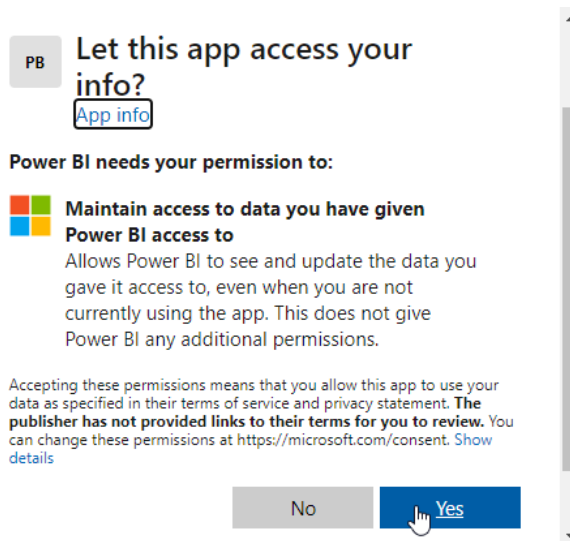


Figure 7.15 – Notification popup asking for permission to access data

5. Select the Power BI dataset you wish to connect to, and then click on the **Connect** icon in the top-right corner of the window:

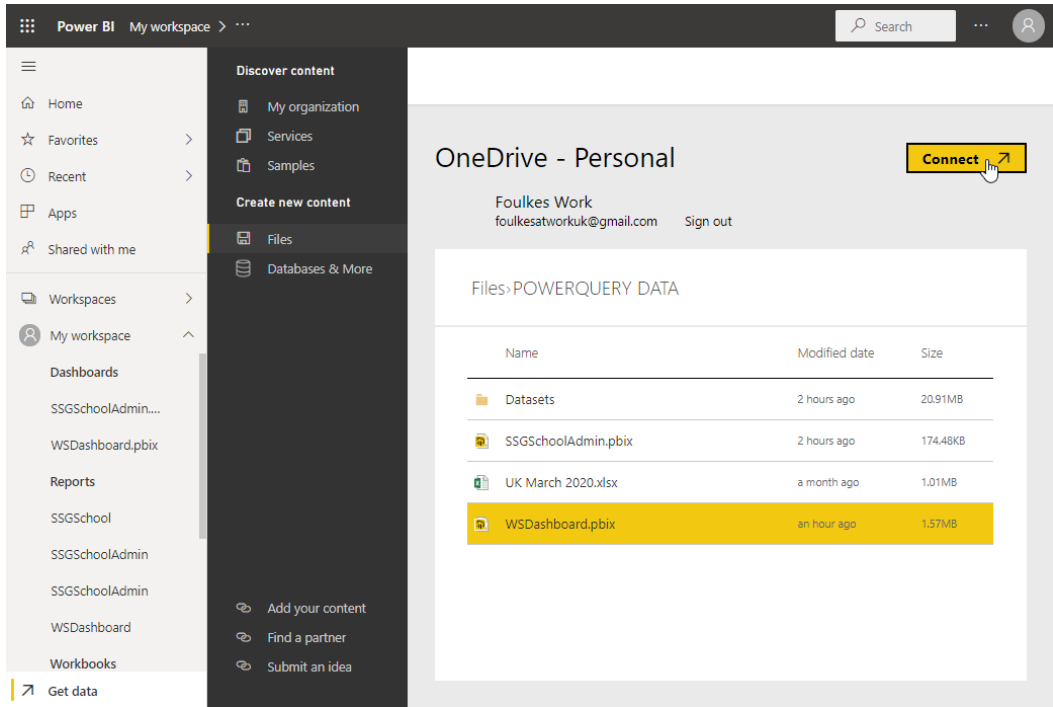


Figure 7.16 – Selecting a dataset from a OneDrive location

The connection is made to the dataset in Power BI online and the report is shown in the interface to the left under the **Workspace** heading.

In this section, we went through the steps to connect a report using OneDrive. In the next section, we will view and perform the refresh.

Viewing and performing a OneDrive refresh

We can see when data was last refreshed by Power BI on the connected OneDrive dataset using a number of methods, of which a few are shown in the following screenshots, along with how to refresh the connection:

- The first is to view the details by visiting the title bar of the Power BI interface:

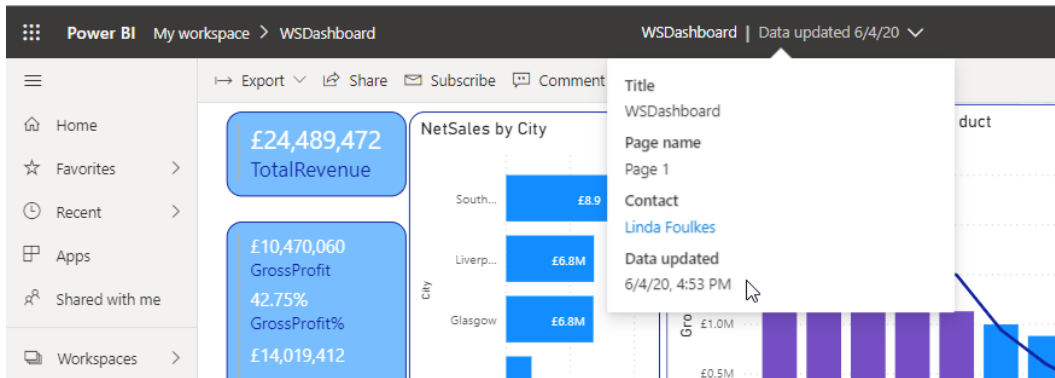


Figure 7.17 – Data last updated shown in the Power BI title bar

- Click on the three dots, located in the workspace menu, from which you can choose to perform a refresh:

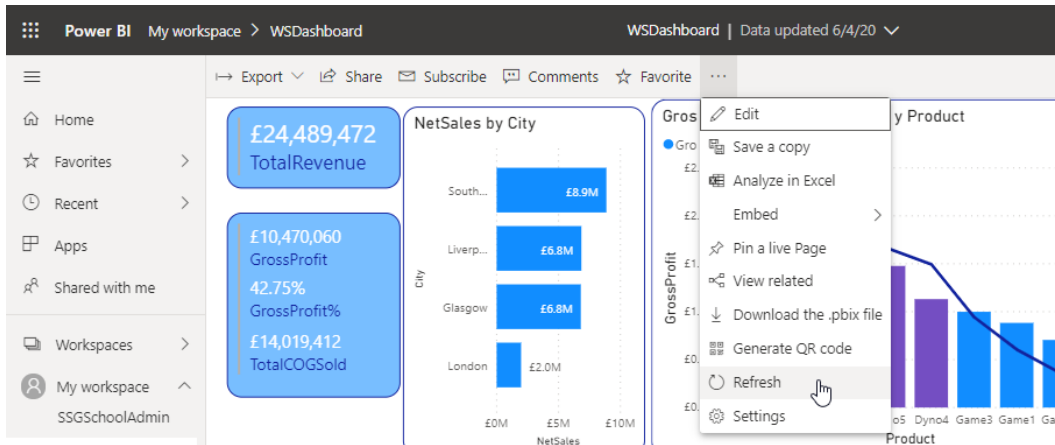


Figure 7.18 – The Refresh option in the workspace menu

Visit the **Workspace** view, where you can select the **Datasets + dataflows** heading to display its contents. Hover the mouse pointer over the relevant dataset to see the **Refresh now** and **Schedule refresh** icons, or click on the three dots to access the refresh settings:

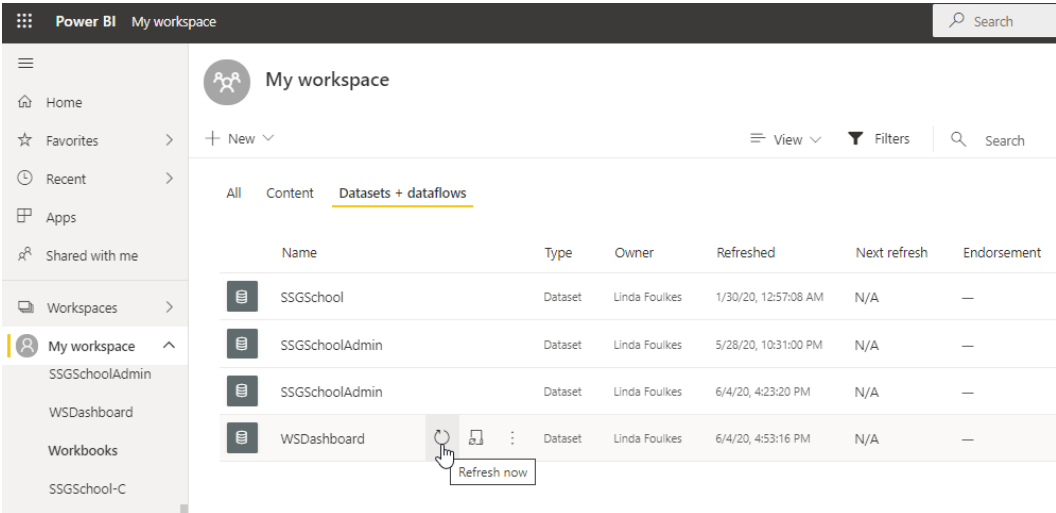


Figure 7.19 – The Datasets + dataflows heading, where you can access the relevant refresh and settings options

The **Refresh now**, **Schedule refresh**, and **Settings** options are also visible by clicking on the three dots at the end of the dataset name in the left-hand side panel in the Power BI interface:

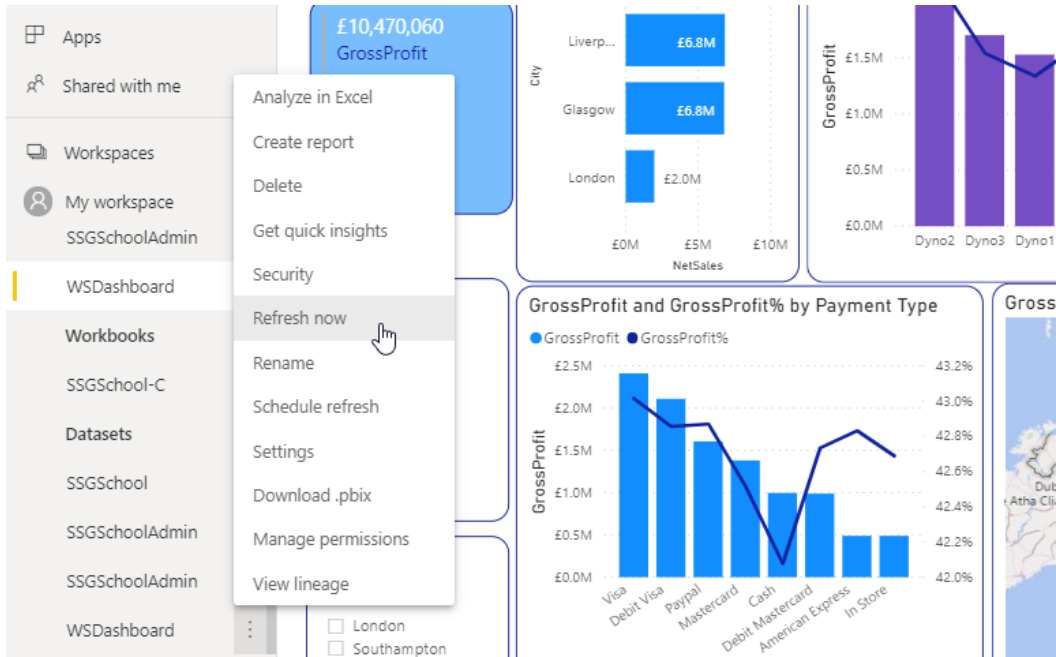


Figure 7.20 – The Refresh now, Schedule refresh, and Settings options in the Power BI interface

Now that you know how to view and perform a OneDrive refresh, we will learn about the **Scheduled refresh** option.

Setting a scheduled refresh

You can control when a refresh happens using the **Scheduled refresh** options in Power BI online. In the following example, we have launched **Settings** for the SSGSchoolAdmin dataset by clicking on the three dots to launch the menu from the navigation pane:

1. Locate the **Scheduled refresh** heading from the **Settings** pane.
2. Switch on the setting by clicking on the **Keep your data up to date** button.

3. Choose a **Refresh frequency** option—either **Daily** or **Weekly**—using the drop-down list provided:

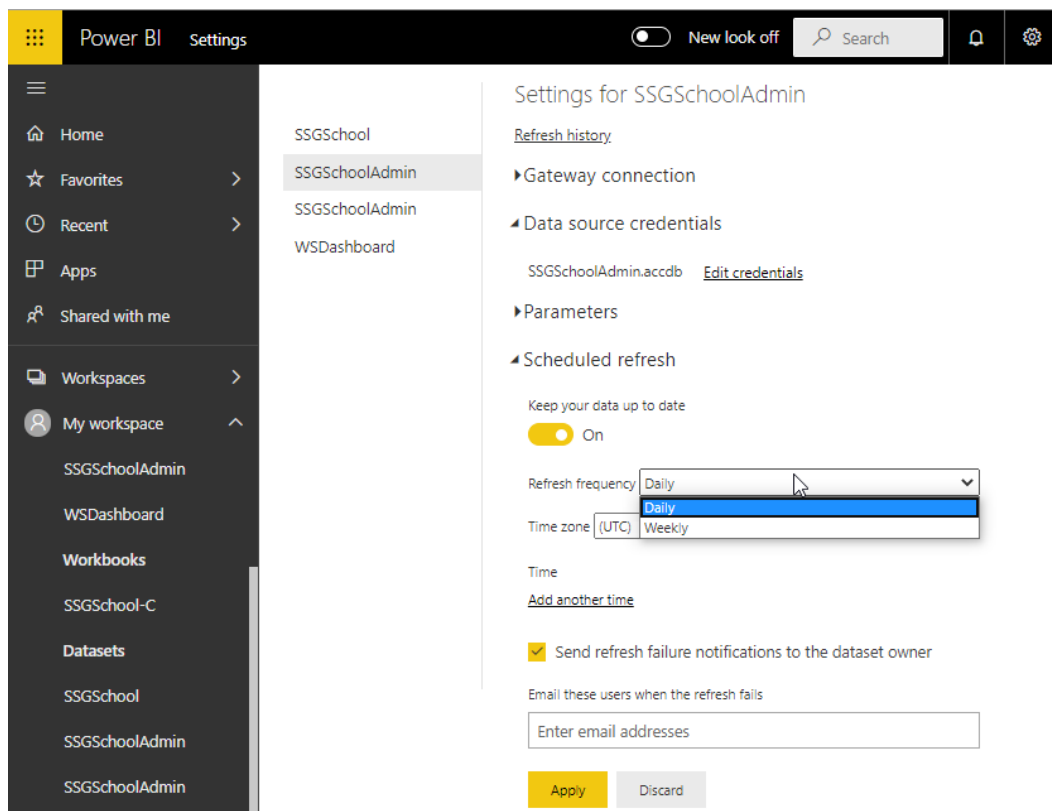


Figure 7.21 – Setting the Scheduled refresh frequency and email notifications of a failed refresh

4. Notice that you can set the option to **Send refresh failure notifications to the dataset owner** by inputting the email address into the area provided so that these individuals are notified if a refresh fails.

In this section, we learned how to set a scheduled refresh using the relevant options, and investigated the various frequency options. In the next few sections, we will introduce even more refresh options as a taster for you to explore further.

Incremental refresh

Incremental refresh is a fairly new feature that needs to be configured in Power BI. It is used to increase the speed of refreshes when using large tables in Premium workspaces in Power BI.

Its main advantage is that it only updates the data that needs to be refreshed. The user will decide on the timeline in terms of the rows to store and refresh, as well as whether to detect data changes within a certain period if the maximum value of the **DateTime** column changes. The feature only works once the report is published to the Power BI service. More information on this refresh type, along with how to set it up, can be found at <https://docs.microsoft.com/en-us/power-bi/admin/service-premium-incremental-refresh>.

Automatic page refresh

Automatic page refresh applies to **DirectQuery** data sources only. Once set up, Power BI Desktop automatically sends queries to the data source (remember, only **DirectQuery** sources), and then, after a short interval, it returns the data, thereby updating any visuals you have in your report dashboard. Note that you can also set page refresh intervals for any reports created in Power BI Desktop but published to the Power BI service. One thing to note here is that the refresh rate of the interval cannot be less than the duration of time it takes for any new data to arrive at the source. So, if data arrives every 5 seconds, then the refresh interval needs to be 5 seconds.

You need to take into consideration the return times of queries to report visuals as sometimes you might configure the setting to load way too much data, and the data source will not be able to cope with this. Use an equation to calculate return times of queries to report visuals according to how many visuals need to be refreshed and the number of users viewing the reports at the same time. This will result in the number of queries you need to support by setting the interval of the automatic page refresh to cope with the load. More information on this is located at <https://docs.microsoft.com/en-us/power-bi/create-reports/desktop-automatic-page-refresh>.

To set automatic page refresh in Power BI Desktop, do the following:

1. Select the report page you need to set a refresh for.
2. Click on the **Formatting** icon (the paint roller icon) in the **Visualizations** pane.
3. Locate **Page refresh** (at the bottom of the list).
4. Click the button at the top right to turn the setting to **On**.
5. Set the **Duration** and **Unit** values for the page refresh interval. The minimum value here is 1 second and the default is 30 minutes.
6. Your report will now refresh at the interval set.

This section introduced the automatic page refresh option, and you have learned how to set this option in Power BI Desktop using the steps provided. We will now look at the dataflow refresh option.

Dataflow refresh

Often, you will see from your **Start** and **End** refresh history report in Power BI that the process of refreshing data is taking a while. You will also notice the **Apply query changes notification** window, which highlights refresh details and chugs along while the refreshing is happening. Dataflow refresh speeds up the refresh step by running a Power Query process separately from any other Power BI reports. So, what we mean is that there is no report or dataset to bind the process and so it is more quickly stored in a Microsoft cloud storage facility named Azure Data Lake Storage. Find out more about this refresh option at <https://azure.microsoft.com/en-gb/services/data-lake-analytics/>.

Summary

In this chapter, we went through the theory relating to the different storage modes and dataset types available in Power BI. Overall, we saw how to automate reports using the modes in Power Query and why it is done.

We also learned how to view the storage mode and choose various settings to manage how Power BI caches table data. We investigated the three types under the **Storage mode** setting—namely, **Import**, **DirectQuery**, and **Dual**. We also looked at how Power BI stores its data and where to locate Microsoft SQL Server Analysis Services to understand how Power BI keeps its models in memory. In the last section, we learned about the various Power BI refresh types, including the **Import**, **LiveConnect/DirectQuery**, and **OneDrive refresh** modes. We also learned how to use the **Scheduled refresh** option to have control over when data is refreshed. Finally, we learned how to connect a report to OneDrive and refresh the connection by synchronizing the dataset, and then we introduced some new features, including dataflow, incremental, and automatic refresh.

In the next chapter, we will concentrate on the creation of a dashboard from connected data, selecting a visualization type, and publishing and customizing the dashboard. We will also cover multi-dimensional reporting.

8

Creating Dashboards with Power Query

Dashboards are business-intelligent, single-canvas pages that allow the user to tell a story through various visualizations created from table data to highlight important data points for an organization.

The great thing about dashboards is that once they are created, they are completely interactive. They can be viewed from anywhere in the world and, depending on the user's access level, it is possible to edit the data as well. Of course, all the data is live, which means that the millions of rows of data on a dashboard are always up to date and current.

You will find this a very practical and hands-on chapter as if you follow along with the steps, you will create your own interactive dashboard, which you will be able to upload onto the web. In this chapter, some of the skills that you will learn include the following:

- Uploading files that have a specific file format, even if there are many different file formats in the folder
- Learning how to transform data to make sure that the files you are trying to upload work correctly
- Creating one-to-many relationships in order to use other tables

- Creating various measures in order to use our calculated figures in the dashboard
- Creating a dashboard from connected data, selecting a visualization type, and publishing and customizing the dashboard

In this chapter, we're going to cover the following main topics:

- Creating a basic power pivot and PivotChart
- Using Power BI to collect and connect data
- Using Power BI to add to a data model
- Selecting data visualization, a dataset, and an appropriate chart
- Publishing a dashboard
- Sharing a dashboard

Technical requirements

You need an internet connection to download the relevant files from GitHub. Also, the code files for this chapter can be found at <https://github.com/PacktPublishing/Learn-Power-Query/>.

There is an additional folder called `folder` that has the files necessary for the import process.

This chapter assumes that you already know how to load different data sources into Power BI and Power Query.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=MJ04RbLXxXk&list=PLeLcwrwLe186O_GJEZs47WaZXwZjwTN83&index=9&t=0s.

Creating a basic power pivot and PivotChart

Using dashboards, we can see the data in a visual representation instead of just the figures. This sometimes makes it easier to spot trends and analyze data.

Many times, you will be given data from another person or you will have something that came from an exported CSV file that you need to create reports on. In this first section, we are going to look at pivot tables and PivotCharts and how we can create them.

To start, we are going to create a basic pivot table and a PivotChart to see how this is done. We are going to use our sales data, which shows what has been sold, to whom, and the sales representative who sold it. Refer to the following screenshot:

data.xlsx - Excel												
File Home Insert Page Layout Formulas Data Review View Help Power Pivot Tell me what you want to do												
<div> <div>Cut</div> <div>Copy</div> <div>Format Painter</div> </div>	Calibri 11 A A			<div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> </div> </div>			<div> <div>Wrap Text</div> <div>General</div> <div> <div></div> <div></div> <div></div> </div> </div>			<div> <div> <div>Normal</div> <div>Bad</div> <div>Good</div> <div>Neutral</div> </div> <div> <div>Conditional Formatting</div> <div>Format as Table</div> <div> <div>Check Cell</div> <div>Explanatory...</div> <div>Input</div> <div>Linked Cell</div> </div> </div> </div>		
	Clipboard BG Font G			Font G			Alignment G			Number G		
A1 <div>Order No</div>												
A	B	C	D	E	F	G	H	I	J	K	L	M
Order No	Order Date	Order Type	Delivery Date	Salesperson	Region	Product	Company	Country	Address	City	State	Ship
11	13684	43467 In Store	43473 Clive	Midlands	Dyno3	Gormley Lore Murphy	Australia	405 W Lee St	VIC	3321	Self	
12	13688	43468 Online	null	Clive	Midlands	Dyno4	Huetter, Glen A Jr	Canada	7787 W Central Ave #2	NS	82N 6/8	TWO
13	13689	43468 Pickup	43471 Lucie	Midlands	Dyno2	Blaney Sheet Metal	Australia	21 W 2nd St	NSW	2429	Four	
14	13694	43468 Pickup	43473 Lucie	West	Dyno3	Preston Trucking Co Inc	Canada	686 S Dupont Hwy	ON	L11 8H1	TWO	
15	13695	43468 Online	null	Jordan	Midlands	Dyno3	Jean Barbara Ltd	Australia	7686 Carey Ave	VIC	3309	Four
16	13699	43469 Pickup	43472 Lucie	Midlands	Dyno2	Alex Fries & Bros Inc	Canada	326 Witherspoon St	ON	M9R 2W5	Four	
17	13700	43469 Pickup	43473 Lucie	North	Dyno3	Woodstown Natl Bnk & Trst Co	Canada	89 Parade St	ON	N4S 3K1	EZ 5e	
18	13702	43469 Pickup	43472 Lucie	North	Dyno3	Conte, Christopher A Esq	US	70 W Main St	Cuyahoga	OH	ZZ 5h	
19	13705	43469 Online	null	Lucie	North	Dyno3	Bork, Terry D Esq	UK	9923 Dinorben St #4838	Greater London	NW11 8DY	True
20	13709	43469 Online	null	Clive	North	Dyno1	American Council On Sci & Hith	US	636 Commerce Dr #42	Scott	MIN	RT 5h
21	13712	43472 Pickup	43475 Lucie	West	Dyno3	Alvris, John W Esq	Australia	2799 Cajon Blvd	QLD	4615	AZZ S	
22	13715	43472 Pickup	43476 Lucie	West	Dyno1	Scott Marlow Agency	UK	8 Village St	Surrey	GU18 5YQ	OT N	
23	13720	43472 Pickup	43476 Lucie	North	Dyno2	Gabriel & Associates	Canada	22 Dearborn St	SK	S4T 4B1	ZZ 5h	
24	13725	43472 Pickup	43479 Lucie	West	Dyno3	Bork, Terry D Esq	UK	9923 Dinorben St #4838	Greater London	NW11 8DY	CMA	
25	13727	43472 In Store	43476 Clive	Midlands	Dyno1	C W D C Metal Fabricators	US	2 A Kelley Dr	Westchester	NY	Self	
26	13731	43472 Pickup	43475 Lucie	North	Dyno3	Bitar Tool & Die Inc	UK	8 Tagus St #9834	Cumbria	CA25 5EP	APA	
27	13732	43472 Online	null	Jordan	North	Dyno1	Hambro Forest Products Inc	Canada	7979 33 1st	NS	86L 6N1	NOG
28	13737	43472 Pickup	43478 Lucie	North	Dyno1	Callender, William C Esq	UK	18 Nimrod St	Cumbria	LA34 3SU	OT N	
29	13738	43472 Pickup	43478 Lucie	Midlands	Dyno3	Fineshriber, Marilyn Esq	UK	87 Eldon Place	Lancashire	B012 7RY	AZZ S	
30	13741	43472 In Store	43475 Clive	West	Dyno2	Dickman, J Scott Esq	Canada	4153 Broughton Ave	BC	V9A 6P6	Self	
31	13742	43472 Pickup	43476 Lucie	West	Dyno1	Lehigh Furn Divn Lehigh	Australia	2 Pompton Ave	NSW	2082	NOG	
32	13744	43472 In Store	43479 Lucie	Midlands	Dyno1	Deltam Systems Inc	US	3270 Dequindre Rd	Southport	NY	Self	
33	13745	43472 In Store	43477 Clive	North	Dyno2	Gormley Lore Murphy	Australia	405 W Lee St	VIC	3321	Self	
34	13748	43473 Online	null	Jordan	North	Dyno1	Kitchen People	Canada	3 E 31st St #77	NB	E3G 0A3	APM
35	13750	43473 Online	null	Lucie	Midlands	Dyno3	Sinclair Machine Products Inc	Australia	75 Elm Rd #1190	ACT	2600	EZ 5h
36	13753	43473 Pickup	43476 Lucie	Midlands	Dyno2	Marc Hotels & Resorts Inc	Canada	611 Grand Ave	AB	T2J 1Y3	ZZ 5h	
37	13755	43473 In Store	43482 Jordan	West	Dyno2	Teel, Louis N Esq	UK	807 Derham St	North Lincolnshire	DN17 4EN	Self	
38	13758	43473 In Store	43480 Clive	Midlands	Dyno3	Valley Hs Bnk	Australia	9892 Hiernand W	NSW	2474	Self	
39	13763	43474 Pickup	43481 Lucie	Midlands	Dyno5	Central Die Casting Mfg Co Inc	US	1610 14th St NW	Newport News City	VA	CMA	
40	13768	43474 Online	null	Clive	North	Dyno3	Bork, Terry D Esq	UK	9923 Dinorben St #4838	Greater London	NW11 8DY	OT N
41	13773	43474 Pickup	43477 Lucie	North	Dyno3	Centro Inc	US	17 US Highway 111	Williamson	TX	AZZ S	
42	13776	43474 Pickup	43478 Lucie	West	Dyno2	Dobscha, Stephen F Esq	US	40 Cambridge Ave	Dane	WI	True	

Figure 8.1 – A basic table

We are going to create a pivot table and chart of the salespeople and how much they have sold.

Note

If you are already familiar with how to create these items, then skip to the next section.

Let's see how this is done:

1. Convert the range to a table (*Ctrl* + *T*) and make sure that the entire table has been selected.
2. Create a pivot table by selecting the **PivotTable** option from the **Insert** menu, which will bring up the **Create PivotTable** window.
3. Select **New Worksheet** and then click **OK**.

Important note

Before creating the pivot tables, make sure that there are no blank columns, blank rows, or merged cells, as this will affect your end product.

After performing the preceding steps, refer to the following screenshot:

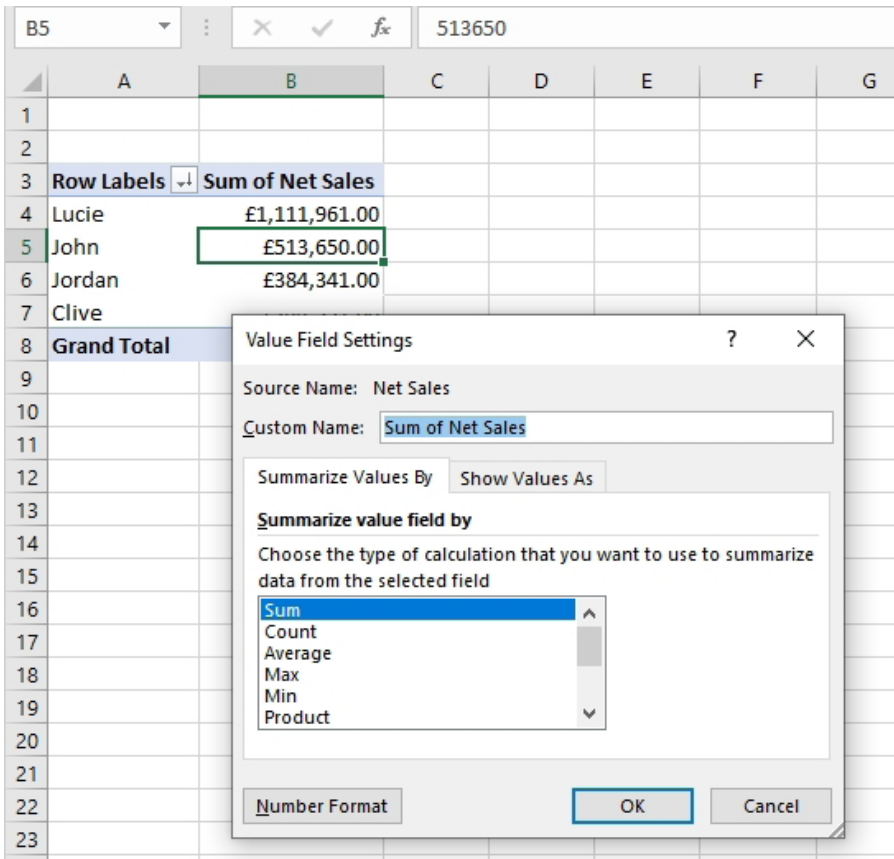


Figure 8.2 – The Value Field Settings dialog box

There are a few things missing that need to be added:

1. Move **Salesperson** to **ROWS** and **Sum of Net Sales** to **VALUES**.
2. The currency numbers are not formatted correctly in the pivot table. Right-click anywhere on one of the numbers and select **Value Field Settings**.
3. Click on the **Number Format** button and change the format to **Currency** before clicking **OK**.
4. Rank **Salesperson** from highest to lowest by right-clicking on any of the currency figures and then selecting **Sort Largest to Smallest** from the **Sort** menu.

Now that we have our data in the correct format, we can create a PivotChart that will depict the sales representatives and their sales in descending order.

Perform the following steps:

1. Select **PivotChart** from the **Analyze** menu and select the bar graph:

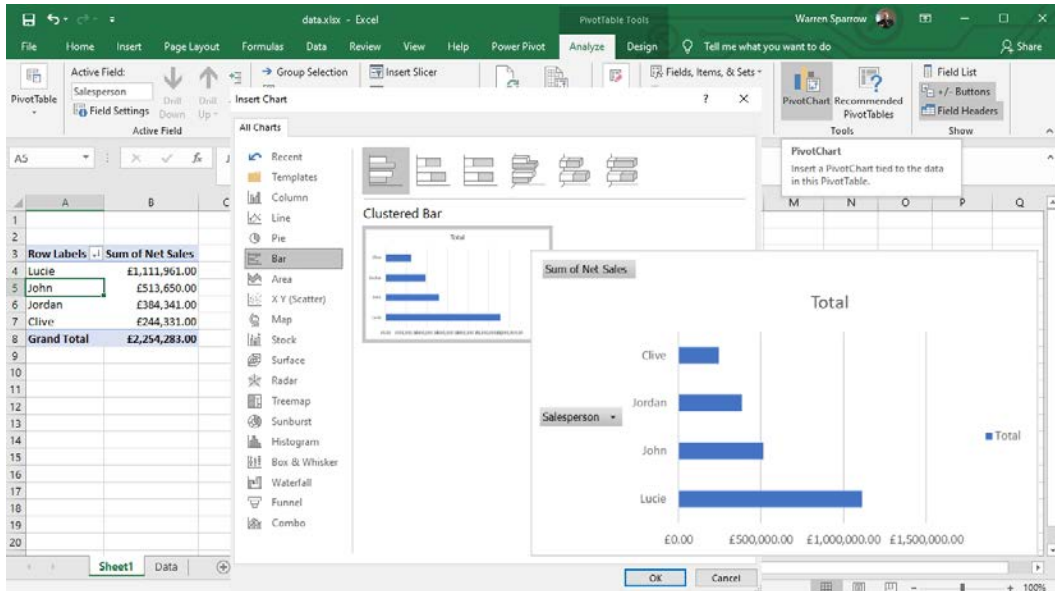


Figure 8.3 – Adding a PivotChart

2. To neaten the chart, right-click on the **Salesperson** button and select **Hide All Field Buttons on Chart**.
3. The name order is not the same as in our PivotChart, and we want this to match. Right-click on any salesperson's name on the PivotChart and then select **Categories in reverse order** from **Format Axis**.
4. Delete the labels, the legend, and the vertical lines. Rename the title by right-clicking on **Total** and selecting **Edit Text**.
5. We want to make the bars in the bar chart a little bit thicker so that they stand out more. Right-click on one of the bars and then select **Format Data Series**. Change the gap width to a smaller number, which will make the bars wider. Depending on the number of salespeople, the bar's thickness needs to be adjusted.

This is what the final outcome will be:

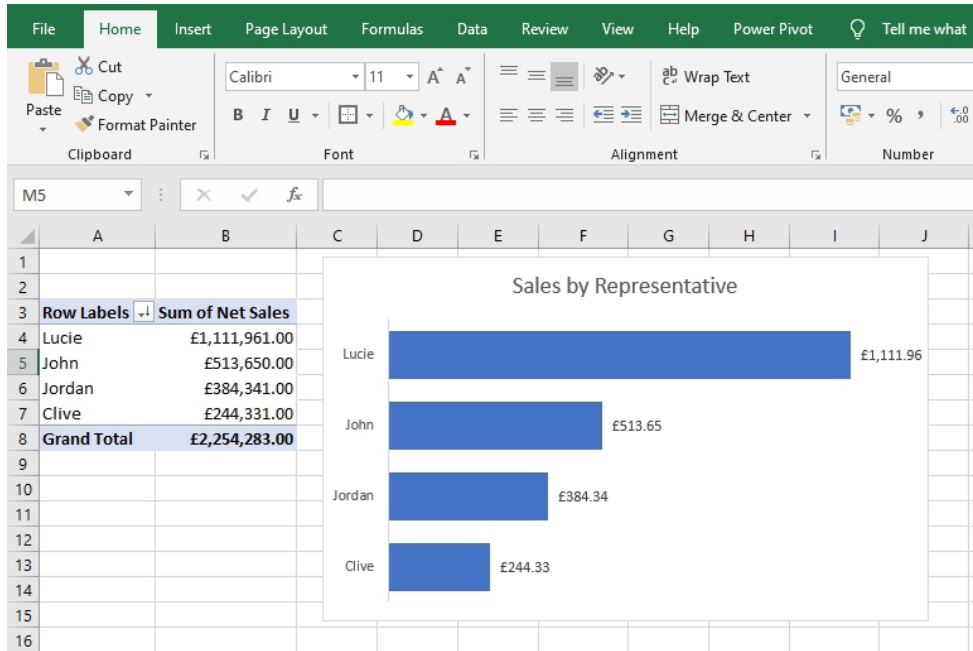


Figure 8.4 – Value Field Settings with a chart

Although this is a basic pivot table and PivotChart, it can be done very quickly and the overall result is brilliant.

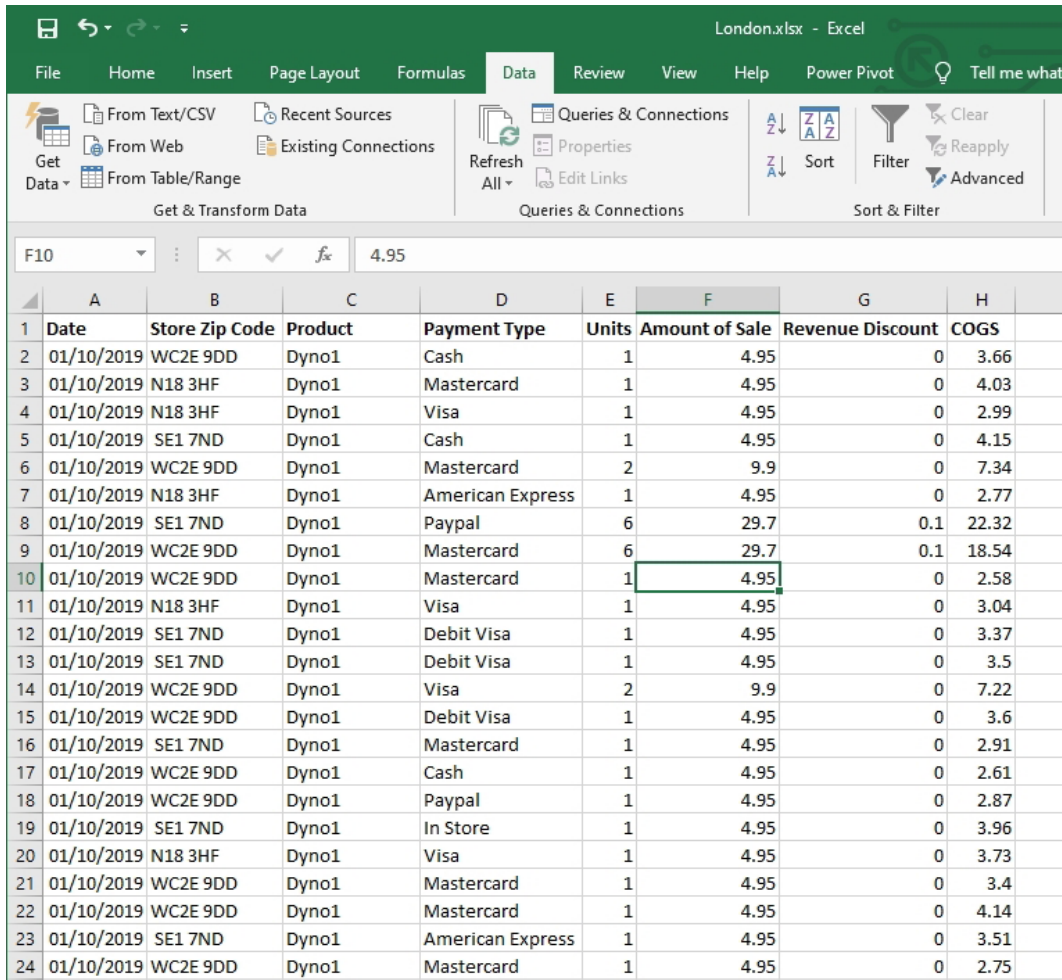
In the next section, we will look at how we can use Power BI Desktop to create an interactive dashboard.

Using Power BI to collect and connect data

There are a number of steps that we need to follow to create a dashboard. The first step is to collect and connect the data we have to the data model so that we can then move on to the next step.

We will use our sales data from different stores based in the UK. The reality is that these files could be online, on a SQL server, or on a website, but you could also have downloaded the files locally, as with this example. The only difference in the steps would be when you get the data. This example shows connecting to a folder, but if your data is somewhere else, you can refer to *Chapter 4, Connecting to Various Data Sources Using Get & Transform*, to see how to get data from various sources into Power BI Desktop. The files for this example can be found in the GitHub repository.

In the following screenshot, we have the ZIP code of the store, the product that was sold, and how the person paid for the goods, and the last column (column H) is the cost of the goods that were sold:



	A	B	C	D	E	F	G	H
	Date	Store Zip Code	Product	Payment Type	Units	Amount of Sale	Revenue Discount	COGS
1	01/10/2019	WC2E 9DD	Dyno1	Cash	1	4.95	0	3.66
2	01/10/2019	N18 3HF	Dyno1	Mastercard	1	4.95	0	4.03
3	01/10/2019	N18 3HF	Dyno1	Visa	1	4.95	0	2.99
4	01/10/2019	SE1 7ND	Dyno1	Cash	1	4.95	0	4.15
5	01/10/2019	WC2E 9DD	Dyno1	Mastercard	2	9.9	0	7.34
6	01/10/2019	N18 3HF	Dyno1	American Express	1	4.95	0	2.77
7	01/10/2019	SE1 7ND	Dyno1	Paypal	6	29.7	0.1	22.32
8	01/10/2019	WC2E 9DD	Dyno1	Mastercard	6	29.7	0.1	18.54
9	01/10/2019	WC2E 9DD	Dyno1	Mastercard	1	4.95	0	2.58
10	01/10/2019	N18 3HF	Dyno1	Visa	1	4.95	0	3.04
11	01/10/2019	SE1 7ND	Dyno1	Debit Visa	1	4.95	0	3.37
12	01/10/2019	SE1 7ND	Dyno1	Debit Visa	1	4.95	0	3.5
13	01/10/2019	WC2E 9DD	Dyno1	Visa	2	9.9	0	7.22
14	01/10/2019	WC2E 9DD	Dyno1	Debit Visa	1	4.95	0	3.6
15	01/10/2019	SE1 7ND	Dyno1	Mastercard	1	4.95	0	2.91
16	01/10/2019	WC2E 9DD	Dyno1	Cash	1	4.95	0	2.61
17	01/10/2019	WC2E 9DD	Dyno1	Paypal	1	4.95	0	2.87
18	01/10/2019	SE1 7ND	Dyno1	In Store	1	4.95	0	3.96
19	01/10/2019	N18 3HF	Dyno1	Visa	1	4.95	0	3.73
20	01/10/2019	WC2E 9DD	Dyno1	Mastercard	1	4.95	0	3.4
21	01/10/2019	WC2E 9DD	Dyno1	Mastercard	1	4.95	0	4.14
22	01/10/2019	SE1 7ND	Dyno1	American Express	1	4.95	0	3.51
23	01/10/2019	WC2E 9DD	Dyno1	Mastercard	1	4.95	0	2.75

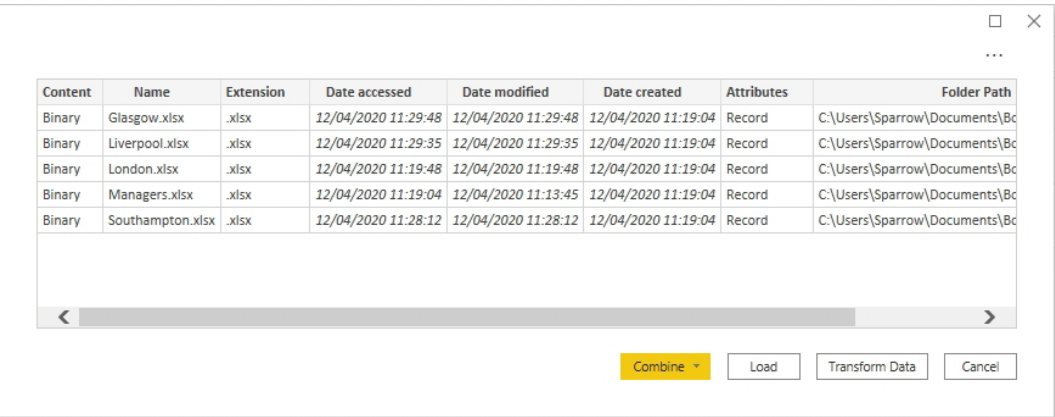
Figure 8.5 – Sales data

To get the data from the Power BI, perform the following steps:

1. To get the data into Power BI, open Power BI Desktop and go to **Data | Get Data | More....** As mentioned earlier, if you have copied the data to a different location, please use the steps found in *Chapter 4, Connecting to Various Data Sources Using Get & Transform*, to see how to load the data.
2. Then, go to **Get Data | Folder | Connect**.

3. From the **Folder** window, click on **Browse** and locate the folder with the relevant data files.
4. Once the folder has been selected, click on **OK**.

Once done, this is how the files will be displayed:



The screenshot shows a file selection dialog box with a table of files. The table has columns: Content, Name, Extension, Date accessed, Date modified, Date created, Attributes, and Folder Path. Below the table are buttons: Combine, Load, Transform Data, and Cancel.

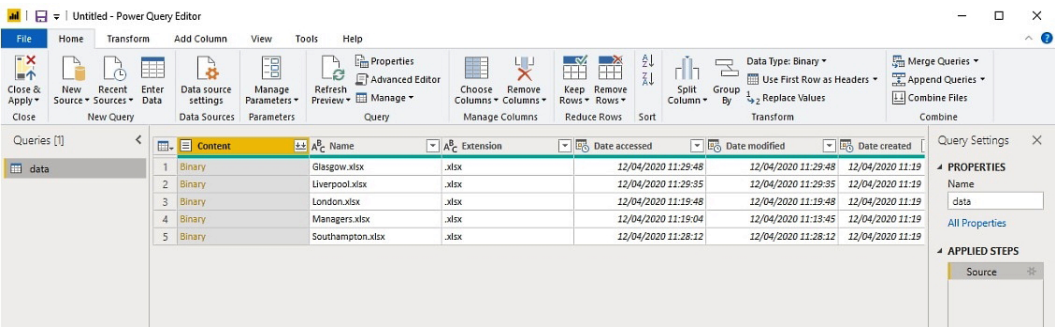
Content	Name	Extension	Date accessed	Date modified	Date created	Attributes	Folder Path
Binary	Glasgow.xlsx	.xlsx	12/04/2020 11:29:48	12/04/2020 11:29:48	12/04/2020 11:19:04	Record	C:\Users\Sparrow\Documents\Bc
Binary	Liverpool.xlsx	.xlsx	12/04/2020 11:29:35	12/04/2020 11:29:35	12/04/2020 11:19:04	Record	C:\Users\Sparrow\Documents\Bc
Binary	London.xlsx	.xlsx	12/04/2020 11:19:48	12/04/2020 11:19:48	12/04/2020 11:19:04	Record	C:\Users\Sparrow\Documents\Bc
Binary	Managers.xlsx	.xlsx	12/04/2020 11:19:04	12/04/2020 11:13:45	12/04/2020 11:19:04	Record	C:\Users\Sparrow\Documents\Bc
Binary	Southampton.xlsx	.xlsx	12/04/2020 11:28:12	12/04/2020 11:28:12	12/04/2020 11:19:04	Record	C:\Users\Sparrow\Documents\Bc

Figure 8.6 – The files are displayed

At this point, I would like to point out that if your data is on a SQL server or in another location, please link it to your data. If you are unsure of how to do this, please refer to *Chapter 4, Connecting to Various Data Sources Using Get & Transform*.

Now, select **Transform Data**, as shown in the preceding screenshot, as we will need to make sure that all the data is formatted correctly before we can use it.

This is how it appears:



The screenshot shows the Power Query Editor interface. The ribbon includes tabs: File, Home, Transform, Add Column, View, Tools, and Help. The 'Transform' tab is active, showing options like Choose, Remove, Keep, Remove, Split, Group, Data Type, Use First Row as Headers, Replace Values, Merge Queries, Append Queries, and Combine Files. The main area displays a table with columns: Content, Name, Extension, Date accessed, Date modified, Date created, and Folder Path. The 'data' query is selected in the left pane. The right pane shows the 'Query Settings' for the 'data' query, including the 'Source' step.

Content	Name	Extension	Date accessed	Date modified	Date created
1 Binary	Glasgow.xlsx	.xlsx	12/04/2020 11:29:48	12/04/2020 11:29:48	12/04/2020 11:19:04
2 Binary	Liverpool.xlsx	.xlsx	12/04/2020 11:29:35	12/04/2020 11:29:35	12/04/2020 11:19:04
3 Binary	London.xlsx	.xlsx	12/04/2020 11:19:48	12/04/2020 11:19:48	12/04/2020 11:19:04
4 Binary	Managers.xlsx	.xlsx	12/04/2020 11:19:04	12/04/2020 11:13:45	12/04/2020 11:19:04
5 Binary	Southampton.xlsx	.xlsx	12/04/2020 11:28:12	12/04/2020 11:28:12	12/04/2020 11:19:04

Figure 8.7 – The query editor

This looks and feels very similar to Power Query in Excel.

We will change the name of the dataset so that it is easier for us to identify it later. Change the **Name** field in the **PROPERTIES** field to `SalesData`. This will now become the name of the query, and it will also become the name of the dataset that we will need to import later.

If we look at the files that we have imported, they are all currently Excel workbook files, but there could potentially be a problem with this later when we are trying to add additional files. It is best practice for us to try to pre-empt any future problems by changing this to make sure that all the extensions are in lowercase. The reason for this is that the formula is case sensitive, as is the case in Excel Power Query. We can sort this out by right-clicking on the **Extension** column and selecting **lowercase** from **Transform** to make sure that the extensions are always lowercase, even if the user manually types it in uppercase.

Refer to the following screenshot:

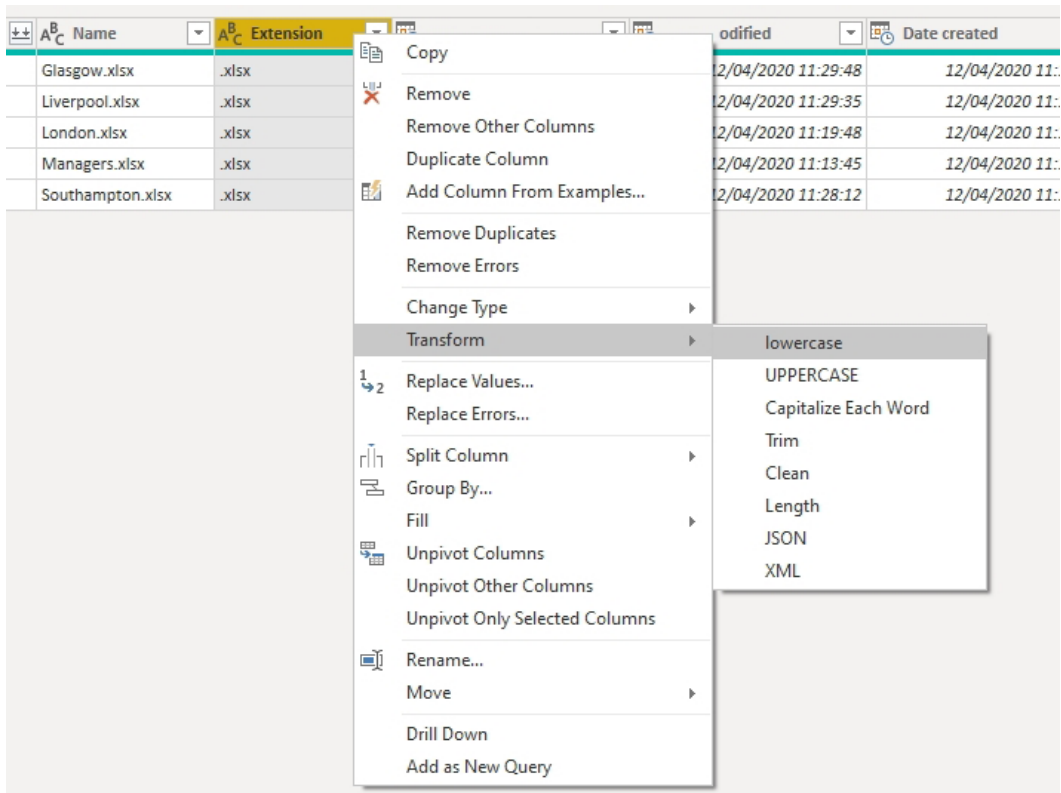


Figure 8.8 – Transforming to lowercase

The second thing we can do is to make sure that only Excel files are imported. Click on the **Extension** column filter and then select **Equals ...** for **Text Filters**, as follows:

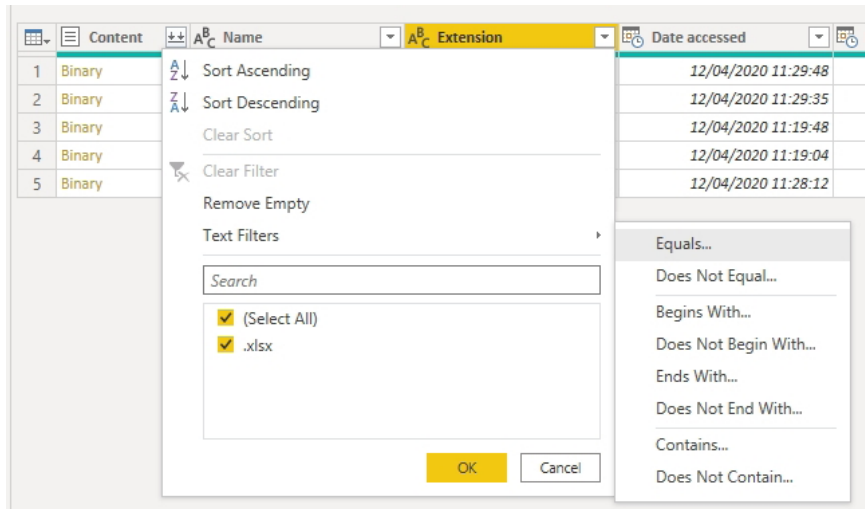


Figure 8.9 – The Equals text filter

We do not want other files to be imported, so we can make a rule to only import Excel workbooks. Refer to the following screenshot:

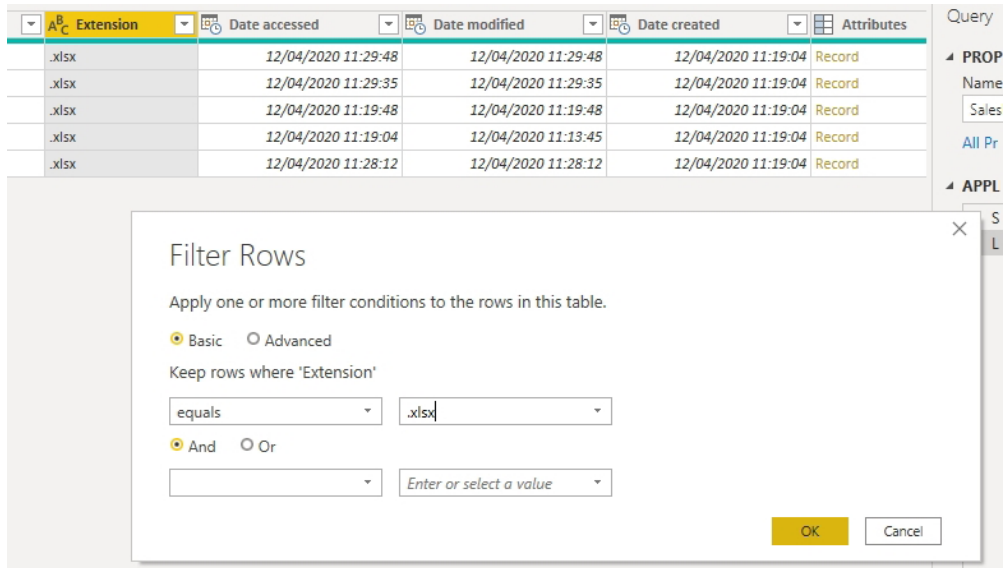


Figure 8.10 – The Filter Rows window

We need to type `.xlsx` into the textbox. This also means that previous versions of Excel will not be imported.

At this point, we do not need any of the other columns except for the **Content** column. So, we will remove the rest by right-clicking on the **Content** column and then clicking **Remove Other Columns**. This is done as follows:

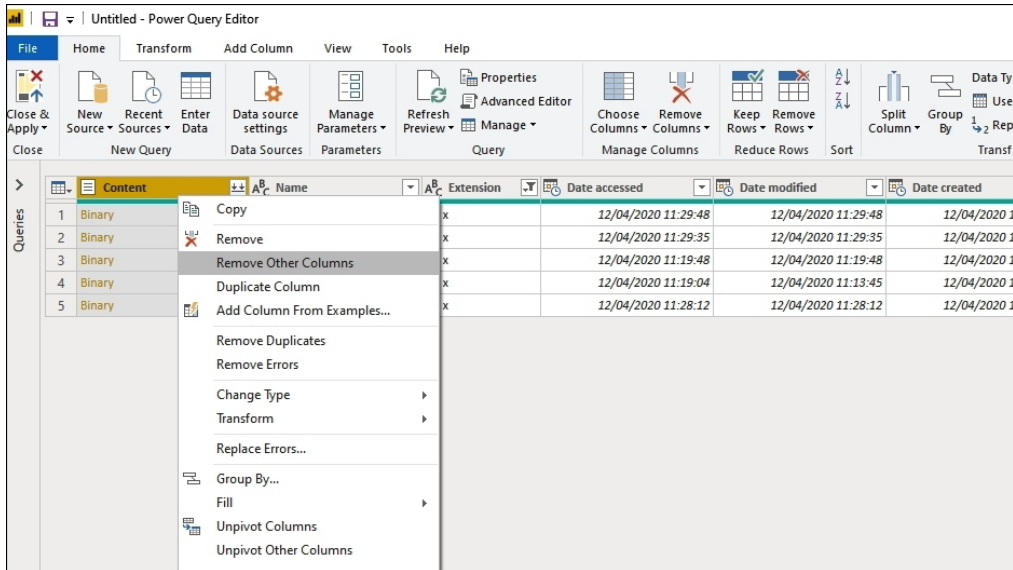


Figure 8.11 – Removing other columns

If we click on the right-hand side of each of the cells where it says **Binary**, next to each number is the Excel file that is associated with this content, as follows:

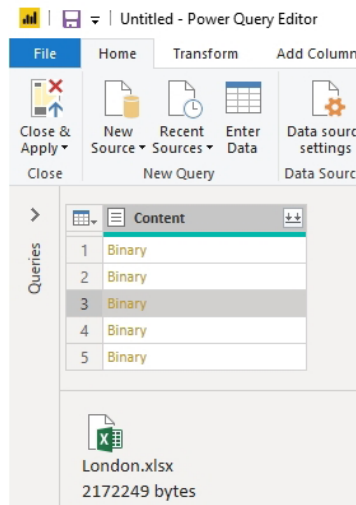


Figure 8.12 – The binary Excel files

After this, we will see how to combine files.

Combining files

Normally, for us to combine numerous Excel files, we can select the **Combine Files** icon, which will extract all the data and combine the files (this is the two down arrows icon on the right-hand side of the Content column). This, unfortunately, will not work for us for this specific example as it does not promote headers as its default behavior.

We will need to create an extra column and use the Excel workbook function to extract the data we need from each of the different Excel files. Refer to the following screenshot:

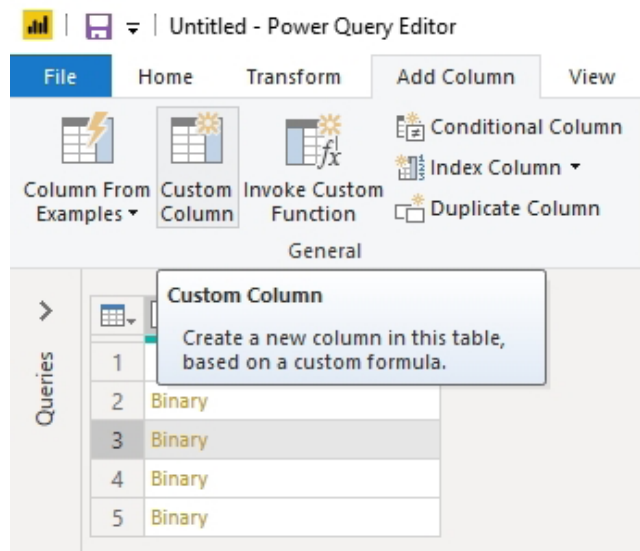


Figure 8.13 – Adding the custom column

Let's see how to do that:

1. From the **Add Column** menu, click on **Custom Column**, which will bring up the **Custom Column** window.
2. Create a new column name, `GetExcelData`, and for **Custom column formula**, type in `Excel.Workbook([Content], true)`, and then click **OK**.

Note

For the [Content] part of the preceding option, you will need to double-click on the **Content** column in the **Available columns** window on the right.

This is depicted in the following screenshot:

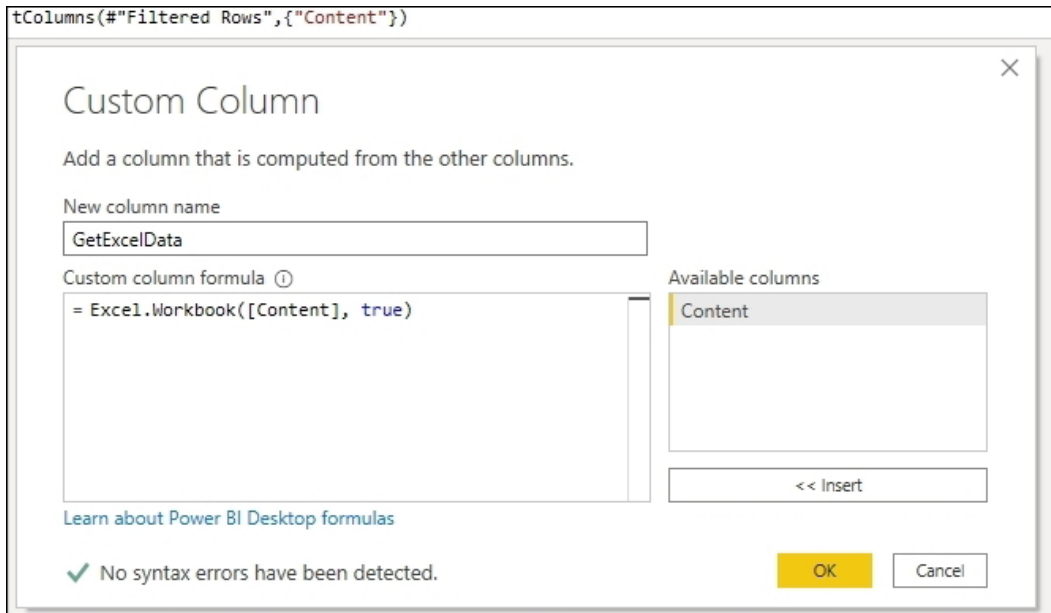


Figure 8.14 – New column formula

This first part of the formula before the comma tells Power BI to open the contents of each of the Excel files. The second part of the formula after the comma (`true`) tells Power BI to promote headers.

The next step will be to keep all the objects that we need and delete the unnecessary content we will no longer need.

When you click on the newly created **GetExcelData** column, you will notice that it has extracted all of the different objects within that file:

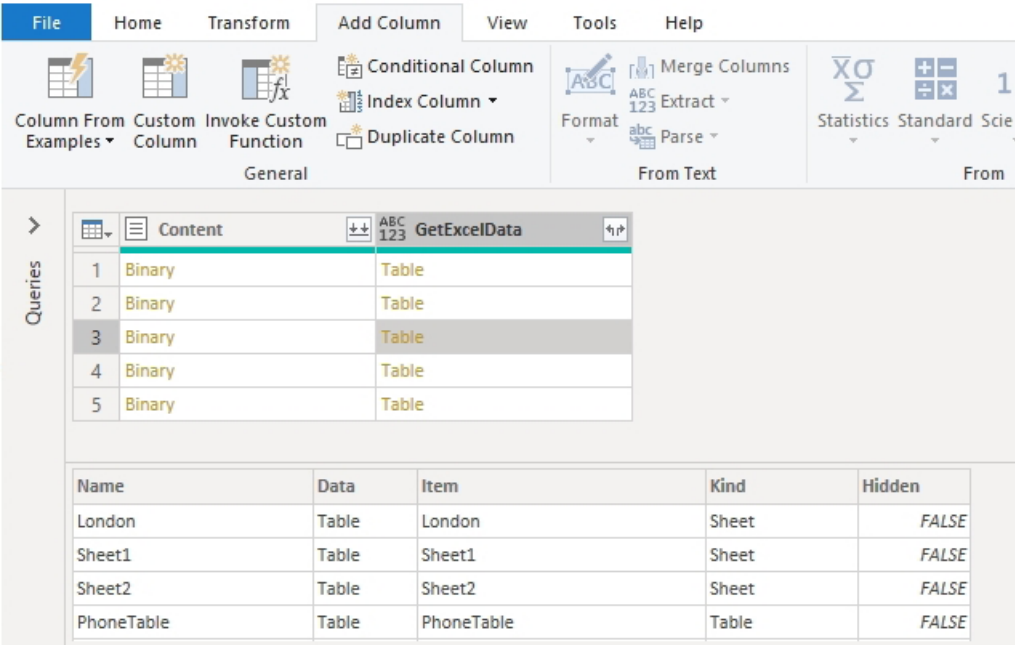


Figure 8.15 – The GetExcelData column

We no longer need the **Content** column as everything is now in the **GetExcelData** column, so we can remove the **Content** column. Now, we have all the Excel documents with promoted headers, so we can click on the **Combine Files** icon on the right-hand side of **GetExcelData**.

The only thing we need to do is untick **Use original column name as prefix** and then click **OK**.

When we look at the window now, there are a number of different objects that have been imported into this query. Some of these objects are not necessary for us to use:

File Home Transform Add Column View Tools Help						
Column From Examples		Custom Column	Invoke Custom Function	Conditional Column	Merge Columns	
				Index Column	Format	
				Duplicate Column	Statistics	
					Standard	
					Scientific	
					Rounding	
					Information	
					Date	
					Time	
					Duration	
					From Date & Time	

Queries	ABC 123 Name		ABC 123 Data		ABC 123 Item		ABC 123 Kind		ABC 123 Hidden	
	1	Glasgow	Table		Glasgow		Sheet			FALSE
	2	_xlnm_FilterDatabase	Table		Glasgow!_xlnm_FilterDatabase		DefinedName			TRUE
	3	Liverpool	Table		Liverpool		Sheet			FALSE
	4	_xlnm_FilterDatabase	Table		Liverpool!_xlnm_FilterDatabase		DefinedName			TRUE
	5	London	Table		London		Sheet			FALSE
	6	Sheet1	Table		Sheet1		Sheet			FALSE
	7	Sheet2	Table		Sheet2		Sheet			FALSE
	8	PhoneTable	Table		PhoneTable		Table			FALSE
	9	_xlnm_FilterDatabase	Table		London!_xlnm_FilterDatabase		DefinedName			TRUE
	10	_xlnm_FilterDatabase1	Table		Sheet1!_xlnm_FilterDatabase		DefinedName			TRUE
	11	DiameterTable	Table		DiameterTable		DefinedName			FALSE
	12	DiametreTable	Table		DiametreTable		DefinedName			FALSE
	13	_xlnm.Print_Area	Table		Sheet2!_xlnm.Print_Area		DefinedName			FALSE
	14	ManagerTable	Table		ManagerTable		Sheet			FALSE
	15	dZipManagers	Table		dZipManagers		Table			FALSE
	16	Southampton	Table		Southampton		Sheet			FALSE
	17	_xlnm_FilterDatabase	Table		Southampton!_xlnm_FilterDatab...		DefinedName			TRUE

Date	Store Zip Code	Product	Payment Type	Units	Amount of Sale	Revenue Discount	COGS
11/12/2019	G3 8AG	Dino1	Visa	1	24.95	0	13.54
20/12/2019	G3 8AG	Dino2	Paypal	1	28.75	0	14.82
21/10/2019	ML12 6HD	Dino2	Visa	1	28.75	0	15.54
13/11/2019	G3 8AG	Dino3	Cash	1	43.96	0	22.93

Figure 8.16 – Different kinds of objects

From the preceding screenshot, we can see several different columns. Here's what each of them indicates:

- The Name column tells us the name of the object.
- The Data column has our Excel data, and if you click on it, it will display a preview of the file at the bottom.
- The Kind column is the one that we need to concentrate on, as the only type that we need to be displayed is the Sheet type.
- Click on the drop-down list on the right-hand side of the Kind column and select **Equals...** from the **Text Filters** menu. Type Sheet in title case, as this is the same as what it says in the Kind column, then click on **OK**.

When we look at the Name column, we can see that there are Excel worksheets that have the default names, such as Sheet1, which does not have any relevant information:

	ABC 123 Name	ABC 123 Data	ABC 123 Item	ABC 123 Kind	ABC 123 Hidden
1	Glasgow	Table	Glasgow	Sheet	FALSE
2	Liverpool	Table	Liverpool	Sheet	FALSE
3	London	Table	London	Sheet	FALSE
4	Sheet1	Table	Sheet1	Sheet	FALSE
5	Sheet2	Table	Sheet2	Sheet	FALSE
6	ManagerTable	Table	ManagerTable	Sheet	FALSE
7	Southampton	Table	Southampton	Sheet	FALSE

Figure 8.17 – The default name sheets

Click on the filter icon on the right-hand side of the Name column and this time we are going to select **Does Not Contain...** from the **Text Filters** menu. Type in Sheet, and then click **OK**.

We now only have the tables that we need to be imported, except for ManagerTable, but we can get rid of that in three steps:

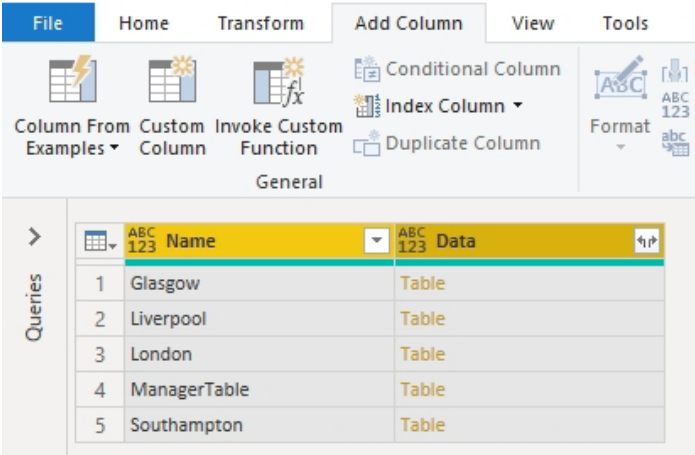


Figure 8.18 – The Expand button

We only need the **Name** and **Data** columns, so we can remove the other columns.

Click on the expand icon on the right-hand side of the **Data** column, as shown in the previous screenshot. We can now select the columns that we would like. We can select them all except the **Manager of Store** column, and then click **OK**:

Table.ExpandTableColumn(#"Removed Other Columns1", "Data", {"Date", "Store Zip Code", "Product", "Payment Type", "Units",

	ABC 123 Name	ABC 123 Date	ABC 123 Store Zip Code	ABC 123 Product	ABC 123 Payment Type	ABC 123 Units
1	Glasgow	11/12/2019	G3 8AG	Dyno1	Visa	
2	Glasgow	20/12/2019	G3 8AG	Dyno2	Paypal	
3	Glasgow	21/10/2019	ML12 6HD	Dyno2	Visa	
4	Glasgow	13/11/2019	G3 8AG	Dyno3	Cash	
5	Glasgow	25/11/2019	G62 8EP	Dyno2	Paypal	
6	Glasgow	07/12/2019	ML12 6HD	Dyno3	Visa	
7	Glasgow	04/11/2019	G62 8EP	Dyno1	Mastercard	
8	Glasgow	05/11/2019	G62 8EP	Dyno4	Debit Mastercard	
9	Glasgow	30/10/2019	G3 8AG	Dyno3	Cash	
10	Glasgow	15/11/2019	G3 8AG	Dyno3	Paypal	
11	Glasgow	29/10/2019	ML12 6HD	Dyno5	Cash	
12	Glasgow	01/10/2019	G3 8AG	Dyno3	American Express	
13	Glasgow	07/11/2019	G3 8AG	Dyno1	Mastercard	
14	Glasgow	17/11/2019	G3 8AG	Dyno1	Visa	
15	Glasgow	28/10/2019	G3 8AG	Dyno5	Debit Mastercard	
16	Glasgow	17/10/2019	G3 8AG	Dyno2	In Store	
17	Glasgow	01/11/2019	ML12 6HD	Dyno3	Cash	
18	Glasgow	01/10/2019	ML12 6HD	Dyno5	Visa	
19	Glasgow	06/10/2019	ML12 6HD	Dyno5	Cash	
20	Glasgow	26/10/2019	G3 8AG	Dyno2	Cash	
21	Glasgow	04/10/2019	G62 8EP	Dyno3	Cash	
22	Glasgow	21/11/2019	G3 8AG	Dyno3	Cash	

Figure 8.19 – The completed import

When looking at the completed import, there are a few minor things that we can do to improve it.

The first column is called Name, and we can double-click on the title to rename this column City. We can now go through and make sure that each of the different columns has the correct data type. City, Store Zip Code, Product, and Payment Type are all text data types. The quickest way to change all of these columns in one go is to select all of the appropriate columns by holding down *Ctrl* while selecting the columns. Right-click on any of the columns and then select **Text** from the **Change Type** menu, which will change all the selected columns at the same time, as follows:

The screenshot shows the Power Query Editor interface. The formula bar at the top displays the query name: `Table.RenameColumns("#Expanded Data",{"Name", "City"})`. The table below has the following columns: City, Date, Store Zip Code, Product, Payment Type, Units, Amount of Sale, and Revenue. A right-click context menu is open over the 'City' column header. The 'Change Type' option is selected, which has opened a submenu. In this submenu, the 'Text' data type is highlighted. The table data includes 26 rows of sample data with various values for each column.

	City	Date	Store Zip Code	Product	Payment Type	Units	Amount of Sale	Revenue
1	Glasgow	11/12/2019	G3 8AG	Dyno1		1	24.95	
2	Glasgow	20/12/2019	G3 8AG	Dyno2		1	28.75	
3	Glasgow	21/10/2019	ML12 6HD	Dyno2		1	28.75	
4	Glasgow	13/11/2019	G3 8AG	Dyno3		1	43.96	
5	Glasgow	25/11/2019	G62 8EP	Dyno2		9	258.75	
6	Glasgow	07/12/2019	ML12 6HD	Dyno3		1	43.96	
7	Glasgow	04/11/2019	G62 8EP	Dyno1		1	24.95	
8	Glasgow	05/11/2019	G62 8EP	Dyno4		12	59.4	
9	Glasgow	30/10/2019	G3 8AG	Dyno3		1	43.96	
10	Glasgow	15/11/2019	G3 8AG	Dyno3				307.72
11	Glasgow	29/10/2019	ML12 6HD	Dyno5				167.7
12	Glasgow	01/10/2019	G3 8AG	Dyno3				263.76
13	Glasgow	07/11/2019	G3 8AG	Dyno1				149.7
14	Glasgow	17/11/2019	G3 8AG	Dyno1				24.95
15	Glasgow	28/10/2019	G3 8AG	Dyno5				27.95
16	Glasgow	17/10/2019	G3 8AG	Dyno2				28.75
17	Glasgow	01/11/2019	ML12 6HD	Dyno3				43.96
18	Glasgow	01/10/2019	ML12 6HD	Dyno5				335.4
19	Glasgow	06/10/2019	ML12 6HD	Dyno5				335.4
20	Glasgow	26/10/2019	G3 8AG	Dyno2	Cash			115
21	Glasgow	04/10/2019	G62 8EP	Dyno3	Cash			43.96
22	Glasgow	21/11/2019	G3 8AG	Dyno3	Cash			43.96
23	Glasgow	07/11/2019	G3 8AG	Dyno1	Debit Visa			149.7
24	Glasgow	20/11/2019	G3 8AG	Dyno3	Debit Visa			527.52
25	Glasgow	20/12/2019	G62 8EP	Dyno1	Mastercard	4		99.8
26	Glasgow	28/11/2019	G3 8AG	Dyno1	Debit Mastercard	1		24.95

Figure 8.20 – Changing to the Text data type

The second column needs to change to the **Date** data type and the Units column needs to change to the **Whole Number** data type:

Units	COGS
1	13.54
1	14.82
1	15.54
1	22.93
9	127.71
1	21.46
1	13.14
12	39.24
1	21.4
7	50.86
6	77.52
6	83.56
6	82.62
1	15.32
1	13.69
1	14.87
12	22.14
12	49.04
4	57.04
1	51.72
1	22.79
6	22.58
12	86.52
	56.68

Figure 8.21 – Fixed decimal number

The last three columns need to be currencies, but in Power BI, we need to change the data type to **Fixed decimal number**, which is Excel's version of currency. One of the major differences between **Decimal Number** and **Fixed decimal number** is that the latter option will only allow 4 decimal places, while **Decimal Number** can have up to 15 decimal places. A quick way to select the last three columns is to select the first column, press and hold *Shift*, and then select the last column.

We have now appended all the different tables, but we might need an additional column to work out the actual cost per item, which would include the discount.

We will insert a custom column by selecting **Custom Column** from the **Add Column** menu. Call the new column name `NetSales` and type the `Number.Round([Amount of Sale]*(1-[Revenue Discount]), 2)` formula, as follows:

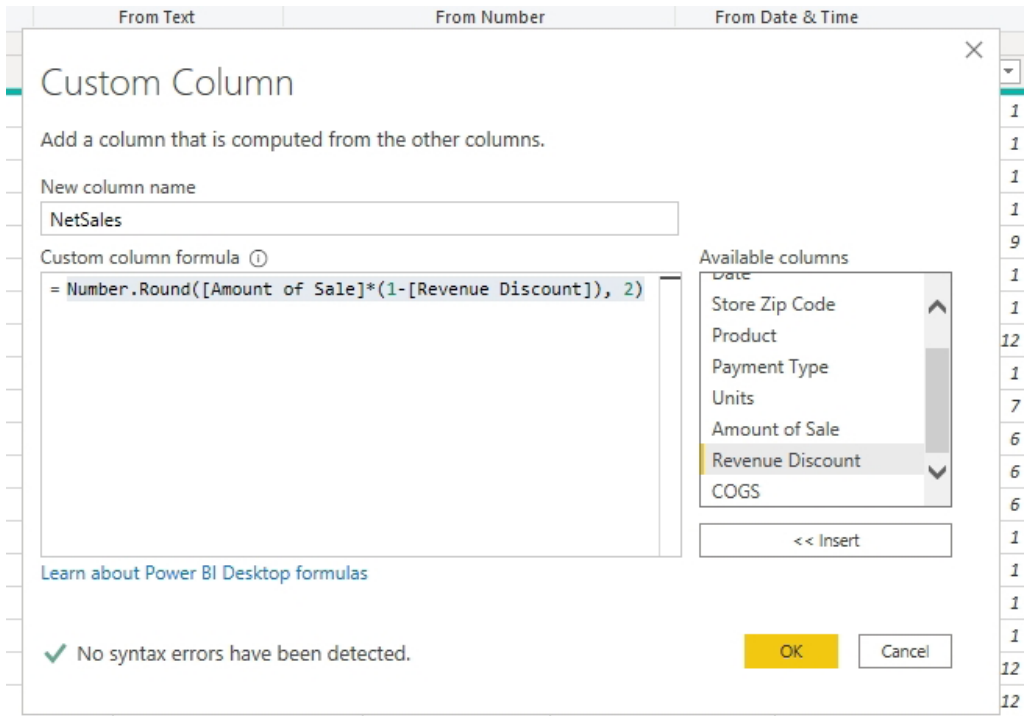


Figure 8.22 – Adding the NetSales custom column

The formula is `Amount of Sale` multiplied by the discount. `Number.Round` rounds the answer, and 2 at the end of the argument tells Power BI to round the number to 2 decimal places.

Once completed, click on **OK** and we can change the data type to **Fixed Decimal Number**.

As we now have this new column, we don't need the `Amount of Sale` or `Revenue Discount` columns, so we can delete them. The table is now complete, and we can load it to the data model by selecting **Close and Apply** from the **Home** menu.

At this point, we have transformed our table, deleted the information we no longer need, and added columns with additional information. As mentioned previously, although we have used a folder for this example, the files could be online, on a website, or in a SQL server. We can now use this query in our data model, which completes the first step of the process. The next section deals with the second step of the process.

Using Power BI to add data to a data model

This section deals with creating relationships and adding more data, specifically measures and calculated columns, using DAX, which allows us to modify and create additional columns that we will need in the dashboard.

When we look at Power BI with the query that we created from the previous section, we can see the name of our table, `SalesData`, on the right with all of the different fields:

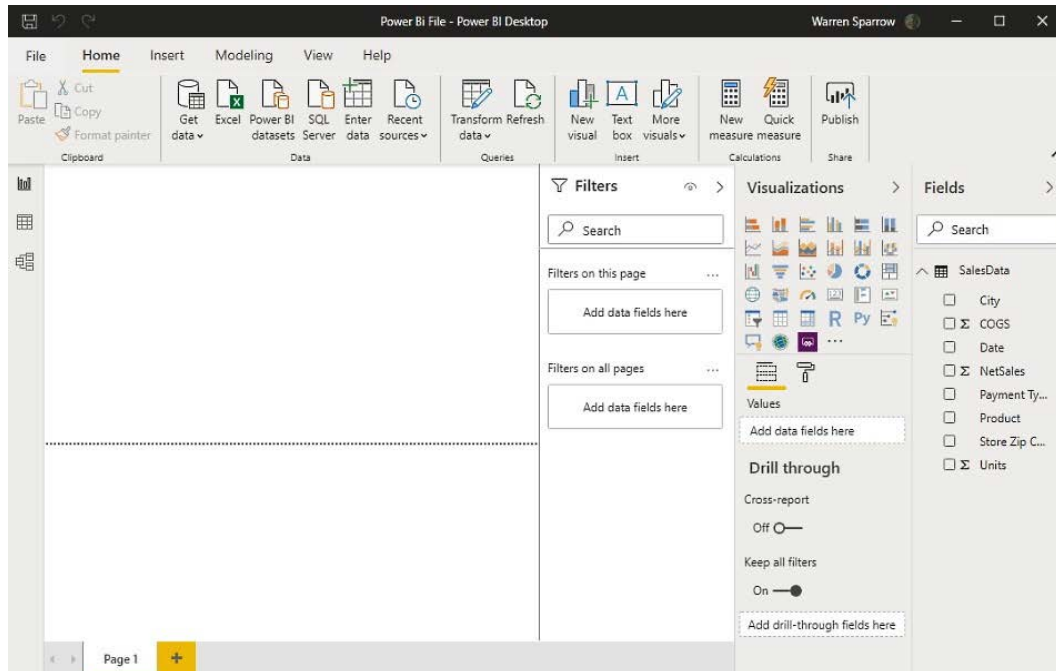


Figure 8.23 – The NetSales table

On closer inspection of `SalesData`, the Σ symbol on the left-hand side of some of the fields means that these are numbers. On the left-hand side of the window, we have the **Report**, **Data**, and **Model** views:

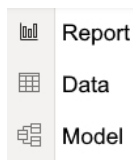


Figure 8.24 – The different views

When we first opened Power BI, we were in the **Report** view by default.

If we click on the **Model** view, we can see the different relationships that we currently have between our tables. At the moment, we currently do not have any relationships, but here, we are going to add another table and build a relationship with a file from a SharePoint server, although you can follow the same steps if you were using a SQL or Azure server as well. From the **Home** tab, select **Get Data** and click **More... | SharePoint folder | Connect**.

You will need to enter your authentication details and then click **Connect**. (Please refer to *Chapter 2, Power Pivot Basics, Inadequacies, and Data Management*, to see how to connect to various sources.) Once you click on the relevant file, click on the table that you need and then select **Transform Data**. It might be necessary to neaten the file up by removing unwanted columns and so on before selecting **Close and Apply**.

When we click on the **Model** view, we can see two different tables—one is the fact table with all the sales transactions and the other is our dimension table with the manager of the store.

We will create a relationship between **Store Zip Code** and **Zip Codes**. Click and hold on **Store Zip Code**, and then release the mouse on **Zip Codes** in the **ManagersTable** window.

This will create a one-to-many relationship between these two tables, which is depicted in the following screenshot by the **1** symbol on the left and the ***** symbol on the right, which indicates many, thus a one-to-many relationship. This means that one row, which is **Zip Codes**, could be linked to many rows, **Store Zip Code**:

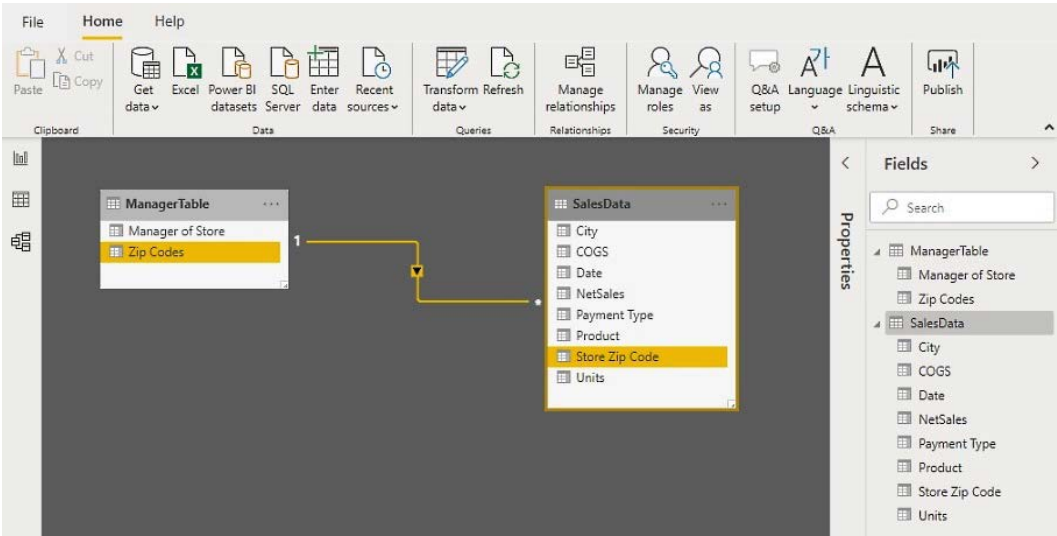


Figure 8.25 – The one-to-many relationship

If we click on the **Data** view, we can see the tables and fields in this data model. Now, when we look at preceding screenshot, we want to add another column to our SalesData table. The number of units sold will depend on whether this column will be retail or wholesale.

There are two different ways in which you can create the column. The first is by right-clicking on **SalesData** on the right and then selecting **New column**; otherwise, you can select **New column** in the **Table tools** menu, as follows:

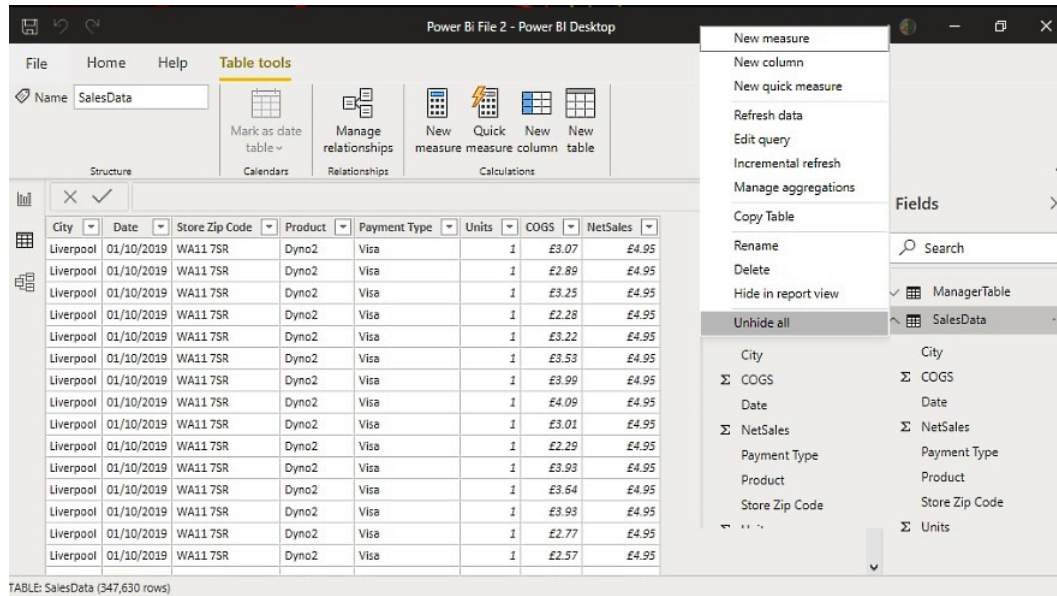


Figure 8.26 – The table tools menu

Note

This is the latest version of Power BI Desktop at the time of writing, April 2020. If you have a previous version, then there might be a **Modeling** menu instead of **Table Tools**.

After selecting **New column**, **Column =** will automatically appear in the formula bar. We are going to type in an **IF** formula that will look at the number of units that were sold and if it is less than 5, the column will say **Retail**; otherwise, it will say **Wholesale**.

Delete **Column =** and type in the following:

```
RetailWholesale = IF(SalesData[Units]<5, "Retail", "Wholesale")
```

When typing the formula, after typing in `=If (`, press the `S` key on your keyboard and the different fields will come up with the `SalesData` table name. You can then click on **Units**, which is the field that we would like.

From the condition we stated earlier, it is evident that anything less than 5 will say **Retail** and anything above 5 will say **Wholesale**. This formula is based on the row context, as it will look at each unit in each specific row and then apply the `IF` formula to that row.

If you take a look at the following screenshot, you will notice that there is a calculated column symbol (**fx**) in the formula bar to the left-hand side of the `RetailWholesale` field. This is due to the `IF` condition that we created when adding the column:

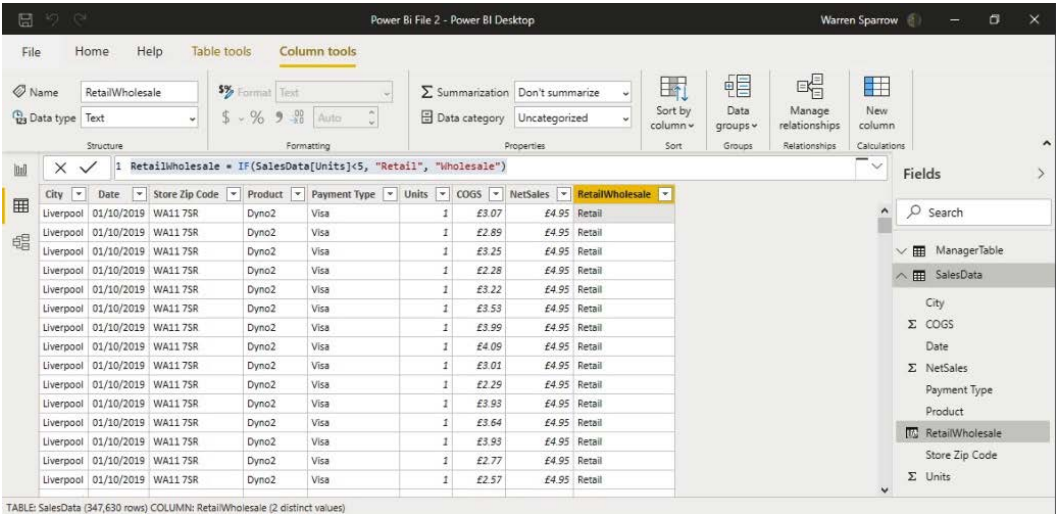


Figure 8.27 – The calculated column

We now have the standard columns that we require. In the next section, we will add measures, which are columns that can carry out more specific mathematical equations.

We are now going to create a few measures in the `SalesData` table. If you have not used measures before, they are used to calculate aggregates—for example, the net revenue, gross profit, sum, the average of a column, and so on. Measures are only calculated at the time of the query, so they are not part of the stored database. Measures are also available to all calculations and visualizations in the model. As they are not stored in memory, they are slightly faster to execute.

The measures that we are going to create for this table are `GrossProfit`, `GrossProfit%`, `TotalCODSold`, and `TotalRevenue`.

From **Table Tools**, select **New Measure** and type `= SUM(SalesData[NetSales])` into the formula bar. Before pressing **Enter**, change this to a currency type and set the decimal places to 0. We are going to do exactly the same and create another measure called **TotalCOGSold** with the `TotalCOGSold = sum(SalesData[COGS])` formula, and we will also change this to a currency type with no decimal places:

The screenshot shows the Power BI Desktop interface. The 'Measure tools' ribbon is active. The formula bar contains the measure definition: `TotalRevenue = SUM(SalesData[NetSales])`. The 'Name' field is set to 'TotalRevenue', the 'Currency' is set to '£', and the 'Data category' is 'Uncategorized'. The 'Fields' pane on the right shows the 'SalesData' table with various fields, and the 'TotalRevenue' measure is selected. Below the table, a status bar indicates 'TABLE: SalesData (347,630 rows) COLUMN: TotalRevenue (0 distinct values)'.

City	Date	Store Zip Code	Product	Payment Type	Units	COGS	NetSales	RetailWholesale
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.07	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£2.89	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.25	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£2.28	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.22	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.53	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.99	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£4.09	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.01	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£2.29	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.93	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.64	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£3.93	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£2.77	£4.95	Retail
Liverpool	01/10/2019	WA11 7SR	Dyno2	Visa	1	£2.57	£4.95	Retail

Figure 8.28 – The new TotalRevenue measure

Create another measure called **GrossProfit** and type `GrossProfit = [TotalRevenue] - [TotalCOGSold]` into the formula bar. Then, add another measure called **GrossProfit%** and type `GrossProfit% = [GrossProfit] / [TotalRevenue]` into the formula bar. This time, we need to set this to a percentage and keep it within 2 decimal places.

At this point, we have two tables (**ManagerTable** and **SalesData**), we have a calculated column (**RetailWholeSale**), and we have four measures (**GrossProfit**, **GrossProfit%**, **TotalCOGSold**, and **TotalRevenue**).

We now have everything that we need for our visualization dashboard. The next section goes through the process of creating a visualization dashboard with the different options we have.

Selecting data visualization, a dataset, and an appropriate chart

In the previous section, we have imported all of our data into a Power BI data model. This section deals with how to create an interactive dashboard that will visually show us the data that we can use to analyze the trends.

In Power BI, click on the **Report** view on the left-hand side, which will bring up a blank working space, which is where we can create our visualizations. On the right-hand side are all the different visualizations that we can use, as in the following screenshot:

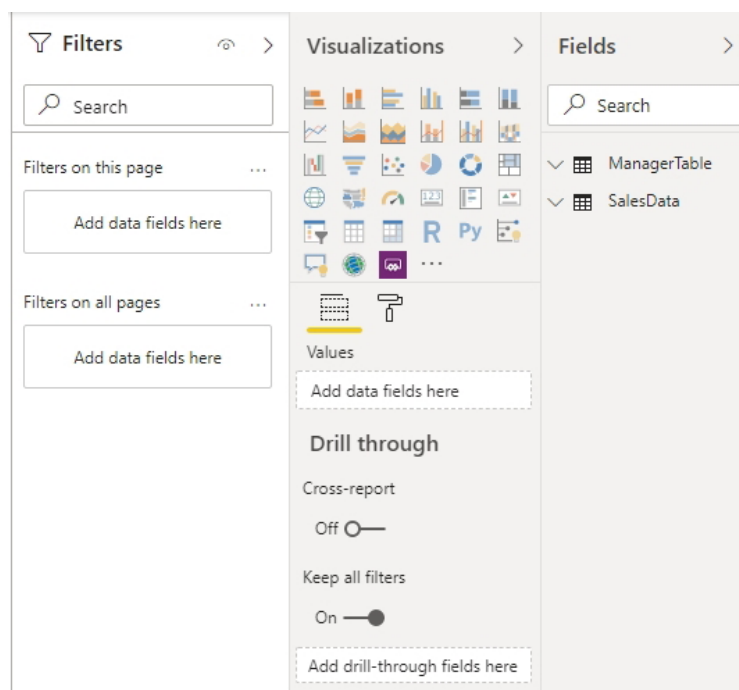


Figure 8.29 – The visualizations menu

Most of these are self-explanatory, but we will be using a few different ones so that we can learn how they work.

Note

The version of Power BI for Desktop that you're running will determine which of these visualizations you have, and some of the icons might look slightly different from the figures that are displayed here. This is the latest version at the time of writing, April 2020.

We are going to start with one of the great features that has been introduced that shows you geographically where in the world your data comes from—the map visualization:

1. Click once on the **Map** visualization icon and it will appear in the top-left corner of the screen.
2. To associate data with it, select **Zip Codes** in the `ManagerTable` table and **GrossProfit** in the `SalesData` table, as follows:

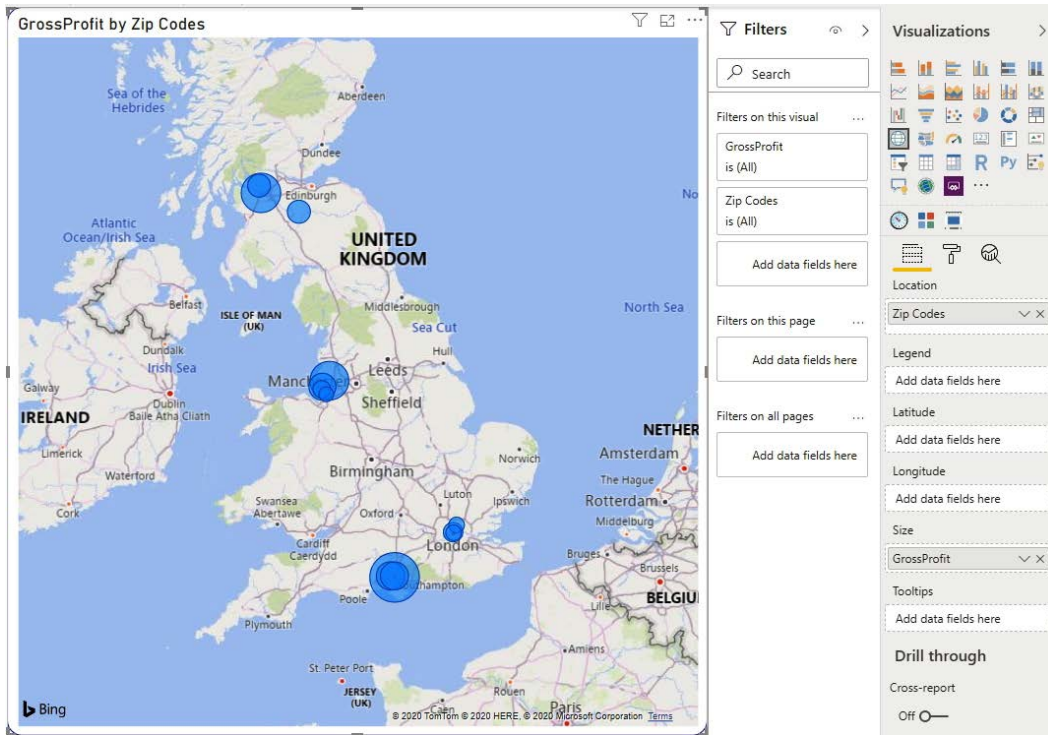


Figure 8.30 – Map visualizations

We can now see the gross profit for each of the different places based on their postal or ZIP codes. The larger the size of the circle, the larger the gross profit is. Quickly, by looking at this, you can see that there are three areas that are doing well—namely, Southampton, Manchester, and Glasgow—while London seems to be doing less well.

There are several different options that we have with regards to what our visualization looks like:

- If you click on the paint bucket, which is the **Format** tool, in the **Visualizations** menu, you can change the size of the bubbles and you can create a border around the visualization.

- Click on the **Clustered** bar chart option in the **Visualizations** menu and select **Manager of Store** in the ManagerTable table and **GrossProfit** in the SalesData table. This is how the chart appears:

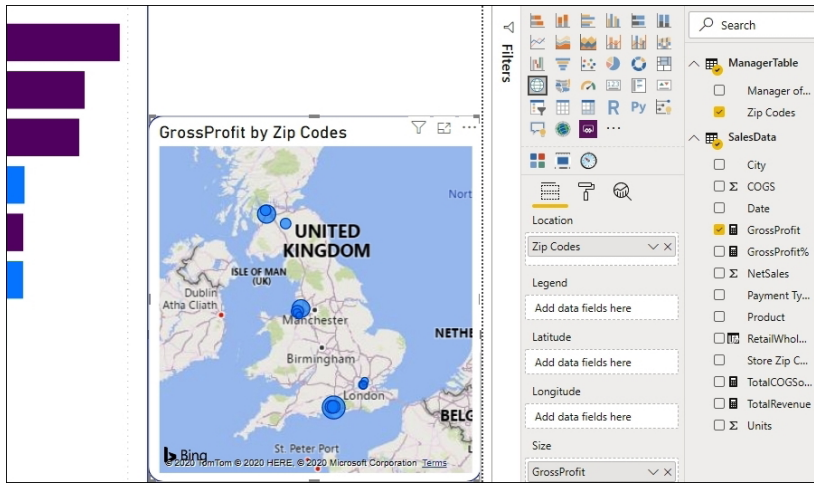


Figure 8.31 – Clustered bar chart visualization

- In the **Format** section, you can change the color of the chart. This can help you distinguish between different sales departments or perhaps different commission rates. As part of the interactive visualizations, if I click on a specific salesperson, it will automatically adjust the map to show me where that salesperson is based.
- When we click on **Sallim Simmonds**, the map changes to **London**, and you can then see that specific salesperson's location:

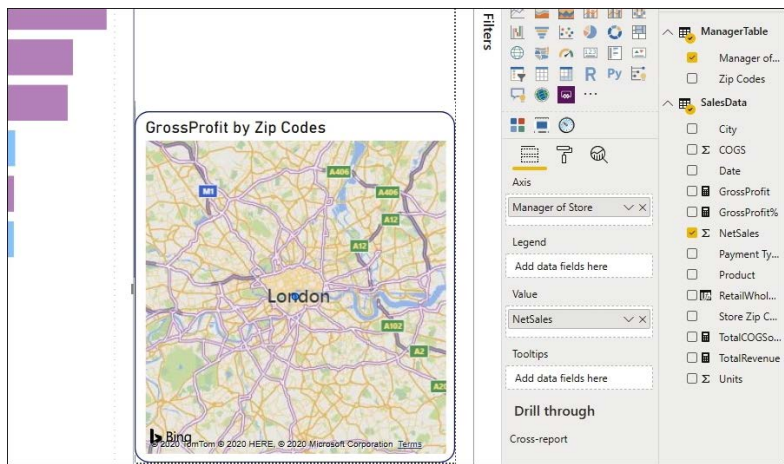


Figure 8.32 – Sallim Simmonds' postcode

Click on **Line and clustered column chart** and select **Product**, **GrossProfit** and **GrossProfit%** from the `SalesData` table:

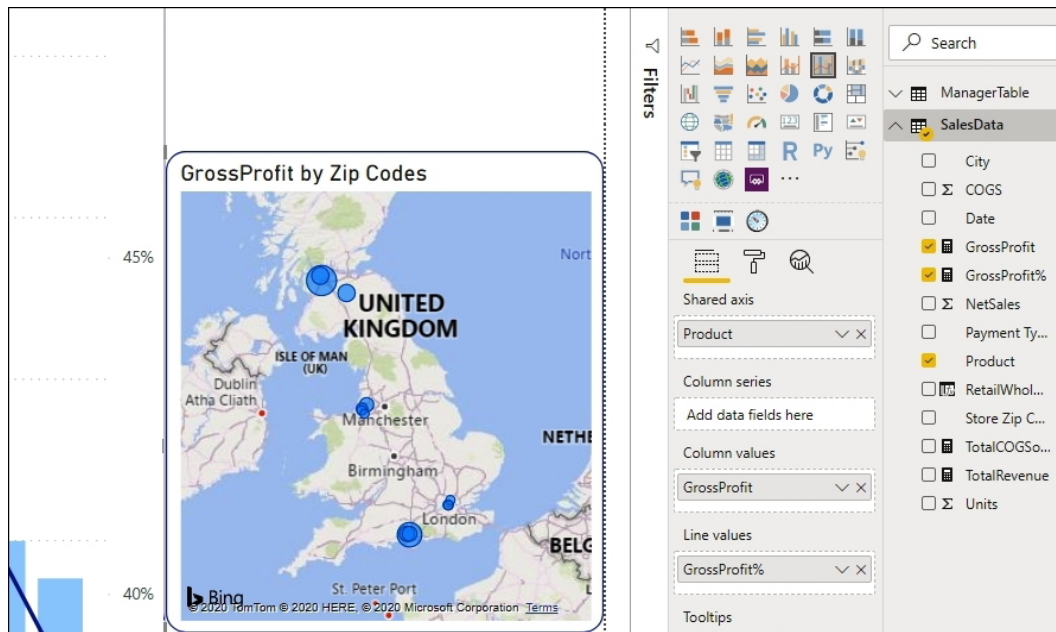


Figure 8.33 – A line and clustered column chart

You will notice that this is not displayed the same as the preceding chart as at the moment, both of these values are sitting in **Column values**. Click and drag **GrossProfit%** to the **Line values** and this will then display the two different charts.

Create an identical line and clustered column chart and select **Payment Type**, **GrossProfit**, and **GrossProfit%** from the `SalesData` table.

Do the same thing as before and move **GrossProfit%** to **Line values**. It is possible to move it directly from its field in the table to this position as well. Resize and rearrange the different windows to the position of your choice:

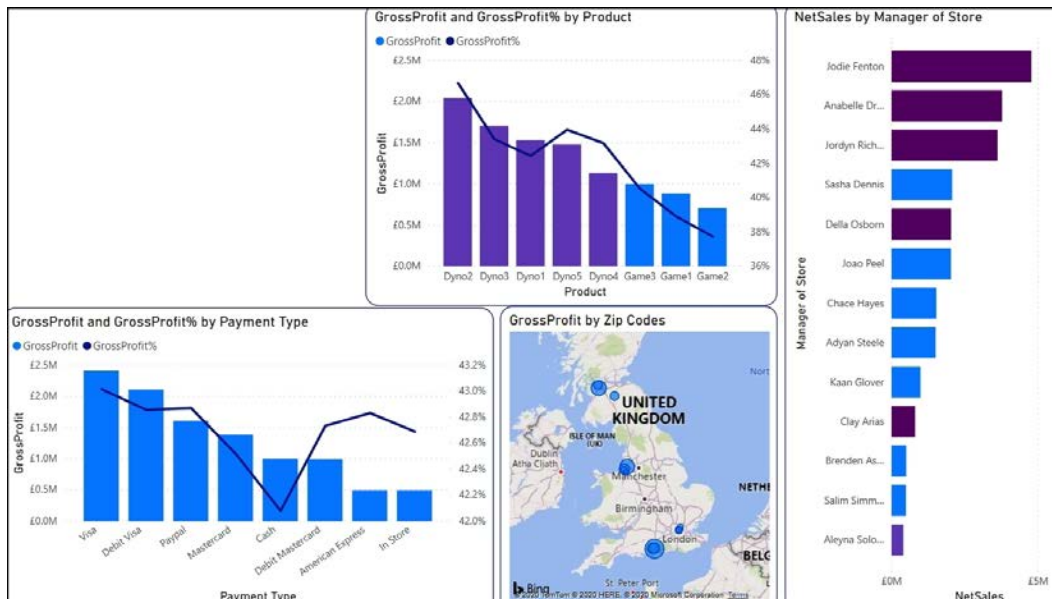


Figure 8.34 – The second line and clustered column chart

We are going to create a few slicers that work in the same way as they do in Excel PivotTables. The first one is a slicer for the different cities.

Click on **Slicer** in the **Visualizations** menu and select **City** from the **SalesData** table. Move and resize the window to a place on the dashboard. Create a second slicer for **Payment Types** by selecting **Payment Types** from the **SalesData** table:

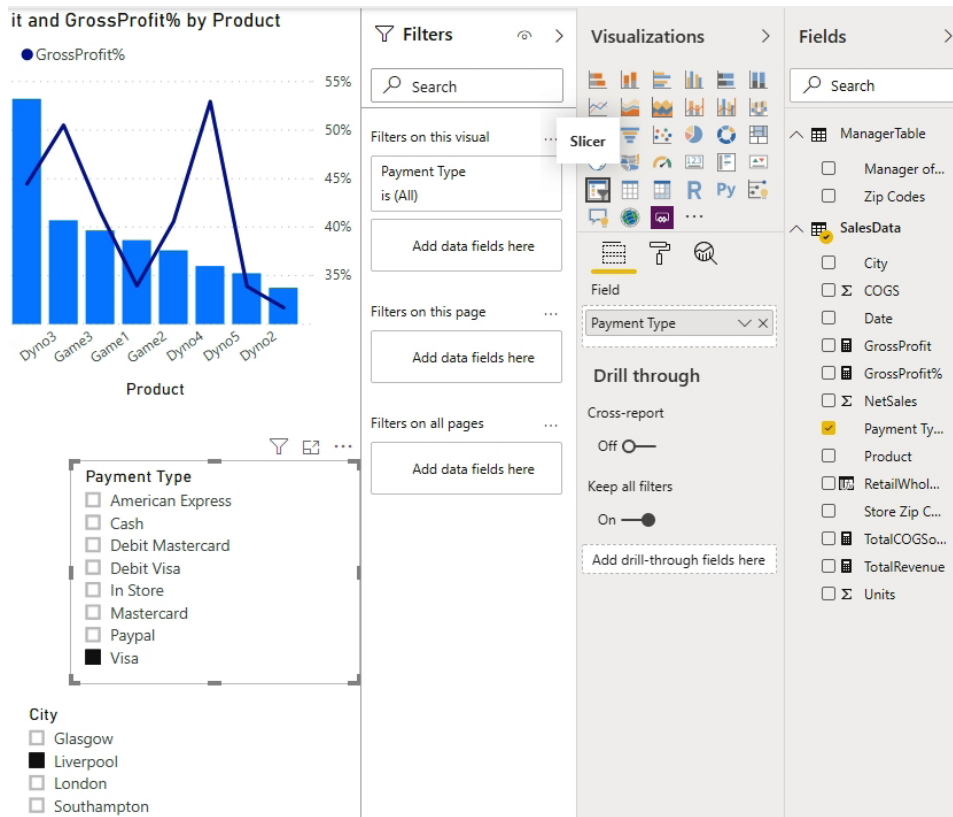


Figure 8.35 – Adding different slicers

There are more things we are going to do to complete our visualization dashboard, namely, create a multi-row card and a **Q&A** object:

- Click on the **Card** icon and select **TotalRevenue**. Change the font, color, and size to make it stand out from the other information.
- A multi-row card is the same as a card except it allows you to display more than one item at the same time.
- Select **Multi-row card** and click on the other three measures that we created earlier. Using **Format**, it is possible to change the color of the background and the **Data** labels, as well as the category color, size, and font.

The last visualization we will be adding is **Q&A**. I find this useful as it allows you to ask a question about any of your tables and fields and the value will be displayed. I want to point out that, by default, it rounds numbers up or down so that they look better. For example, if you asked "What is the total revenue?," it would come up with a figure of 24 million, while the revenue is actually 24.489472 million.

Select the **Q&A** icon from the **Visualization** menu and then type in the What is Netsales by city? question, as follows:

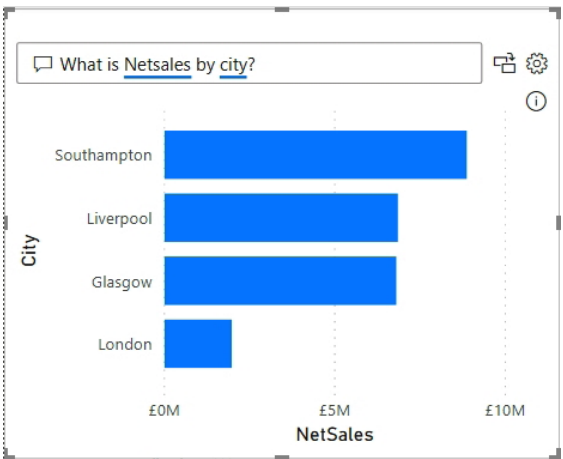


Figure 8.36 – The Q&A visualization with the What is the Netsales by city? question

It is now possible to select the two windows with the arrow, and this will then turn this **Q&A** visualization into a standard visual, which we will select. We have now looked at all the possible types of visualization. Refer to the following screenshot:

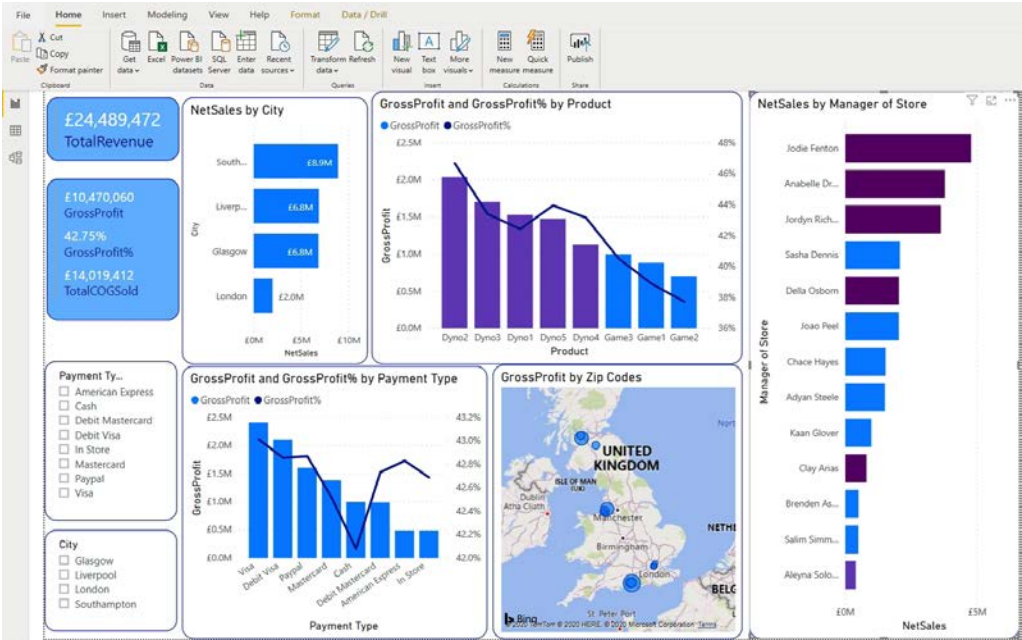


Figure 8.37 – The completed dashboard

We now have a completed dashboard. When you are clicking around, you will realize how powerful it really is. Every time you click on an element, it will create a filter for that element. You can even click on a product and it will show you the total revenue, the net sales by city, and your percentage gross profit, all live.

The next section deals with the different options for where and how we can save dashboards.

Saving, publishing, and sharing a dashboard

This section deals with how to save and publish your interactive dashboard. Before beginning, you need to know that once you publish your dashboard on the web, other people can see the data and run the reports without any authentication, so make sure that this complies with your organization's policy.

To publish a dashboard, do the following:

1. Go to **File | Publish | Publish to Power BI**.
2. Select a location.

There is also a **Publish** button on the **Home** tab, which is a bit easier to access. The following screenshot shows this option:

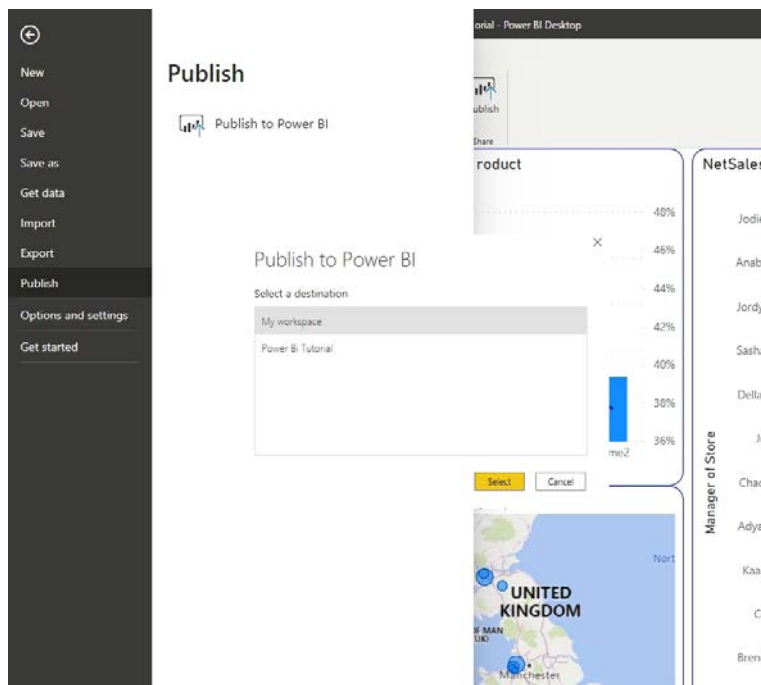


Figure 8.38 – The Publish to Power BI window

3. By default, it will show **My workspace**:

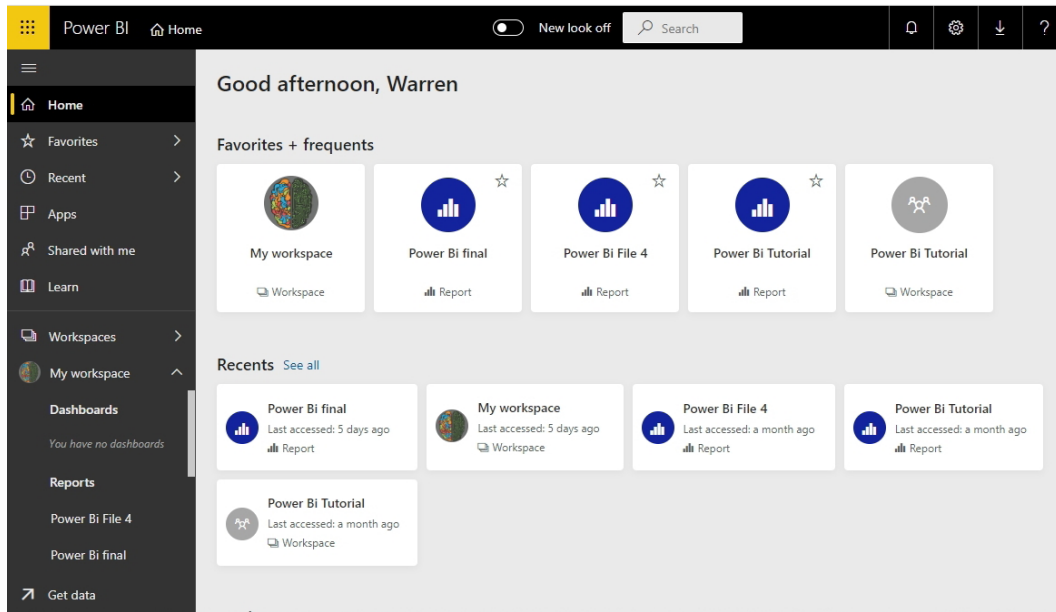


Figure 8.39 – The online Power BI portal

4. Click on **Select** and after a few seconds, you will get a message to say that this has been successful.
5. Click on **Got it** to get rid of the window.

You have now published the work, but it is not obvious where you have published it to.

There are two different options that you have, depending on your subscription. Go to office.com or <https://powerbi.microsoft.com/en-us/landing/signin/> and sign in with your Microsoft account. This needs to be the same account that you used to log in to Power Query.

Once you are signed in, you will see **My workspace**, which is where your data and reports are. When you look around, you will notice that your uploaded file can be found in two places, as shown:

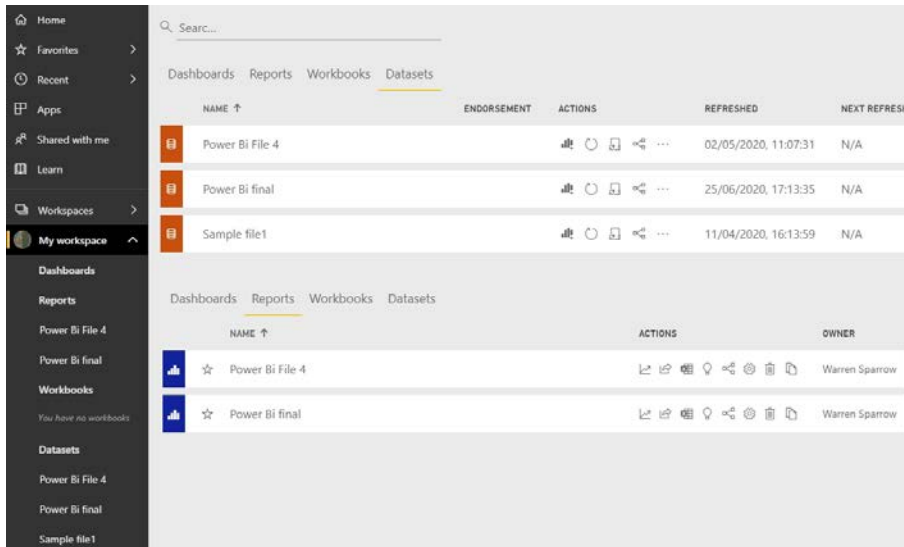


Figure 8.40 – The Power BI files

If you click on the **Datasets** menu, this is where the actual datasets with all the data sits. You have a few options regarding security, permissions, and downloading the **.pbix** Power BI file:

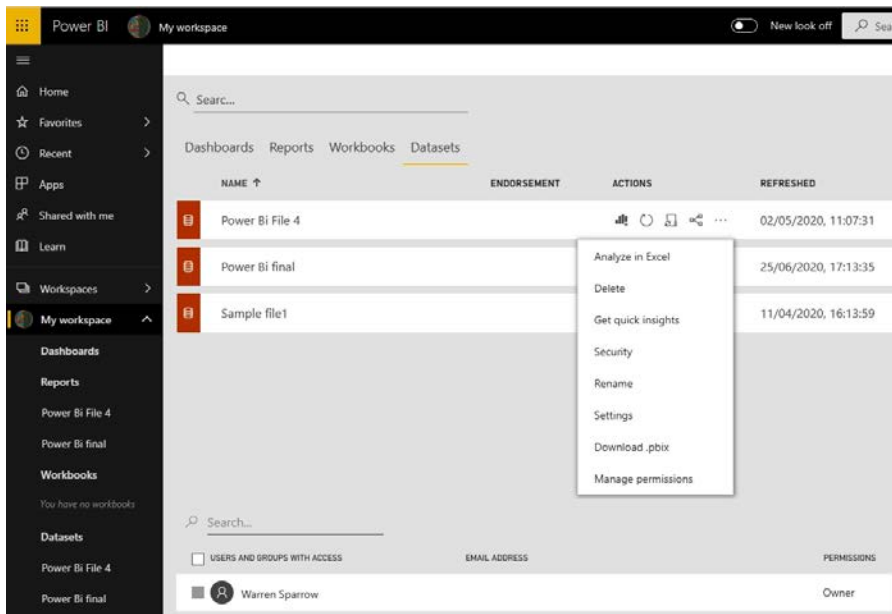


Figure 8.41 – The Datasets options

Although you could then send or share this file with another person, this is not the ideal way of sharing it. One of the reasons for this is that each time there is an update, you would then have to resend or share the file again.

Depending on your license, you may be able to share your datasets with anyone. You can only do this if you have the Pro version, however.

If you click on the **Reports** tab, there are more things that you can do concerning the report that you created, as shown:

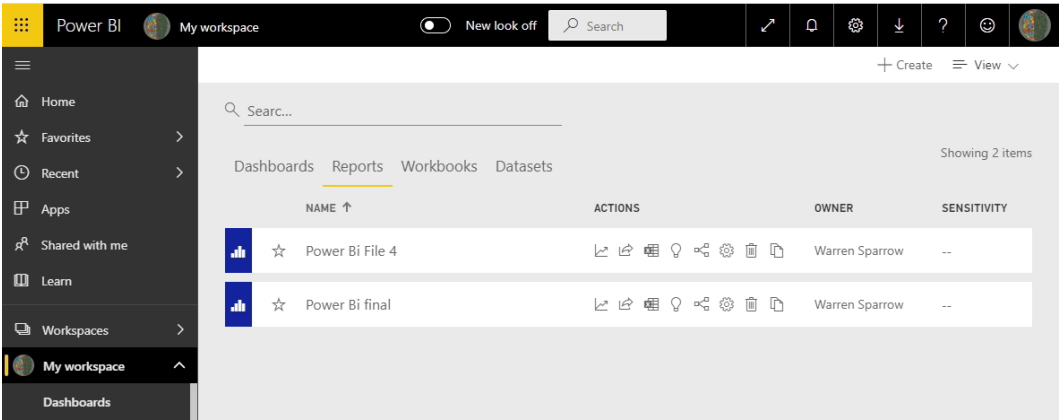


Figure 8.42 – The Reports options

In the **ACTIONS** section, we can share, analyze, view related versions, change settings, delete, and save a copy of the report. Once again, your license type limits what you are able to do.

In the next section, we will look at sharing all of this content with other people both inside and outside of your organization.

Sharing a dashboard

There are several different things that we can share. We can share reports, dashboards, and some data, and we can also decide on various levels of sharing. The basic version allows you to share basic information, but the professional version allows more functionality.

If you click on **Pro trial**, you will get 60 days free and you do not have to give your credit card number and other details. With an upgrade to Pro, there are many more options that you can do. One of the best options in the Pro version is the following option to share your report with multiple people:

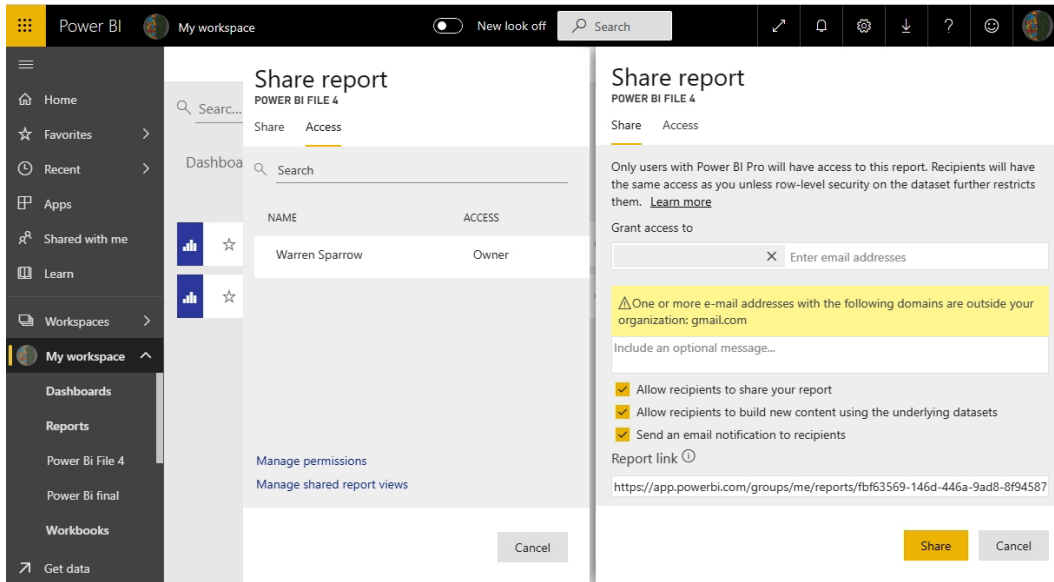


Figure 8.43 – The sharing options

If you select the **Share** button in the top-right corner of the screen, you can share your report with specific people and depending on the options that are selected, they can share your report and allow other people to use your datasets to build additional reports, and you can send an email notification to the people that you are sharing with.

Note

Be aware that people can use your data in their own sets, so make sure that this is allowed.

Once a person has access, it is possible to change their level of access by selecting **Access** and then clicking on the three dots to the right-hand side of their name. This allows you to remove access or change their access to **Read** or **Read and reshare**:

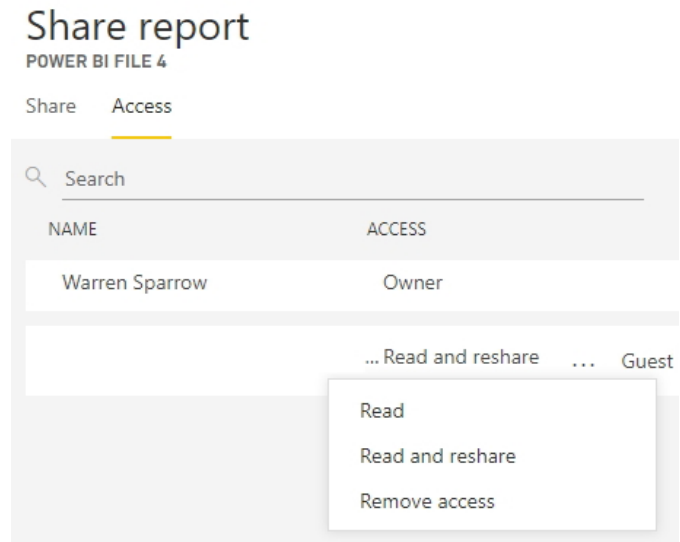


Figure 8.44 – The ACCESS sharing options

There is also a very quick way to share and that is with a QR code. If you click on the three dots to the right of **Share** in the top-right corner, you can select **Generate QR Code**:

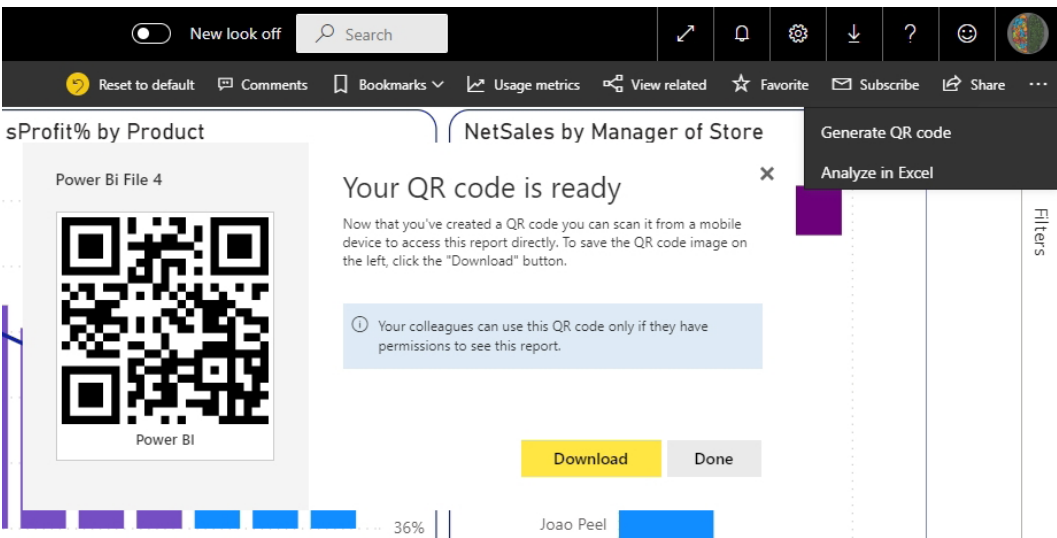


Figure 8.45 – Generate QR Code

Once the code is generated, you can send it to another person or upload it to a social media platform to send it to multiple people if you would like them to view the dashboard.

Both of the preceding methods are the basic sharing options for dashboards and reports. The great thing with these sharing options is that they are very quick to do and easy to set up.

There is, however, a big disadvantage with this basic sharing options in that the people you share with cannot edit the information. The other disadvantage is that you can only share one dashboard at a time, which could be a problem if you have multiple dashboards.

If you click on the **File** menu, this gives you many additional options for sharing and embedding your dashboard:

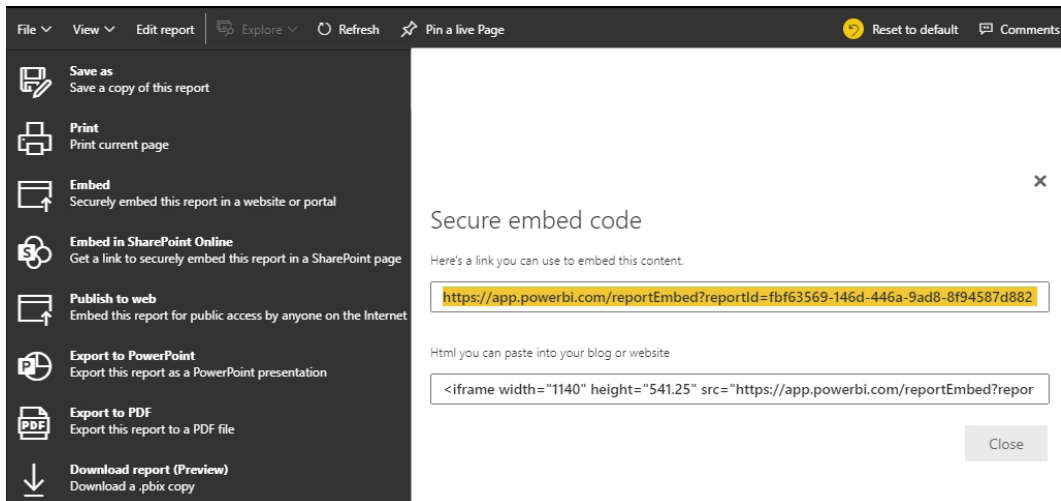


Figure 8.46 – The File menu with sharing options

From the **File** menu, we can share both the dashboards and the reports. There are a few options that are very self-explanatory, such as **Save As**, **Print**, **Export to PDF**, and **Download report**, which are all non-interactive and basically give you the equivalent of a hard copy of the reports or dashboards. Even **Export to PowerPoint** just provides an image of the dashboard and is not interactive:

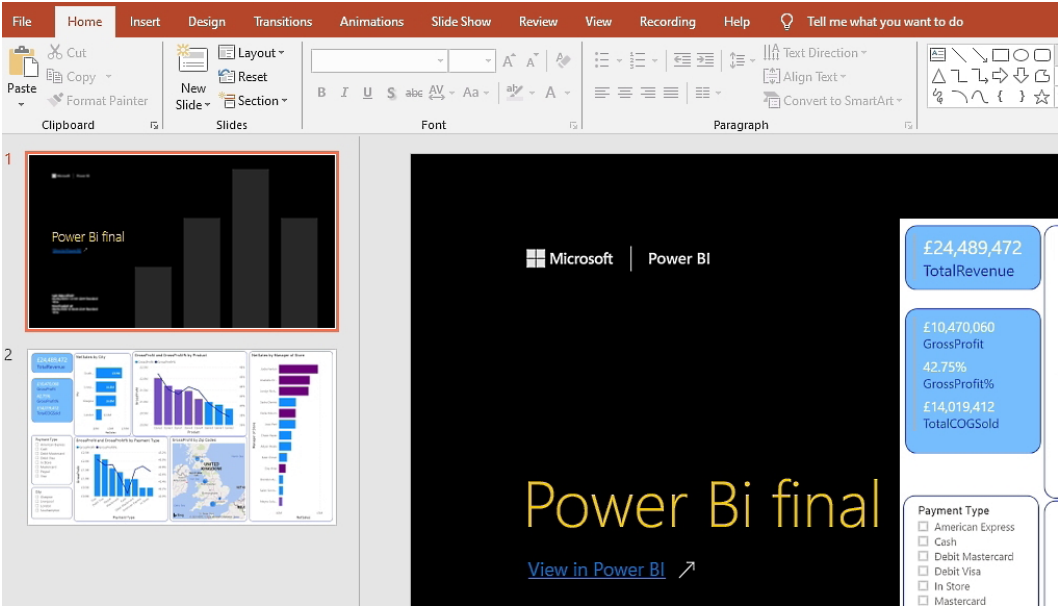


Figure 8.47 – A PowerPoint export

The other options in the **File** menu offer more interactivity. **Embed**, **Embed in SharePoint Online**, and **Publish to web** are all options that allow full interactivity with the dashboard.

These options are all slightly different but do the same thing, which is either giving you code to embed into a website or a web address:

- When you click **Publish to web**, you are given a link that you can send to other people in an e-mail (<https://app.powerbi.com/reportEmbed?reportId=fbf63569-146d-446a-9ad8-8f94587d8824&autoAuth=true&ctid=85931486-6860-4c35-8c73-06af6716f695&config=eyJjbHVzdGVyVXJsIjoiaHR0cHM6Ly93YWJpLXVrLXNvdXRoLWl0cHJpbWVyeSlyZWVpcVjC5hbmFseXNpcy53aW5kb3dzLm5ldC8ifQ%3D%3D>) and you are given code to embed into a blog or website (`<iframe width="1140" height="541.25" src="https://app.powerbi.com/reportEmbed?reportId=fbf63569-146d-446a-9ad8-8f94587d8824&autoAuth=true&ctid=85931486-6860-4c35-8c73-06af6716f695&config=eyJjbHVzdGVyVXJsIjoiaHR0cHM6Ly93YWJpLXVrLXNvdXRoLWl0cHJpbWVyeSlyZWVpcVjC5hbmFseXNpcy53aW5kb3dzLm5ldC8ifQ%3D%3D" frameborder="0" allowFullScreen="true"></iframe>`).
- When looking at the embedded code, it is important to see that the width is 1140, so when you upload it to a blog, it might be necessary to change this to something a bit smaller. If this is done, the dashboard becomes smaller and can pose some legibility problems. So, ensure that you use a proper scale.
- **Publish to web** is a free way of sharing, but remember that any person with the link can view it as there is no security or authentication.

- **Embed in SharePoint Online** works in a similar way to **Publish to Web**—you are given a URL—but the one difference is that you can then give specific people access to the file through SharePoint:

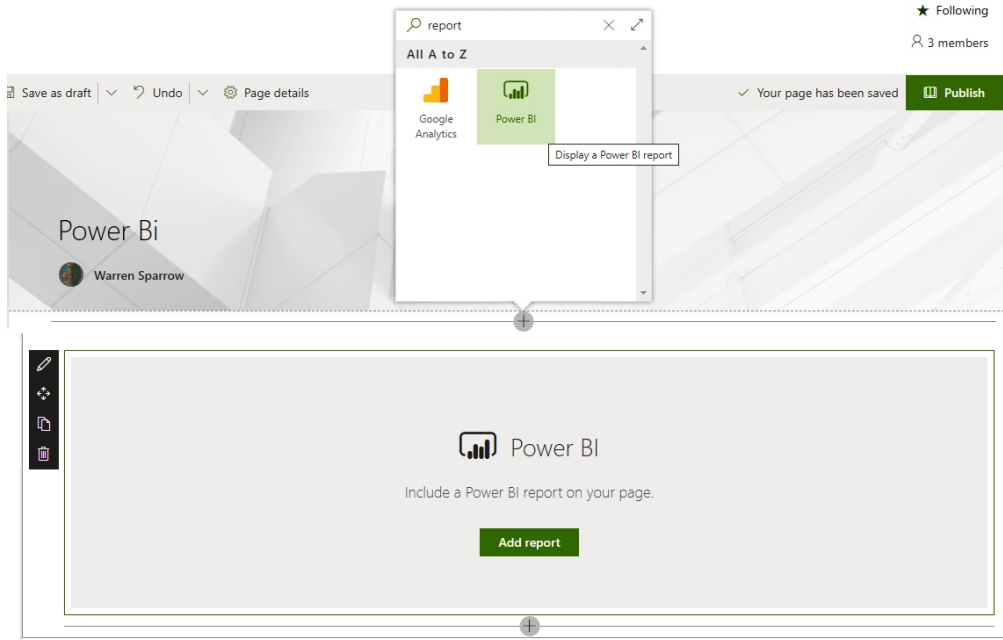


Figure 8.48 – Using SharePoint Online to embed Power BI

From SharePoint Online, create a page and then select **Add a new web part in column 1** and choose the **Power BI** option. Click on the **Add report** button and insert the link that was provided earlier:



Figure 8.49 – The linked report

Once the link is inserted, it will automatically come up with the page name and you can then choose whether you want a 16:9 or 4:3 aspect ratio.

Once completed, it will automatically appear and then you can publish it. Refer to the following screenshot:

Site Pages

✓	Name	Modified By	Modified	Created
✓	Created By : System Account (1)			
	Home.aspx	System Account	March 7	March 7
✓	Created By : Warren Sparrow (6)			
	Power Bi Dashboard.url	Warren Sparrow	May 8	May 8
	vpam1ngo.aspx	Warren Sparrow	May 8	May 8
	Power-Bi-Dashbaord.aspx	Warren Sparrow	May 8	May 8
	The-Power-Bi-Dashboard.aspx	Warren Sparrow	May 8	May 8
	sjeqlje3.aspx	Warren Sparrow	12 minutes ago	12 minutes ago
	Power-Bi.aspx	Warren Sparrow	About a minute ago	11 minutes ago

Link settings

Who would you like this link to work for? [Learn more](#)

- Anyone with the link
- People in with the link ✓
- People with existing access
- Specific people

Other settings

- Allow editing ✓

Apply Cancel

Figure 8.50 – The sharing options in SharePoint Online

From the preceding screenshot, we see that within SharePoint Online, you can change the permissions to certain people or your entire organization as with any other file. Using this option is fairly easy as you do not need to write any code; the only thing you need is the URL.

Embed allows you to share your reports through a web portal, but only authorized users have access to the data:

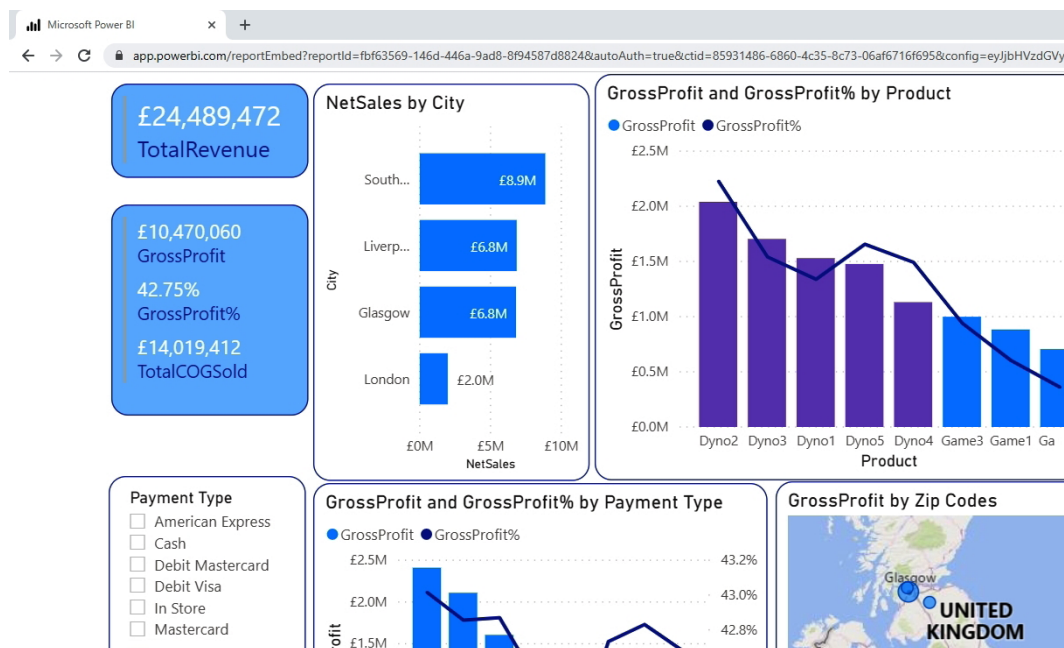


Figure 8.51 – Embedding a report

To sum up, the different ways that you can share are as follows:

- The basic sharing options make it quick and easy to share with other people. They are also great for testing reports and dashboards.
- **Publish to web** is a free service of sharing dashboards and datasets and is useful when sharing non-confidential information.
- Using **SharePoint Online** is an easy way to share your information with additional security and authorization. This is an excellent option to use when sharing within an organization.
- **Embed** allows you to use your on-premises SharePoint environment or a web application with security, but without needing a web developer to write the necessary code for it.

We have now created a dashboard and shared it with the relevant people or departments. These people can all access your live datasets and use the full interactivity functionality. Any time you add or edit any of your data and publish it, this will automatically be updated for everyone.

The next step once this is shared is to view a report on the usage of our data and dashboard:

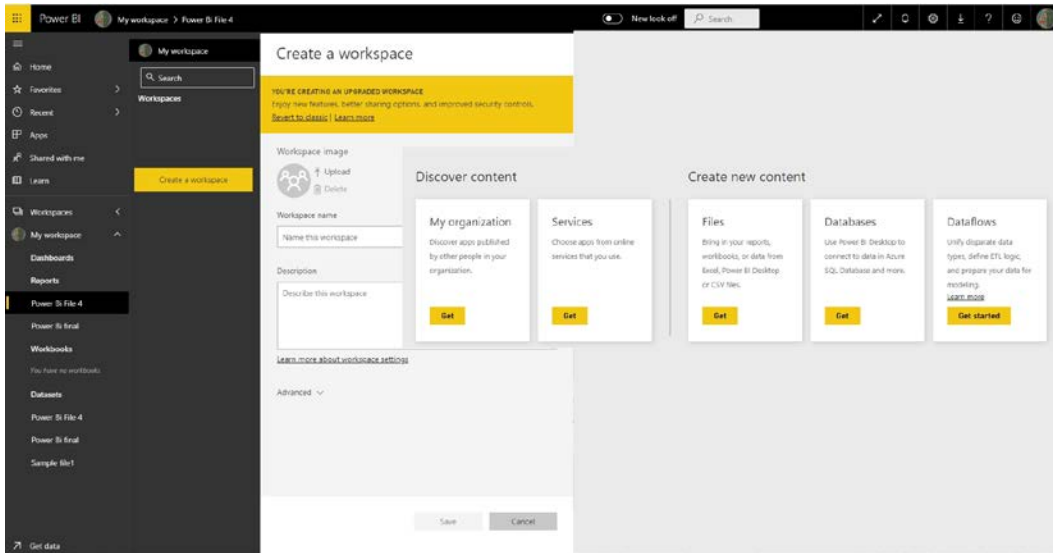


Figure 8.52 – Report usage metrics

When you click on the **Usage metrics** button, Power BI helps you understand who is using the dashboards and for what purpose in your organization. The **Usage metrics** reports are read-only, but it is possible for you to copy the report, which then enables you to edit the report. When looking at the usage metrics, it is possible to see whether one of the dashboards that you have created is not being used or you can see whether a specific department is using a dashboard all the time.

Most people can make a dashboard, but few people can make awesome dashboards that stand out. The next section will give you a few hints and tips on what to do and what to avoid.

Best practices

There is no concrete answer to what the best dashboard to have is. It depends on a number of different factors that will influence the type of dashboard that you create.

Audience

Who will be looking at the dashboard? Is it a member of the public who is looking at government statistics from a particular county or will it be used by a business data analyst who is looking for financial trends? This will establish how much information and detail you need to have on your dashboard. Another thing to consider is where the dashboard is going to be viewed. Will it be on a user's laptop or tablet or will it be on a large screen? This determines how much content you can have on your dashboard.

Clutter

One of the biggest mistakes people make is when they try to put too much data, visuals, and information on the dashboard. This is a twofold problem as firstly, the more visuals you have, the more this will adversely affect the performance. Secondly, there will be too many visuals and too much other information for the user to understand quickly. Try and remove all but the necessary information and use one screen, in fullscreen mode, for all the tiles on your dashboard.

Size and color

Most people read from the top to the bottom of a screen, so put the most important information in the top-left corner of the dashboard. You could use the **Card** visualization, which allows you to make a number or a total stand out with a different font, color, and size. If everything is the same size on your dashboard, it becomes difficult to know what is the most important information. Make the more important information slightly bigger, making it stand out. There are some useful color themes that you can apply to the entire report, but at the same time, you can customize the color of each data point in a graph to make them stand out more.

Although there are many more tricks you could apply, the best is to look at the few that I have mentioned and then play around and practice with creating different dashboards. As with anything, the more you practice, the better and easier it will become.

Summary

We started this chapter by following some steps. The first step was to retrieve data before we edited it created the applied steps. We then imported a table so that we could build relationships between the tables. We created calculated columns and measures before creating our interactive dashboard. After publishing the content, we then shared our dashboards and datasets in a variety of different ways. Although this might look like it is very complicated, the reality is that if you follow these steps, it is a little time-consuming but a relatively simple process.

In this chapter, we connected data, and although we connected data from a folder, we could have used the same process to connect data from the web, a portal, or a SQL server as well. We used transform tools to make sure that only specific file formats were included, as well as making sure that extensions would always be in lowercase as Power BI is case sensitive. We created a relationship between different tables so that we could use `Managers of Store` as well as the `Zip Codes` data, before creating calculated columns and measures. We created the measures that we need for our dashboard using mathematical equations, before finally creating the dashboard and uploading it.

Personally, I love dashboards as they are so intuitive. They take vast amounts of data and instantaneously create a visual depiction of what is happening. Not only can this data come from one place, but it can also come from a multitude of different places, including other live web pages, such as the stock exchange. By clicking on a query, the tiles showing the visualizations update immediately and you can see trends.

In the next chapter, we will begin an advanced topic and learn about the M language and how to work with it.

Section 3: Learning M

In this section, you will be introduced to the Power Query program language. We will look at the basic M syntax and learn how to write M with some examples of M usage, such as unpivot and pivot, and look at keywords, which load all the library functions offline in Excel and Power BI. At the end of this section, we will discuss the difference between M and DAX and look at how results are produced in M.

This section comprises the following chapters:

- *Chapter 9, Working with M*
- *Chapter 10, Examples of M Usage*
- *Chapter 11, Creating a Basic Custom Function*
- *Chapter 12, Differences Between DAX and M*

9

Working with M

In this chapter, we will be introduced to the Power Query M language and how to use and write the syntax, including the steps to reveal a list of functions and definitions.

This chapter will look at the structure and syntax of M. All programming languages have their own specific syntax and structure, and once you master the structure of M, it becomes easier to understand. We will look at the main data types and functions and provide a walk-through demonstration of how to use each of these data types. Some of the text data types that we will cover include numeric data types, lists, records, tables, searches, and shares, before looking at how to import a `.csv` file using M.

In this chapter, we will cover the following main topics:

- The beginnings of M
- Understanding the M syntax and learning how to write M
- Using `#shared` to return library functions, including function definitions, without having an internet connection or connecting to an outside data source

This chapter provides an introduction to the M language and how it is used. We will go through some basic M syntax for writing some programming lines. There will be a series of simple walkthroughs of different types of programming and we will look at how each of these types can be used.

By the end of this chapter, you will be able to look at M code and see whether there are syntax or structural problems and be able to fix them.

Technical requirements

You need an internet connection to download the relevant files from GitHub. Also, the code files for the chapter can be found at <https://github.com/PacktPublishing/Learn-Power-Query/>.

This chapter assumes that you already know how to open the Power Query editor and that you are comfortable with using the different commands in the editor. We also presume that you are familiar with at least one type of programming language, so you will understand the syntax and logic behind M.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=pQ6Iyh7mxZE&list=PLcLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=10&t=0s.

The beginnings of M

As we have learned so far, Power Query allows you to connect to data sources, as well as clean, transform, filter and then, finally, publish them. Beyond this, it also allows you to mash up or collate data from multiple sources. The language is called **M**, as **Mashup** is the jargon name that it started with.

Power Query has a really good interface that allows most people to use it effectively without doing too much programming. Everything that you create in Power Query is translated into M. Although we have dabbled a bit with M, we have allowed Power Query to do most of the work for us. In *Chapter 8, Creating Dashboards with Power Query*, we edited M and created our own code when we inserted additional columns. Without M, we would not be able to write more complex queries, transformations, and calculations. There are certain things, such as connecting to web services, that can only be done with custom M code.

The first thing I would like to say is that M is a functional language, and like every programming language, it has its own structure and syntax. Before looking at the basic syntax structure, I would like to remind you that M is case-sensitive for both variable and function names. This is very different from Excel, where you can type in lowercase or uppercase and Excel changes it to uppercase in the formula. VBA usually changes the syntax automatically. As an example, when you are using functions, Excel normally changes the first letter to uppercase. DAX, on the other hand, allows you to use either uppercase or lowercase, and you can also mix them up in your programming, although this is definitely not recommended.

Of course, M is found in more than just Power BI Desktop; it can be used in Power Query, Excel, and the **Get & Transform** import feature in older versions of Excel.

The next section deals with the M syntax and explains the differences between `let` and `in`, which are the basis of the M syntax.

Understanding the M syntax and learning how to write M

In this section, we are going to look at the syntax of M, explaining what it is, and then we will go through a few examples. As mentioned earlier, M is made up of individual language elements, including functions, variables, expressions, and values, which are all used to transform data.

Let's start by seeing what M looks like when we open it for the first time.

In Excel, open **Power Query Editor**, and then select **Advanced Editor** from the **View** tab:

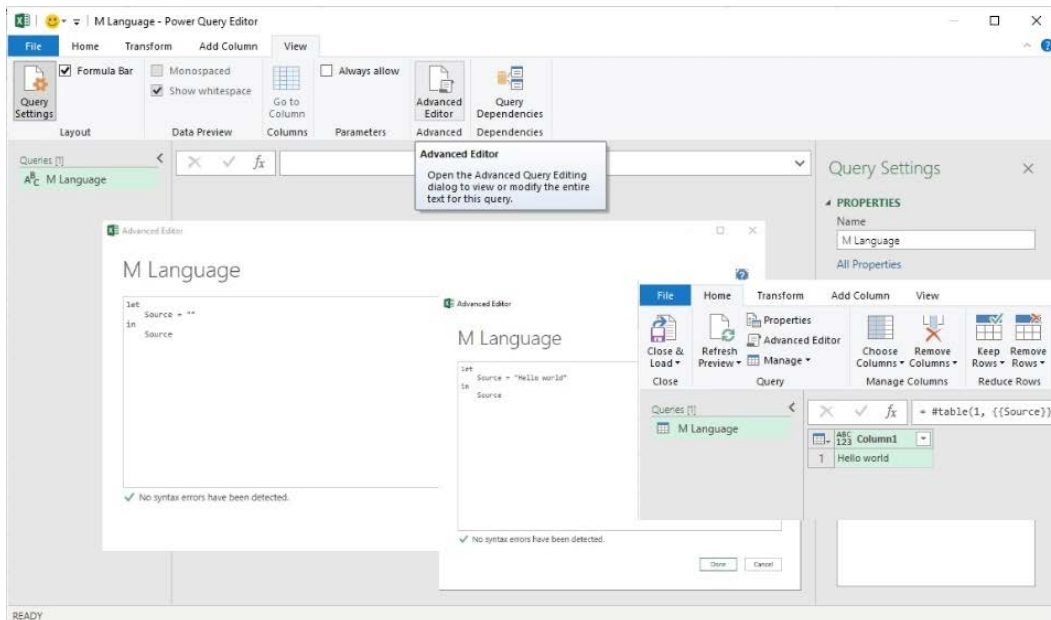


Figure 9.1 – The Advanced Editor window

By default, we see the following code snippet:

```
let
    Source = ''
in
    Source
```

There are two blocks—namely, the `let` expression block and the `in` expression block. Let's understand each one.

The `let` expression block contains the programming code—or, as we can call it, the procedural steps—that we use to define the query. Each one of these steps has a variable name that has been assigned an expression, and this expression is the logic for transforming the data. There can be multiple steps, and each step usually evolves using the step before it, although this is not always the case. The procedural steps do not have to follow the physical or logical order and can be mixed. This, of course, can be a complete nightmare for you or anyone else trying to work out what you are trying to do, so the accepted best practice is to keep them in the correct order. Thus, each query in M is a single Power Query `let` expression. This is the code that defines the datasets, and each procedural step is known as a variable.

The `in` expression block is an output. I know it sounds wrong, but it will return a variable value based on `let`.

Looking at *figure 9.1*, the two blocks come up automatically and all we need to do is populate them. I have entered `Hello World` into the `let` expression block, and when we run this, we get **Column 1** with **Hello World** in it. Although this is not very impressive, it does give us an indication of how the syntax works.

Important note

There are a few other basic things to note when writing in the `let` expression block. For example, you need a comma (,) at the end of every line, except for the line before `in`. Variable names are normally one word, using camel case or an underscore (_), but if you would like a long variable name with spaces, then you need to use # followed by the variable name in quotation marks (for instance, `#'This is the long variable name'`). The variable names can also include special characters, such as % or &. The variable name is also the name of the applied steps found on the right-hand side of the screen.

To use comments, you can use `//` at the beginning of the line (the same as in JavaScript). Parentheses (()) are used to pass parameters to a function. Square brackets ([]) are used to encapsulate a set of records. Set braces ({ }) are used for lists.

Most of the functions in M are part of an object class, and the syntax is as follows:

```
ObjectClass.Function() ,  
ObjectClass.Function(Parameter) ,  
ObjectClass.Function(Parameter1, parameter2)
```

There are a few functions that came out in the original release of Power Query that do not follow the same object class name, such as the date. To use the date, we write `#date()`, and it is then followed by three parameters—`year`, `month`, and `day`. This will be discussed in greater detail in *Chapter 10, Examples of M Usage*.

The basic M syntax is relatively straightforward, provided that we follow the correct structure with `let` and `in`. The one trick to remember is to look at the type of parentheses that are used to determine whether we are dealing with variables or lists, or whether we are trying to encapsulate a set of records.

In the next section, I will go through a number of different examples of M code. Some of the code will not need data sources as the values will be generated by the code itself. Of course, using the same techniques, we will be able to use the same functions on any data source.

Using #shared to return library functions

In this section, we will look at the `#shared` libraries, which loads functions and enumerators in a `result` set. This means that we do not need any datasets as the code that we create will automatically make use of the `#shared` libraries. We will concentrate on creating text data types, numeric data types, lists, records, tables, searches, and shares, as well as importing a CSV file. With each of the different types, we will write the code so that we can see how it works and be able to apply it to other programs. We need to associate data with data types so that M knows whether the data is text, a number, or a string, and so on. We will briefly look at each of the most common data types and see the similarities and differences between them.

Text data types

We already created one example of this at the beginning of this chapter, `Hello world`. You will notice that it was not necessary for us to explicitly assign data types as the variable object as data types are assigned automatically. Referring to *figure 7.1*, if you look to the left-hand side of **Hello world**, you will see that it has a text data type (**ABC**) in the output window on the right. Although there is one text data type, there are many different ways in which we can use it.

One of my favorites is the `Text.format()` function, which allows you to insert values into a piece of text. For example, take the following command:

```
Text.Format('[StudentName] has chosen #[Subject]',
[StudentName = 'Alice', Subject = 'Maths'])
```

From the preceding example, the sentence would become Alice has chosen Maths. This same function can be used to pass a list instead of a record, as in the following expression:

```
Text.Format('The first number is #{0}, the second number is  
#{1}, the third number is #{2}',  
{15,9,29})
```

This would return the following text:

```
The first number is 15, the second number is 9, the third  
number is 29.
```

The next data type is the number data type, which, as its name implies, deals with numbers and number formats.

Number data types

If we look at a simple number data type, we can see that when it is complete, it automatically assigns the number data type to it. Let's see how this works.

Create a new blank query and type the following:

```
let  
    variable1 = 5,  
    variable2 = 10,  
    variable3 = variable1 + variable2  
in  
    variable3
```

From our comma-delimited list of variable declarations, we get an answer of **15**. You will notice that I have three different variables—the first two variables have a number assigned to them, while the third variable refers to the value of the other two. If you look at the **APPLIED STEPS** section, the three variables have been turned into steps alongside the query. Refer to the following screenshot:

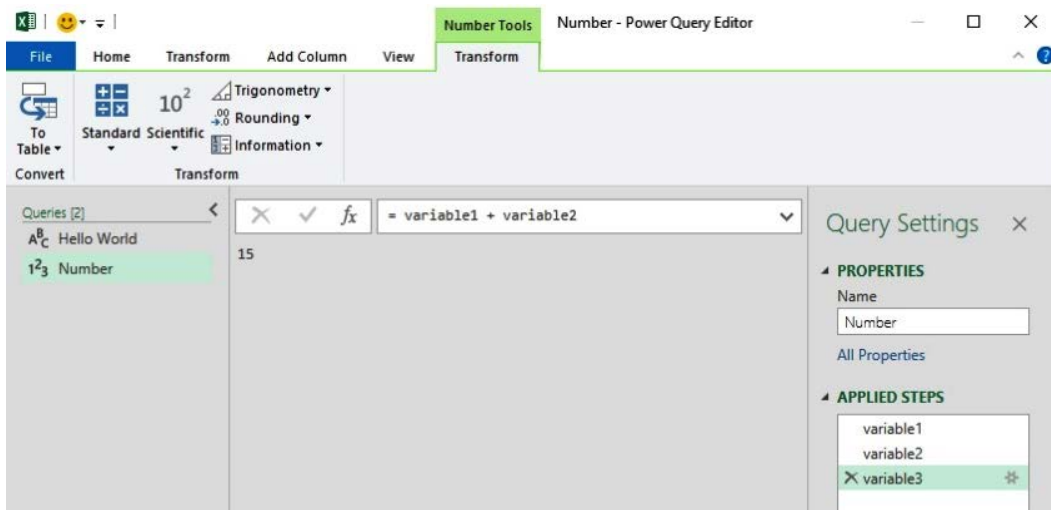


Figure 9.2 – Steps inside a query

If we wanted to change the variable name by creating a space, we can do so, but then we would have to have `#'variable 3'` for it to work. If you look at the preceding code, it looks like procedural code that works from the top to the bottom. The Power Query engine starts by looking at what we need for our `let` expression. In this case, we are looking for the value of `step3`. It will then look at what step 3 is, and then it will have to evaluate step 1 and step 2 as they are mentioned. If step 1 is not needed, the Power Query engine will not look at it, even though it is the first step.

One of the other things that can be really useful is that it is possible to refer to the output of other queries. We already have a query called `Number` that we have already created. If we wanted to call that query, we wouldn't have to have the `let` or `in` expression blocks; we can delete everything and type in `Number`, and the output of the `Number` query will be displayed.

The number data type is used very often as most of the time we work out totals, averages, and other mathematical formulas. The number data type can include integers, decimals, and currency.

We use lists in programming all the time, from numbers to objects and everything in between. The next section will look at how we can create lists, including nested lists.

Lists

Lists are ordered sets of values. To create a list in M, open the Power Query editor and type the following:

```
let
    fruit={'Apple','Grapes','Pear'}
in
    fruit
```

On the left-hand side, you will see that this is a list data type, but there is also a **List Tools** menu that appears. Refer to the following screenshot:

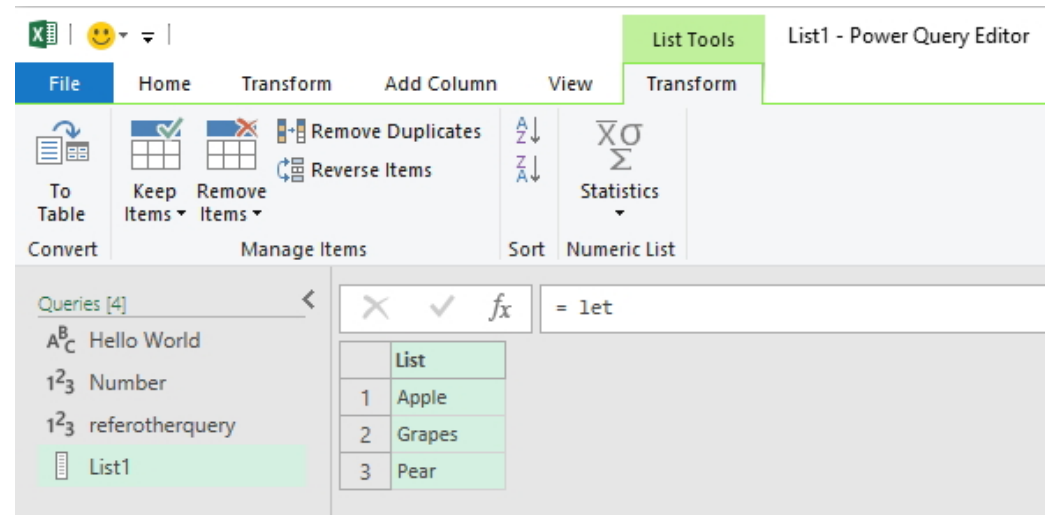


Figure 9.3 – The List Tools menu

With this menu, you can convert the list into a table, remove items, and do much more. As an output, the list we created is an ordered list; Apple will be first and Pear will be last.

It is possible to have nested lists that are separated by { }, { }. For example, type the following command:

```
let
    fruitNested ={{'Apple','Grapes'},{'Limes','Lemons'}}
in
    fruitNested
```

If you click on the first list, you will see that the items in this list are at the bottom.

It is possible to do things with numbered lists that would save you time—for example, you could type `numbersToTen = {1..10}`, which will give you all the numbers from 1 to 10.

We can now expand on lists to create records, which are lists of fields.

Records

A record is written as a comma-delimited list of fields with the values that are associated with the various fields. They are written with square parentheses (`[]`). Let's see how this works.

Create a new blank query and type the following into the **Advanced Editor** window:

```
let
    record = [firstName='Warren',surname='Sparrow',title='Mr'],
in
    record
```

From the preceding record, we can see that we have formed a comma-delimited list of fields. There are three different fields—`firstName`, `surname`, and `title`—and the field names and the values associated with them are recorded after each equals sign:

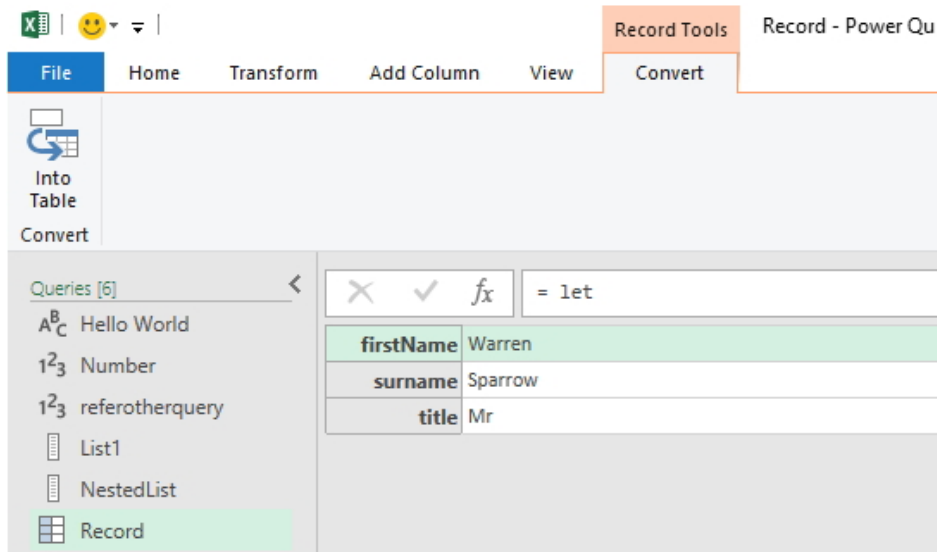


Figure 9.4 – The Record Tools menu

On the left-hand side in the query, you will notice that the icon for a record is next to this query, and there is also a **Record Tools** menu.

Once again, we can expand on the records and use them to create table data types.

Table data types

The table data type is probably the most important structured data type. You could think of it as being made up of a combination of lists and records.

Create a new blank query and type the following into the **Advanced Editor** window:

```
let
    Source = #table({'A', 'B', 'C', 'D'}, {{'1', '2', '3', '4'}, {'10', '11', '12', '13'}})
in
    Source
```

The `#table` function is a function that will create a table for us. There are two parts to this. The first list is the names of the column names in a text data type. In this example, the column names are A, B, C, and D. The second list, and each additional list, creates one item in each column for each row, as follows:

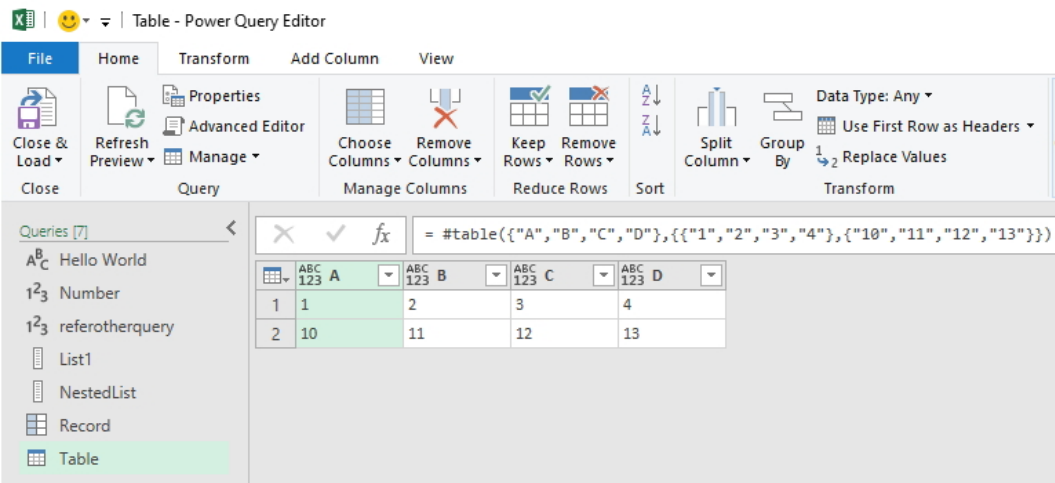


Figure 9.5 – The Table data type

There is now a **Table** data type in the query, so we know that this has been associated with the table data type. Notice that some of the data types in the column headings use the **ABC123** icon, which depicts any data type. We would need to associate a specific data type later to correct this.

There is a different way of creating a table, but it is slightly more complicated.

Try it out by typing the following:

```
Source = #table(type table[A=number, B=number, C=number,
D=number], {{1,2,3,4},{100,101,102,103}})
```

In the preceding code, we created a table, but this time, the code says that the data type is a number, so we will not have the problem of it showing the **ABC123** icon as this is a generic data type. If we wanted to change the data type to a number data type, we can change this to the number data type—**123**.

One of the things that we need to do once we have the data is search or find the specific data that we are looking for. The next section deals with how we can find relevant data.

Searching for relevant data

In the previous section, we learned that tables are made up of a combination of lists and records. So, this is important as we can find values from lists, records, and tables by using either the positional or lookup operator.

Let's see how.

Create another blank query and, in the **Advanced Editor** window, type the following:

```
let
    fruit={'Apple','Grapes','Pear'},
    numbersToTen = {1..10}
in
    numbersToTen{2}
```

In the preceding code, we have a variable called `numbersToTen`, which is all the integer numbers from 1 to 10. In the `in` expression block, we are asked for the number at index 2. When we run this code, we will get the value 3 because the index starts at 0, not 1. So, the value at index 0 will be 1, the value at index 1 will be 2, and our answer will be the value at index 2, which is 3.

If we changed the last line of the preceding code and we typed in `fruit{0}`, we would get the value of `Apple` as our answer. If we type in `fruit{6}`, we would get an error message as a result, as we do not have enough items in our list. However, if we typed in `fruit{6}?`, we would get an answer of `null`, as the question mark will say that we are out of range and returns `null`.

Using the `{ }` parentheses allows us to do a search for the *n*th item in a list, and we can use exactly the same process to search for a record.

We can use the same function to find lists and records from a table. Create a new blank query and, in the **Advanced Editor** window, type the following:

```
let
    Source = #table({'A','B','C','D'},{{'1','2','3','4'},{'10','11','12','13'}}),
    Output = Source[A]
in
    Output
```

The source table has fields with the A, B, C, and D headings, and it then has numbers in the rows underneath. The preceding code is similar to what we have used before, with `Output = Source[A]` as the only difference. From the square bracket `[]` operator, we get an individual field from a record, but the source, in this case, is a table, so this will give us all the items in column A as a list.

If we change `[A]` to `{A}`, this will change the search to the first row and give us a result of 1, 2, 3, 4.

We can also put this together and type `Source [A] {A}`, which will give us the value in the first row and first column.

Of course, there are so many different functions that are already built into M, and we have only touched on the most common ones. If you create a new blank query and type `= #shared`, this will give you a list of all the different functions that are available for use. It will display a record and each field is a name of a function that you can use. You will notice that these have **functions** as their data type. When you click on one of the functions, it will give you a preview, as well as the documentation for that function.

Up to this point, we have only used code to create the data that we needed. The next section deals with how to import a CSV file using M.

Importing a CSV file using M

One of the things that we do on a fairly regular basis is open CSV or Excel files. Although we can do this within Power Query, if we wanted to do this with M, we could do so in two different ways. The first way is to use the **Advanced Editor** window in the Power Query editor, and the second way is to type M into the formula bar in the Power Query editor.

When we open **Advanced Editor**, we can type the following:

```
let
    salesData = Csv.Document (File.Contents ('C:\DataFiles\sales.csv'),
        [Delimiter=',', Encoding=1252])
in
    salesData
```

If, however, you wanted to type this in directly without using the editor, then type `type = Csv.Document (File.Contents ('C:\DataFiles\sales.csv'), [Delimiter=',', Encoding=1252])` into the formula bar, which will do the same thing.

This will then import the CSV file, but there are two problems with this. The first is that there is a blank column, and the second is that we need to promote the first row as headers.

To remove the column, open **Advanced Editor** and add the following to the next line:

```
RemoveCols = Table.RemoveColumns(salesData , 'Column3')
```

You can view this in the following screenshot:

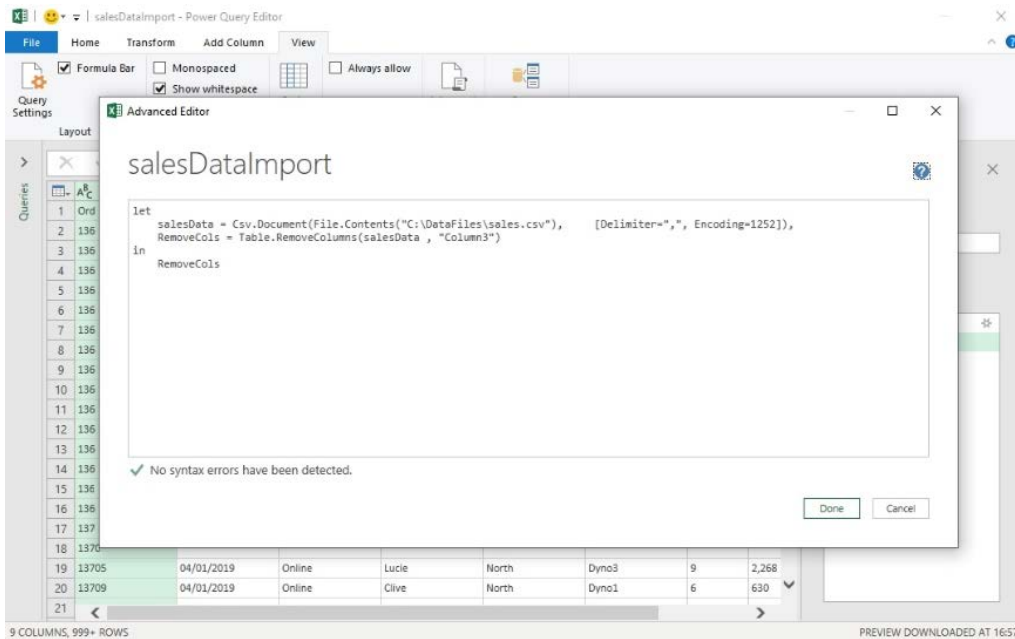


Figure 9.6 – Removing columns

Once we have removed the columns, we can then move on to the second step, which is promoting the headers:

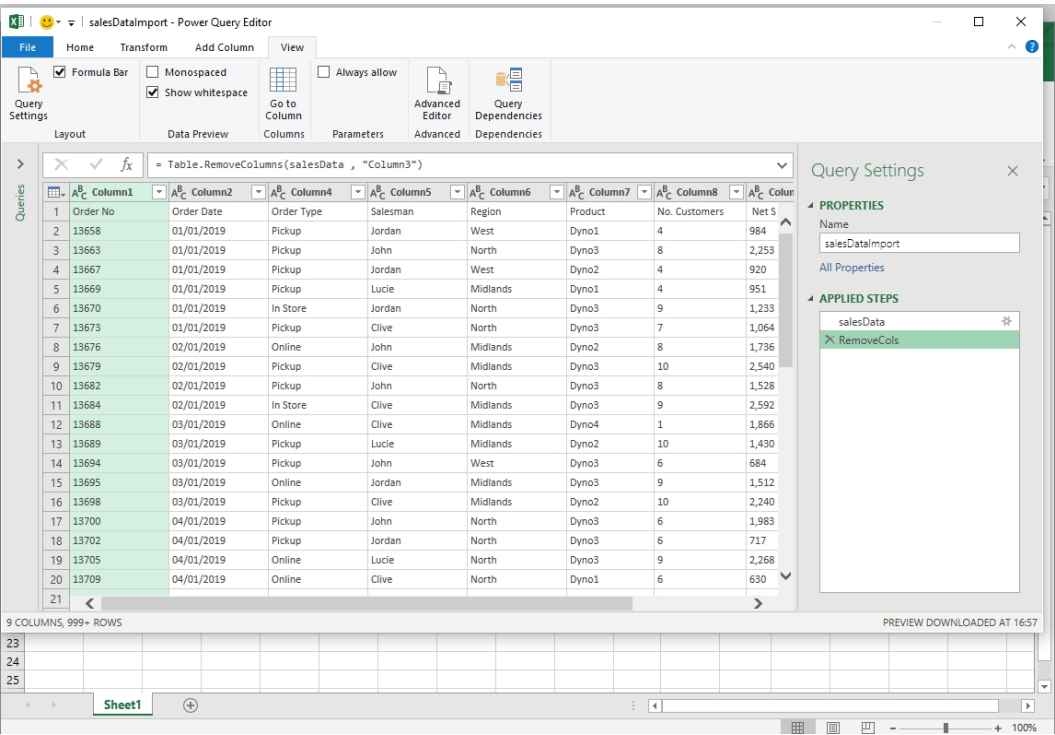


Figure 9.7 – Promoting the headers

Lastly, to promote the headers, type `PromoteNames = Table.PromoteHeaders(RemoveCols, [PromoteAllScalars=true])`.

This section has helped us go through the basics of how to use the M syntax with examples on how to use the most common data types and functions, including text, numbers, lists, tables, and records.

Summary

This chapter provided us with an introduction to how M is used. You have now acquired the necessary knowledge to understand how the syntax is used, and you should be able to look at some code and work out how it works. It makes life a great deal easier when you can look at code and determine what it is doing, especially if there is a problem that you need to solve.

In this chapter, we covered how M came about and how it works, and how it is a functional language that has its own structure and syntax. We made a quick comparison between M, DAX, and VBA with regard to structure and syntax, before looking at how to write M code. Although we have mostly used the **Advanced Editor** window in the Power Query editor, using the same code, we could type it into the formula bar; however, it is better to use the editor, as once your code goes over multiple lines, it is not always easy to spot a mistake in the formula bar.

This chapter included many working examples, going through text data types, numeric data types, lists, records, tables, searches, shares, and importing a CSV file. I find these particularly helpful, especially as each section is an entity in itself, which allows you to use a specific piece of code in other queries that you have already created. Having gone through the main data types, it should be evident just by looking at the data type in the query what it is associated with and which type of parentheses you should be using. The last section covered importing a CSV file, and although we have done this previously in this book, I felt that it was necessary for us to be able to see the code and the steps and understand what the code is doing.

The next chapter will deal with the aid of M with more in-depth scenarios and formulas.

10

Examples of M Usage

This chapter concentrates on a few examples of M usage, including the concatenate function. We will first compare the difference between formulas in Excel and Power BI, before looking at the **ampersand operator** (&) and how it can be used. We will go through an example of how you can do this by using a simple name and surname concatenation formula.

We will examine how `Text.From` and `Text.Combine` can be used to join and concatenate different strings, dates, and columns. We will also learn how to set up our own SQL server legally and for free to use for non-commercial purposes; it will have full functionality. In doing this, we will also cover how to import the `AdventureWorks` databases into SQL to use them as a resource.

Lastly, this chapter concentrates on **parameters** and how they can be used effectively to filter data sources, adding parameters to control statements that allow us to filter them according to different dates. We will continue by adding parameters to order objects and columns in ascending and descending order, before looking at how we can make these changes in Power BI's **Data** view. Some of the skills that we will cover include defining various parameters, as well as creating and renaming them.

In this chapter, we will cover the following main topics:

- Merging using the concatenate formula
- Data type conversions
- Setting up a SQL server
- Using parameters

Technical requirements

You need an internet connection to download the relevant files from GitHub. Also, the code files for the chapter can be found at <https://github.com/PacktPublishing/Learn-Power-Query/>.

This chapter assumes that you understand the M syntax and structure and know how to code text data types, numeric data types, lists, records, tables, searches, and shares. You should also know how to import a CSV file. If you are unsure on how to do this, please refer to *Chapter 9, Working with M*.

You will also need to have a SQL server. If you do not have one, I will show you how to download and install a free version of SQL Server legally. You will also need the AdventureWorks database, which can be found at <https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=mzRQLLQCmvg&list=PLcLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=11&t=1s.

Merging using the concatenate formula

I am always amazed by how many different ways data is exported from a CSV file depending on where it comes from. With student and project management software, how the data was typed in sometimes also makes a difference. A typical way in which a person's name and surname is displayed is SMITH, John. In Excel, it is possible to take this field using the **Text to Columns** comma-delimited method to split the name and surname into two columns, as shown:

	A	B	C	D
1			Together	Proper
2	SMITH	John	John SMITH	John Smith
3				
4			=CONCATENATE(B2," ",A2)	=PROPER(\$C2)
5				
6				
7				

Figure 10.1 – Excel concatenation

You can then concatenate the two cells using the appropriate formula, and you can use the **Proper** function to get the correct case. Of course, this is time-consuming and there is no quick way of doing the first step automatically. The other problem is that if you are doing this with multiple different documents, it is a complete nightmare.

This first section will compare what we have just done in Excel with getting the same result in Power BI. You will soon see how much easier it is to use Power BI and the advantage of this is that once you have created this step, you can use it with different data sources, and Power BI Desktop will do this automatically for you.

So, this is what we do:

1. Launch Power BI Desktop and select **Excel** from the **Get Data** tab.
2. Select **names.xls** and click **Open**.
3. Select the **SurnameNames** tab, and then click on **Transform Data**.

Depending on the version of Power BI Desktop that you have, the **Transform Data** tab may not be in the same place as in the following screenshot. In the latest version of Power BI Desktop (June 2020), you need to select **Split Column** from the **Transform** tab. If you have an older version of Power BI, you will need to select **Data source settings** from the **Transform data** group under the **Home** tab:

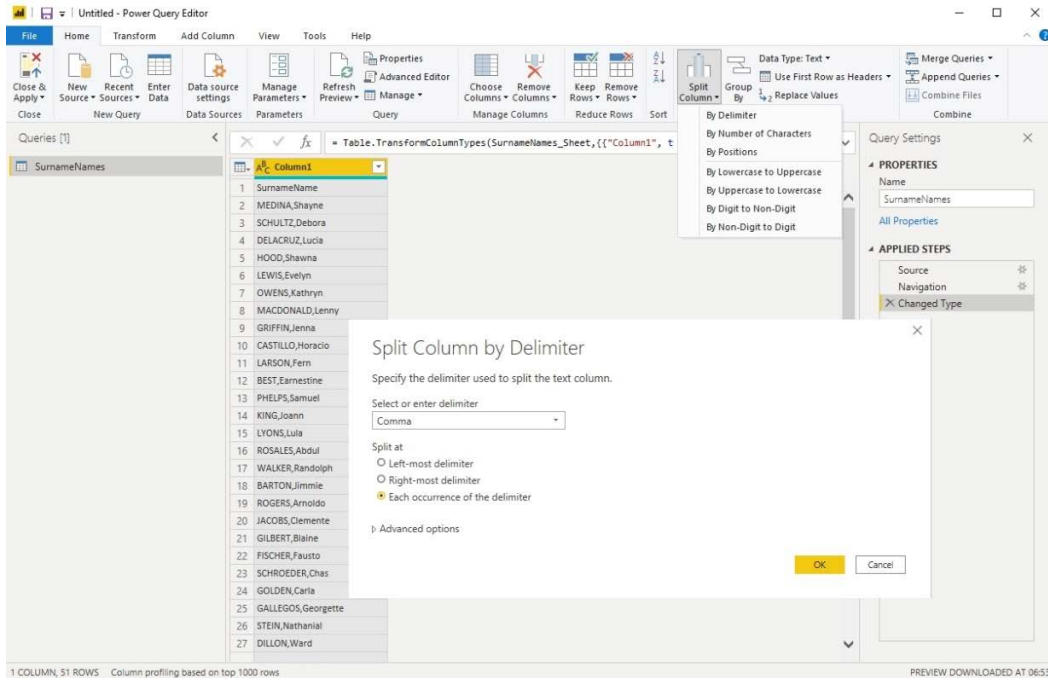


Figure 10.2 – The Split Column menu

Our delimiter is a comma, but if it were something different, this is where you would change it. There are a few extra options when selecting the advanced options, such as splitting it into columns or rows, but for now, we will keep the default settings and click **OK**.

We have now split our original column into two columns, but the surnames are in uppercase and we want them to be in title case. There are two different ways in which we can make this change

- The first way is to write some M code: `= Table.TransformColumnTypes("#Changed Type1",{{"Column1.1", Text.Proper, type text}})`. Here, you will notice that we have asked to change the case of the first column, `Column1.1`, to `Proper` or title case. The one good thing with this is that it uses the same code as Excel—`proper`.
- The second way is to right-click on the **Column 1.1** heading and select **Capitalize Each Word** from the **Transform** menu, as shown:

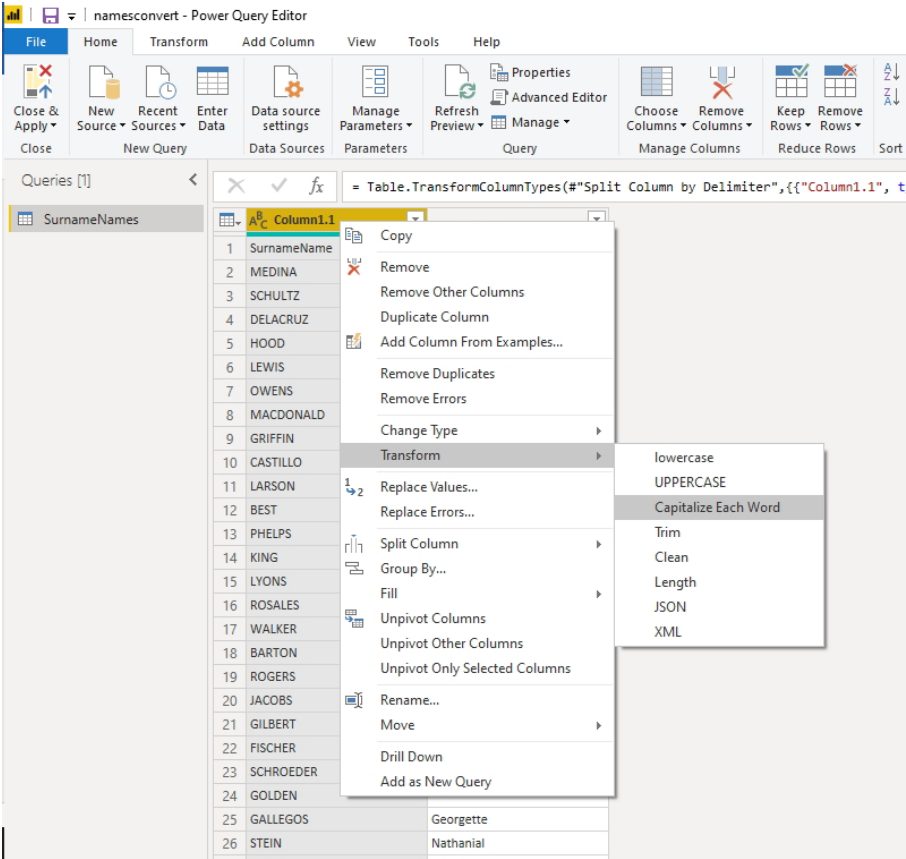
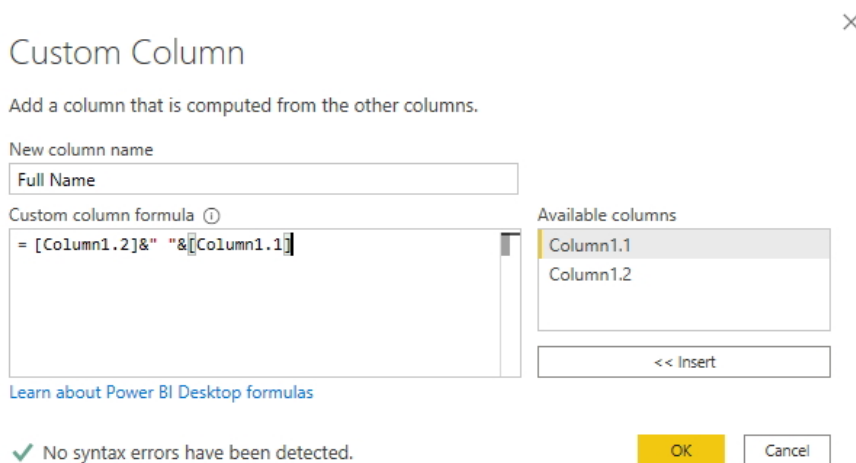


Figure 10.3 – Capitalize Each Word

The last step is to concatenate the two fields that we have. In order to do this, we will create a new custom column and use a formula that has a `[Name] & " " & [Surname]` syntax. Here, we are selecting our two fields—Name and Surname—and we are combining them. The " " characters create a space between the Name and Surname fields to prevent this from becoming one string. We have to remember that we want Name first and then Surname, so when we write the code, we need to refer to `Column2` and then `Column1`.

Click on **Custom Column** from the **Add Column** tab and type `[Column1.2] & " " & [Column1.1]`. As you are typing, the relevant field names will appear, and you can press the *Tab* key to accept them and then continue typing:



Custom Column

Add a column that is computed from the other columns.

New column name

Full Name

Custom column formula ⓘ

= [Column1.2] & " " & [Column1.1]

Available columns

Column1.1

Column1.2

<< Insert

[Learn about Power BI Desktop formulas](#)

✓ No syntax errors have been detected.

OK Cancel

Figure 10.4 – The available columns

An alternative to this method is to merge the two columns. Select **Add Column Menu** and click on **Merge Columns**. Choose **Custom** from **Separator** and set **-** as the separator. Click **OK** when done:

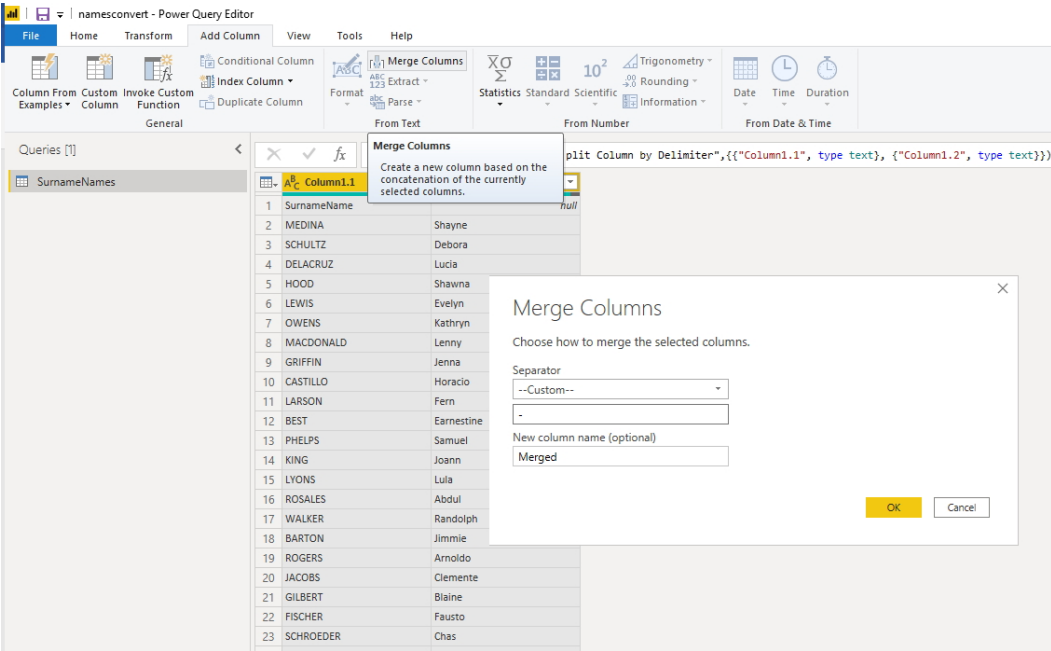


Figure 10.5 – The merge method

In the following screenshot, you will see that I have promoted the names and deleted the first row to make it look pretty, but the reality is that you would want to delete the first two columns and only have the `Full Name` column that you created:

The screenshot shows the Power Query Editor interface. The main table has three columns: Surname, Name, and Full Name. The data is as follows:

	Surname	Name	Full Name
1	Medina	Shayne	Shayne Medina
2	Schultz	Debora	Debora Schultz
3	Delacruz	Lucia	Lucia Delacruz
4	Hood	Shawna	Shawna Hood
5	Lewis	Evelyn	Evelyn Lewis
6	Owens	Kathryn	Kathryn Owens
7	Macdonald	Lenny	Lenny Macdonald
8	Griffin	Jenna	Jenna Griffin
9	Castillo	Horacio	Horacio Castillo
10	Larson	Fern	Fern Larson
11	Best	Earnestine	Earnestine Best
12	Phelps	Samuel	Samuel Phelps
13	King	Joann	Joann King
14	Lyons	Lula	Lula Lyons
15	Rosales	Abdul	Abdul Rosales
16	Walker	Randolph	Randolph Walker
17	Barton	Jimmie	Jimmie Barton
18	Rogers	Arnoldo	Arnoldo Rogers
19	Jacobs	Clemente	Clemente Jacobs
20	Gilbert	Blaine	Blaine Gilbert

The 'Applied Steps' pane on the right lists the following steps:

- Source
- Navigation
- Changed Type
- Split Column by Delimiter
- Capitalized Each Word
- Added Custom
- Renamed Columns
- Removed Top Rows

Figure 10.6 – The completed concatenation

The completed file has taken less than 4 minutes to change from start to finish, and that includes if you typed in the formula yourself without using the shortcut to convert the column to title case. I personally find that this saves me so much time as I have so many different class lists that need to be converted from a CSV file that I had extracted from somewhere into a proper name and surname, which are then used in other documentation. The beauty of this as well is that it does not matter whether the usernames are in lower or uppercase as you can transform them with one of the previous steps. I personally like to create another step and create a title case version of the first names as well so that they always look the same.

There are times where you might want to do the reverse of this, where you have a name and surname and you want to concatenate them into one string so that they can be used as the login name, as a username for software, or something similar.

I am going to use the same file, but this time I will use the **Names** tab for my data source. You can follow all the same steps, but when you create the custom column, the formula will be slightly different. Instead of using " ", you would simply type `= [Column2] & [Column1] & [Column3]`.

In the following screenshot, you will notice that I have a column that gives me the year that each person started working at the company, which I use in their username:

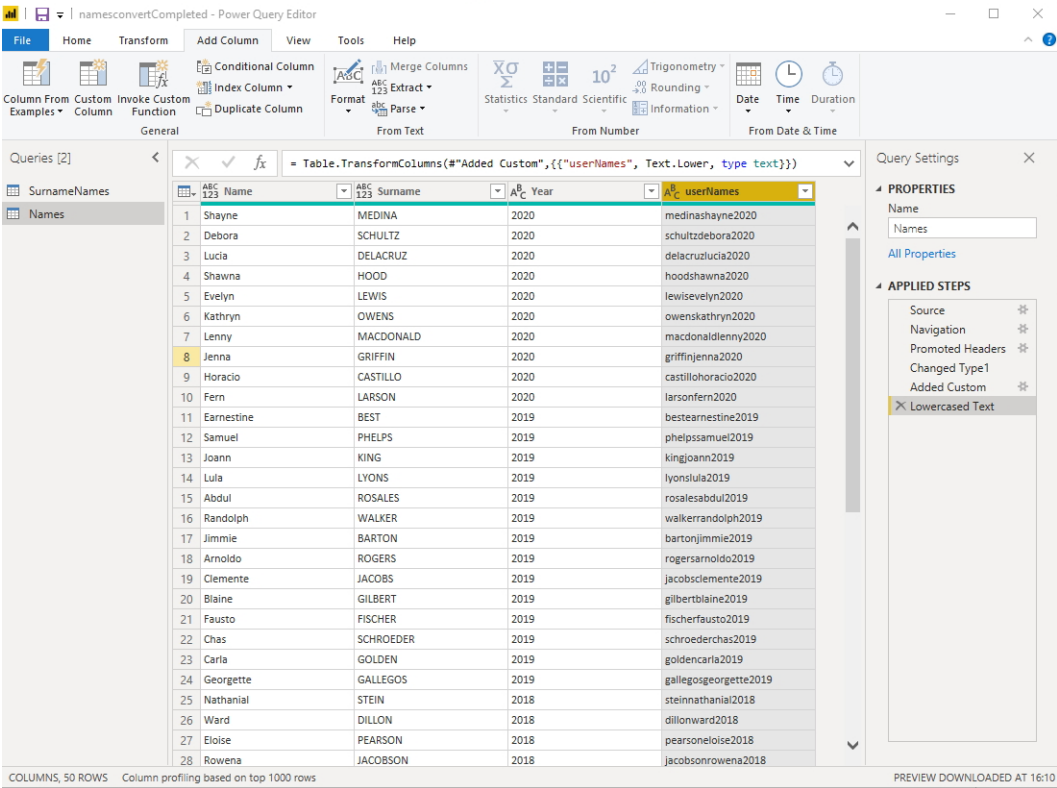


Figure 10.7 – Creating usernames

If you use this formula, you get an error as Power BI is trying to concatenate text and number data types together. This is possible to do in Power BI and will be discussed in the next section of this chapter. As I wanted to have text data types for their username, a quick way around this is to convert the number into a text data type, which then allows me to concatenate them all together without any problems.

In this section, you have seen the benefits of concatenating different fields in order to use the transformed data for other applications.

The next section deals with more complex data type conversions, which will build on your knowledge from *Chapter 9, Working with M*.

Data type conversions

In the previous section, we tried to concatenate a text and number data type together, which produced an error message. The same thing happens when we try to add a date or a few other things as well. In this section, we will look at the different ways in which we can join various data types.

To join two different data types, we can use the `Text.From` function. This function allows us to input a number, date, time, or even a binary value and it will return the numerical representation of that value. For example, if we type `Text.From(7)`, it will give an answer of 7. This, however, is not the number 7—it is text that represents the number 7. One thing to remember is that if the value is null, then `Text.From` will also return null. We can use the `Text.From` function to convert not only numbers, but also dates and times.

There are a number of different ways in which we can concatenate data using the **ampersand operator** (&). In Excel, the easiest way to concatenate is to use the & symbol. If we want to concatenate the **Title**, **Name**, **Street**, **City**, **Zip Code**, and **Country** columns from the following screenshot, we could type `=C2&" "&A2&CHAR(10)&D2&" "&E2&CHAR(10)&F2&" "&G2`:

The screenshot shows an Excel spreadsheet with a table of addresses. The formula bar at the top displays the formula `=C2&" "&A2&CHAR(10)&D2&" "&E2&CHAR(10)&F2&" "&G2`, which is used to concatenate the Title, Name, Street, City, Zip Code, and Country columns. The table data is as follows:

	A	B	C	D	E	F	G	H
1	Name	Gender	Title	Street	City	Zip Code	Country	
2	Faith C Schofield	female	Mrs	19 St Denys Road	POYSTON CROSS	SA62 3NX	USA	Mrs Faith C Schofield 19 St Denys Road POYSTON CROSS SA62 3NX USA
3	Liam E Pollard	male	Mr	123 Bootham Cresce	RIPE	BN8 4FX	USA	Mr Liam E Pollard 123 Bootham Crescent RIPE BN8 4FX USA
4	Benjamin M Bentley	male	Mr	38 Bootham Cresce	RICKLING GREEN	CB11 5HF	USA	Mr Benjamin M Bentley 38 Bootham Crescent RICKLING GREEN CB11 5HF USA
5	Bradley J Hayes	male	Mr	27 Conssett Rd	HILGAY	PE38 5JJ	USA	Mr Bradley J Hayes 27 Conssett Rd HILGAY PE38 5JJ USA
								Mr Ewan M Nash

Figure 10.8 – Concatenating using the & symbol

I specifically have not put the columns in the correct order so that I can demonstrate that it does not matter in which order you concatenate the data. I have also used `CHAR(10)`, which creates a line break so that it looks more structured than one long string of text.

We can also do this in Power Query and Power BI as it involves data transformation:

1. Open the `Address.xlsx` document and then add the table to the Power Query editor (Select this from **Table/Range** in the **Data** tab). We will use the `Text.Combine` function to convert all the fields that are not text fields into text types. This includes the fields that have a generic data type and shows both the text and number data types.
2. To add a step, we can either right-click on the applied steps and then select **Insert Step After** or we can click on the **fx** icon.

Paste the following formula into the formula bar:

```
= Table.AddColumn(#"Changed Type", "Address Labels", each  
Text.Combine(Record.ToList(_), "#(lf)"))
```

This formula creates a new column called `Address Labels` and concatenates the data from the other fields from left to right. `#(lf)` is the Power Query line break character that is the same as `CHAR(10)` in Excel.

When we concatenate strings, we can either use the `CONCATENATE` function or we can use the `%` operator. They are very similar, but if we are using the `CONCATENATE` function, then there is a 255 string limit. The reality is that 255 characters are not going to be used, and many people prefer to use the `CONCATENATE` function as it is slightly easier to read. However, I personally prefer using the `%` operator. At the end of the day, use the method that you prefer.

We will next look at using basic operators, but you will need a SQL server. If you already have a SQL server, then you can skip the next section and load the database into your SQL server. If you do not have a SQL server, then the next section covers how to set up a legal and free SQL server on your computer.

Setting up a SQL server

In this section, we will need a few more things to run the server effectively. I realize that many individuals do not have their own SQL server, but you can download **SQL Server Developer edition** for free. The major difference between SQL Server Developer edition and the other editions is that although the Developer edition has almost all of the same features as the Enterprise edition, it is not for commercial use. If you want to see the SQL datasheets that compare the various versions, you can download them from <https://www.microsoft.com/en-au/sql-server/sql-server-2017-editions>.

You will also need a Microsoft or an MSDN subscription, which is free, to download the relevant files. I am currently running the 2017 version, but you can download the SQL 2016, 2017, or 2019 Server Developer editions. There are a few differences among them, but overall, they are very similar. Please note that depending on the version that you download, the file size will, on average, be between 2.9–4 GB.

There are a few different ways in which you can download the Developer edition. If you have a slightly older computer, you might want to download **SQL Server 2016 Developer edition**, which can do everything except advanced **Transact-SQL (T-SQL)** querying, which we will do later in this chapter. Since advanced T-SQL querying isn't available on this version, I would not recommend using this edition unless you have an old computer.

There are two different ways to download and install the software.

The first way is to download SQL Server Developer edition from <https://my.visualstudio.com/Downloads?q=SQL%20Server>, shown as follows:

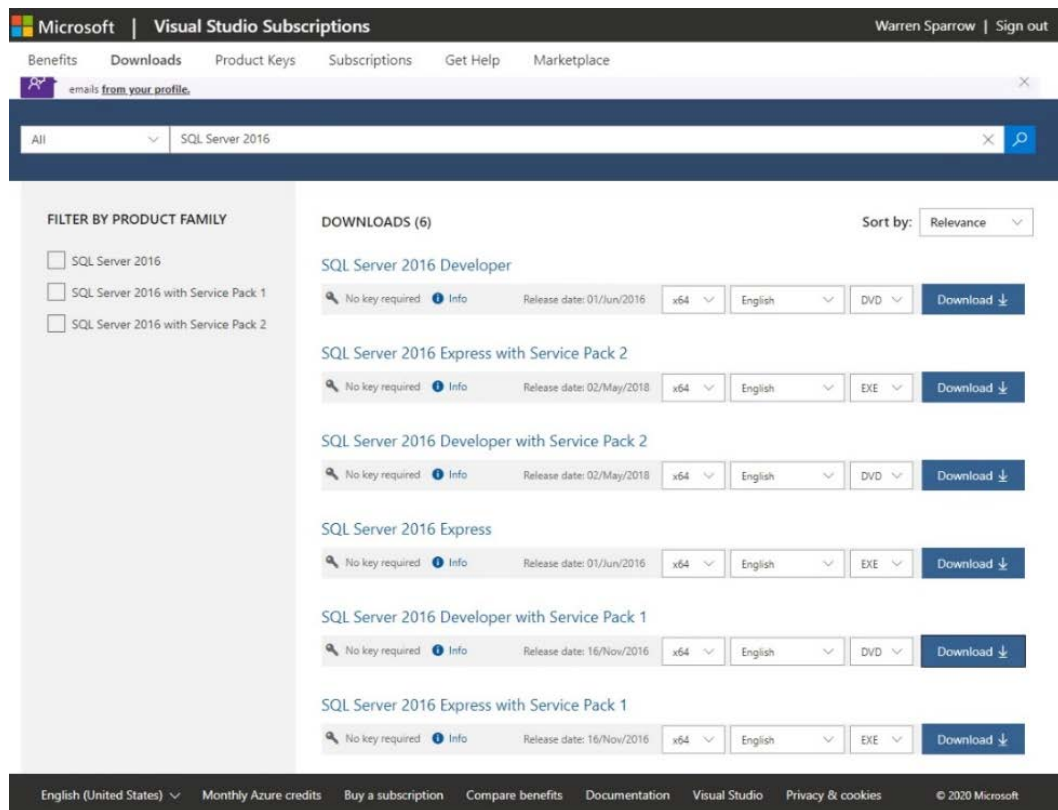


Figure 10.9 – The various SQL Server options

At this point, you can decide which version you would like to download. They can all be installed in very similar ways.

Once the file has been downloaded, there are a number of steps you will need to follow. Some steps can be skipped, but others have to be completed correctly. Let's go through them:

1. The first step is the **Planning** steps (which shows you the steps you will be following. Apart from reading this, there is nothing more to do). You can skip this step and move on to the **Installation** step if you wish:

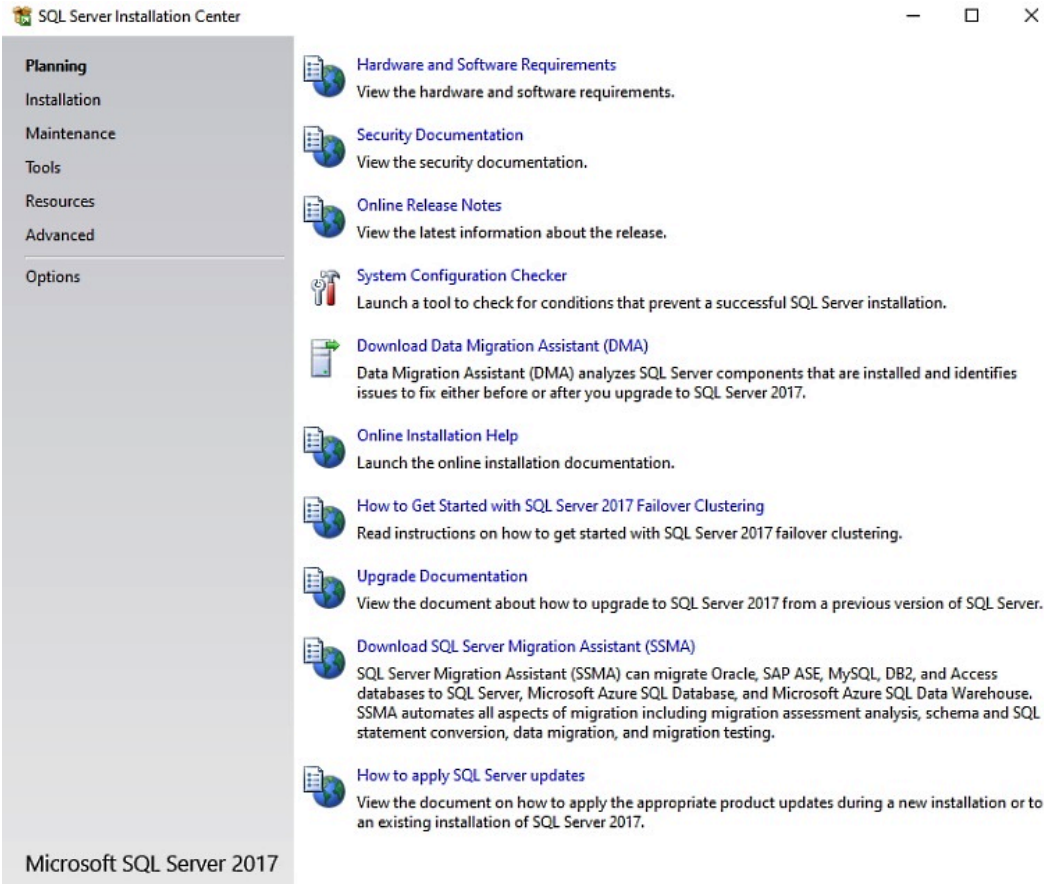


Figure 10.10 – The Planning steps

2. The **Installation** step is where you choose what you would like to install. Select **New SQL Server stand-alone installation or add features to an existing installation** and then wait for the window to pop up:

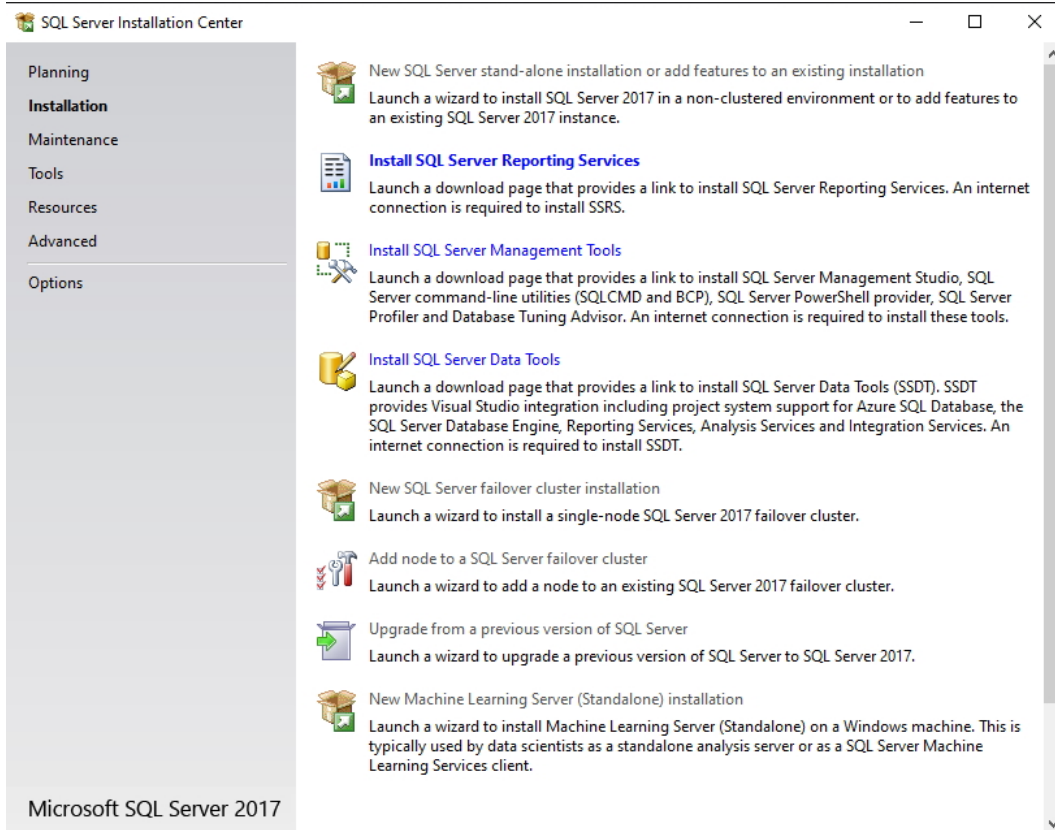


Figure 10.11 – The installation step

It sometimes pops up behind the current window, so you might not always see it straightaway. The easiest way is to move the current window to the left or right so that it is possible to see when another window opens.

3. You will see the **Product Key** window; you need to choose **Developer** from the **Specify a free edition** drop-down list box.
4. Once selected, click on **Next** and accept the **I accept the license terms** option before clicking on **Next** again.
5. The next window is the **Install Rules** window, which verifies the **Active Template** library, the registry keys, and whether the computer is a domain controller. Do not worry if a warning from your firewall appears.

6. The next step is **Feature Selection**, and this is where you can choose additional features to install:

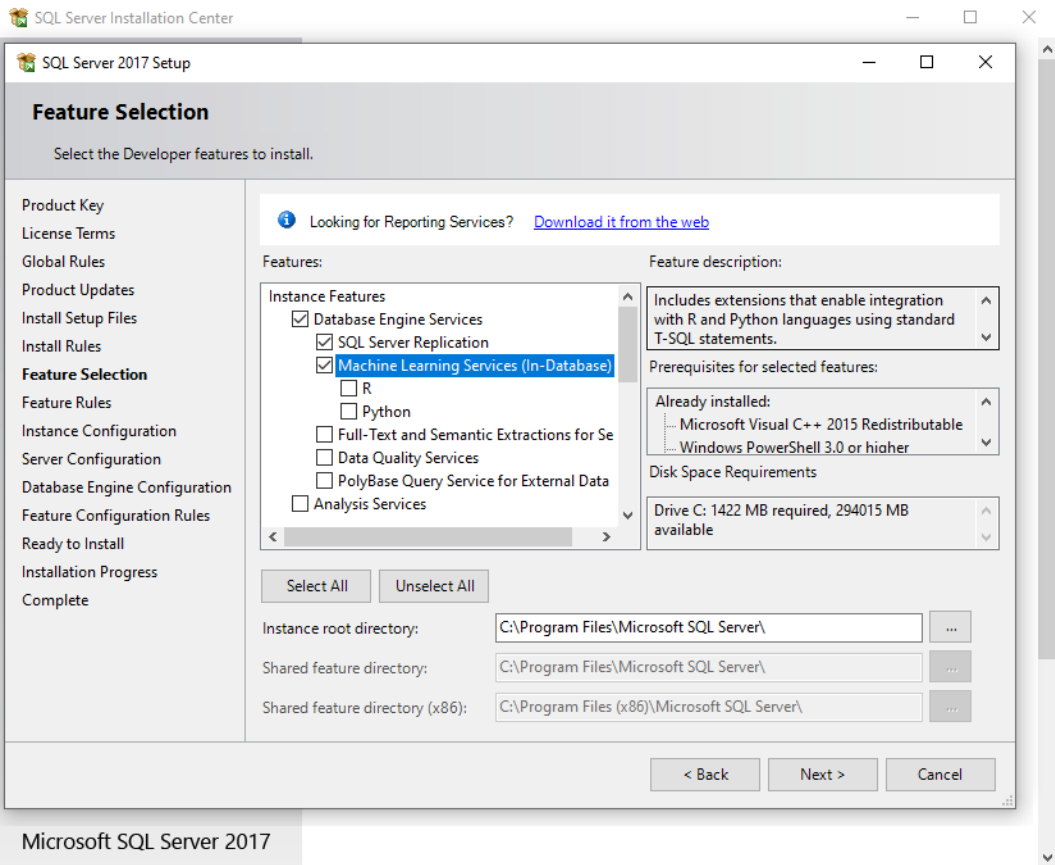


Figure 10.12 – The additional features

Select **Database Engine Services**; the other features are not needed for this exercise, although this is the time to install any additional features if you would like to continue using them after finishing this chapter.

7. The next step is **Instance Configuration**, and this is where we create and name our instances. Your screen might look different to this and not have any instances already installed:

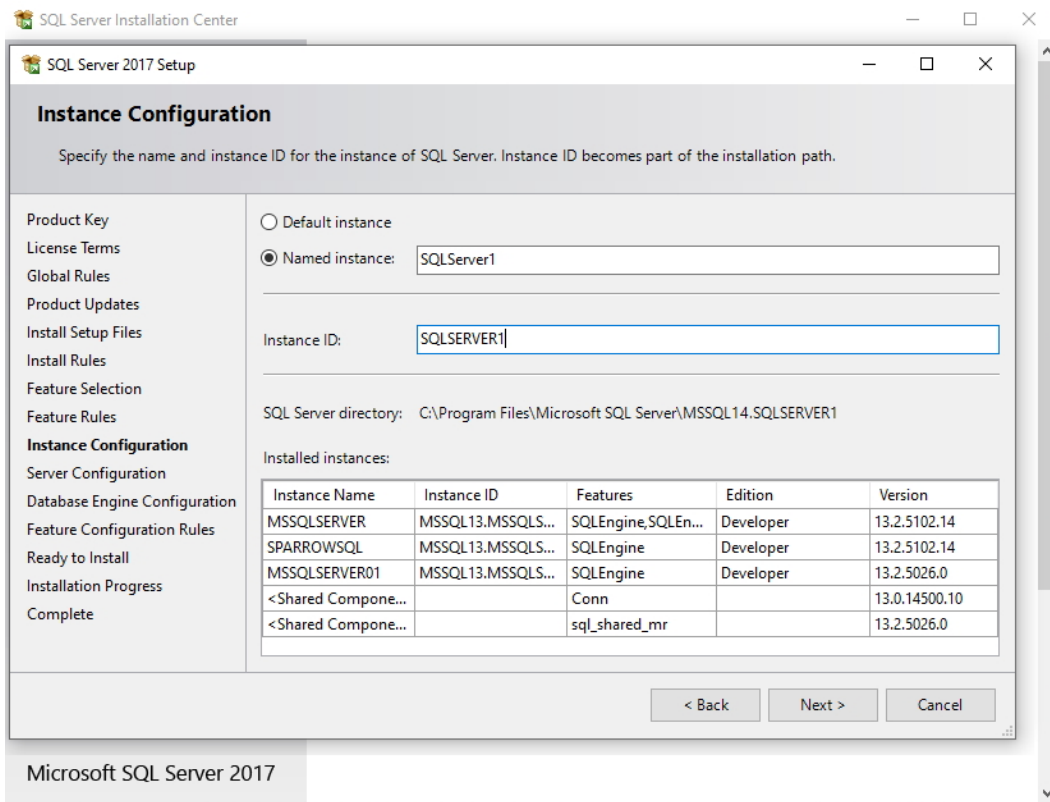


Figure 10.13 – Instance Configuration

If you do not have any other instances, it is fine to use the default one—**MSSQLSERVER**. At this point, I would like to mention that while you only need one instance, this exercise looks at using parameters from two different instances, so it might be useful to create a second instance.

In **Database Engine Configuration**, you have the option of selecting which method you would like to use to authenticate. Personally, I like to use **Mixed Mode**, which allows both Server and Windows Authentication. Click **Next** until everything is finished and installed.

We will now look at the second way to download the software, which is from Microsoft Visual Studio Dev Essentials:

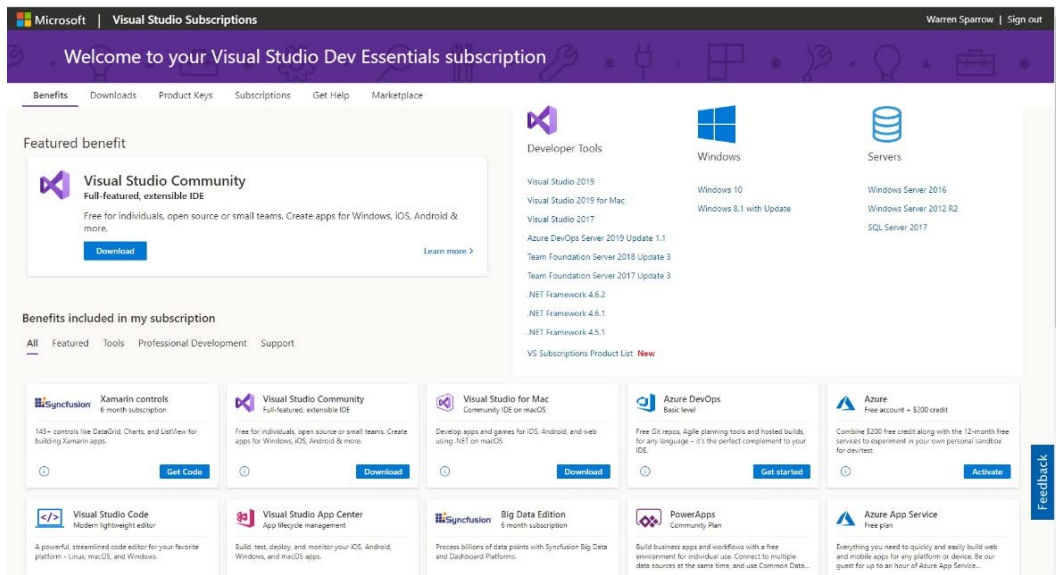


Figure 10.14 – Visual Studio Dev Essentials

You will have to agree to the terms and conditions and sign in with your Microsoft or MSDN account. You can download the **Visual Studio Community** tool if you like, but it is not essential. The site has changed; the SQL Server editions used to be on the first page, but they have now moved to under the **Downloads** section.

After selecting the developer version that you would like, select and download the file. This will download an ISO file, which you will either need to extract or burn onto a DVD. I find extracting the ISO file easier and less time-consuming than burning it to a disk. Once it is extracted, you will need to run the setup file. You then follow exactly the same instructions as in the previous steps. It takes a few moments to complete the installation, but you then have a SQL server that you can use.

Installing SQL Server Management Studio

The next step is to connect to the SQL server using Microsoft **SQL Server Management Studio (SSMS)**. Although there are a number of ways to connect and update databases, I find this one of the easiest. You can either click on **Install SQL Server Management Tools**, shown in *figure 10.11*, or you can open an internet browser and go to <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>. Download the Microsoft SQL Server Management Studio setup file. It is roughly around 550 MB in size, so depending on your internet speed, it should not take too long. It does take longer to install the software, so you might have to wait approximately 10 minutes:

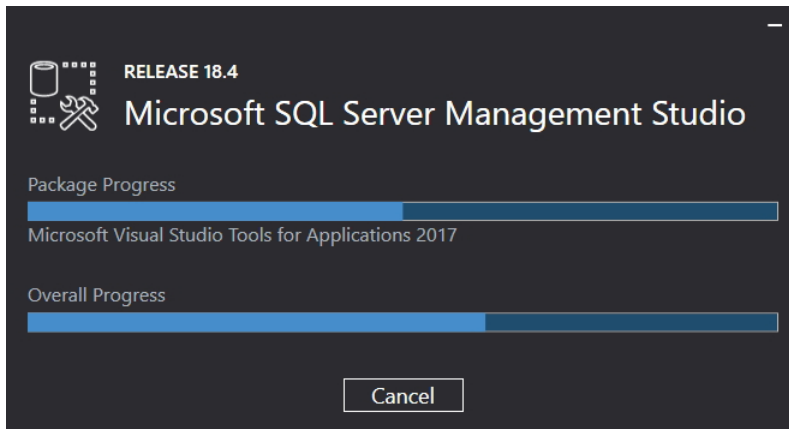


Figure 10.15 – The SSMS installation

Once the software has finished installing, you will need to restart your computer.

Open SSMS and **Connect** to the server by typing in the server name of the database that you created in the previous step:

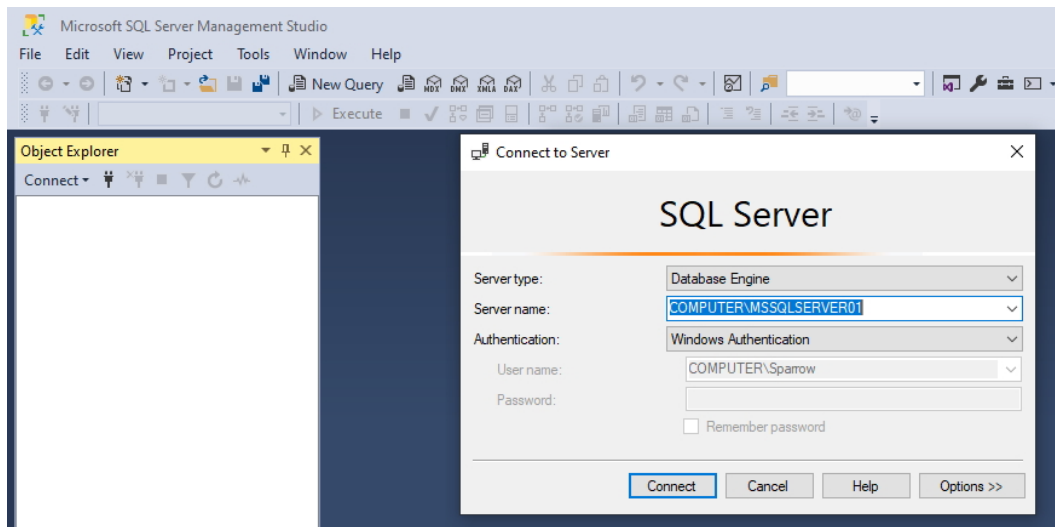


Figure 10.16 – Connecting to the SQL server

If you have successfully connected, you will see the **Object Explorer** panel. We need to update the database that we will use for this exercise. For this example, we will be using the Microsoft AdventureWorks database, which can be found at <https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15>.

Once again, you have the choice of downloading different versions, but I am using the four versions prior to and including the 2017 DW version. For this exercise, it might be worthwhile downloading more than one of the databases so that we can change the parameters to read another database, although it is only necessary to download one and then do the same steps to create the other parameters.

Download the files, which we will now use to restore the database in SSMS:

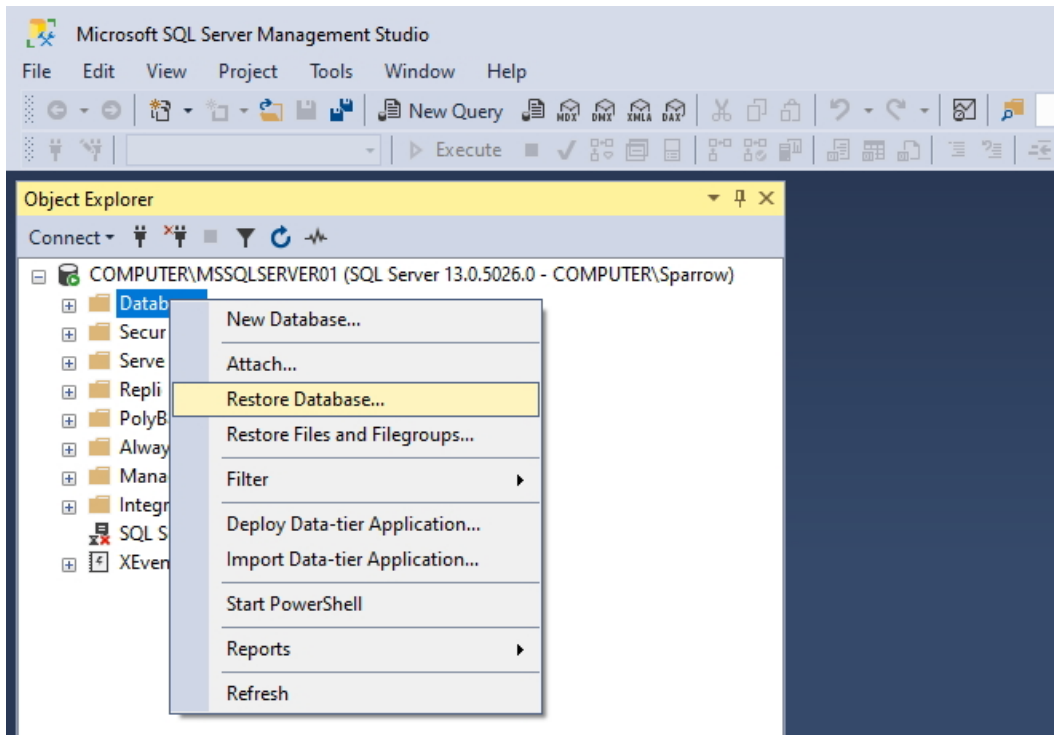


Figure 10.17 – Restoring the AdventureWorks database

Right-click on **Databases** and then select **Restore Database...** Select **Device** in the **Source** section and click on the three ellipses (...). Locate the file that you have saved and then click **OK**. Refer to the following screenshot:

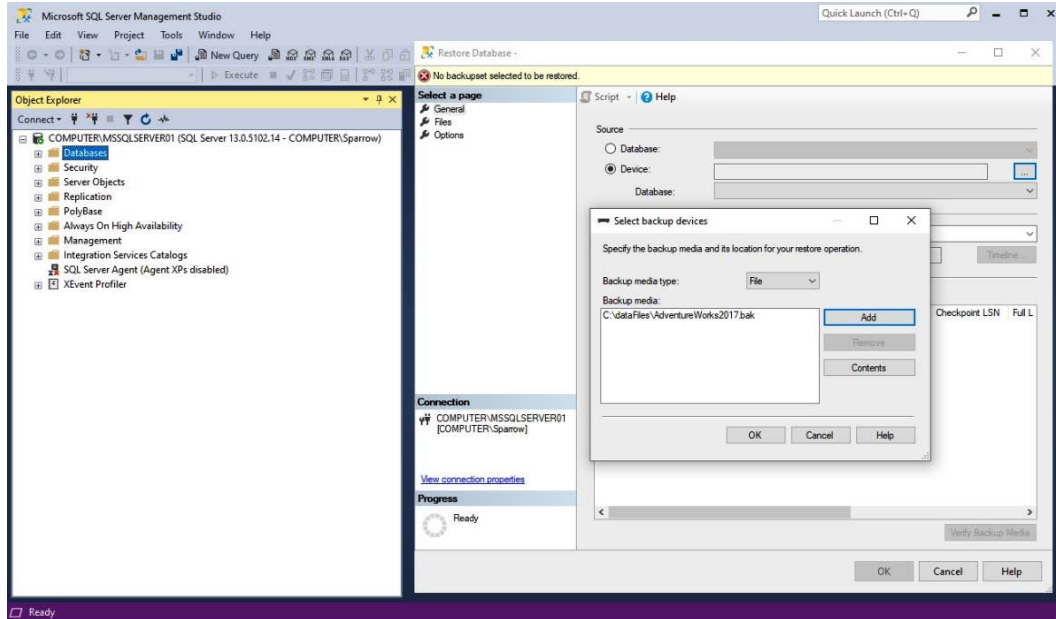


Figure 10.18 – Adding the AdventureWorks database

This will restore the entire database. Depending on which version you downloaded, the name of the database will now be visible on the left-hand side:

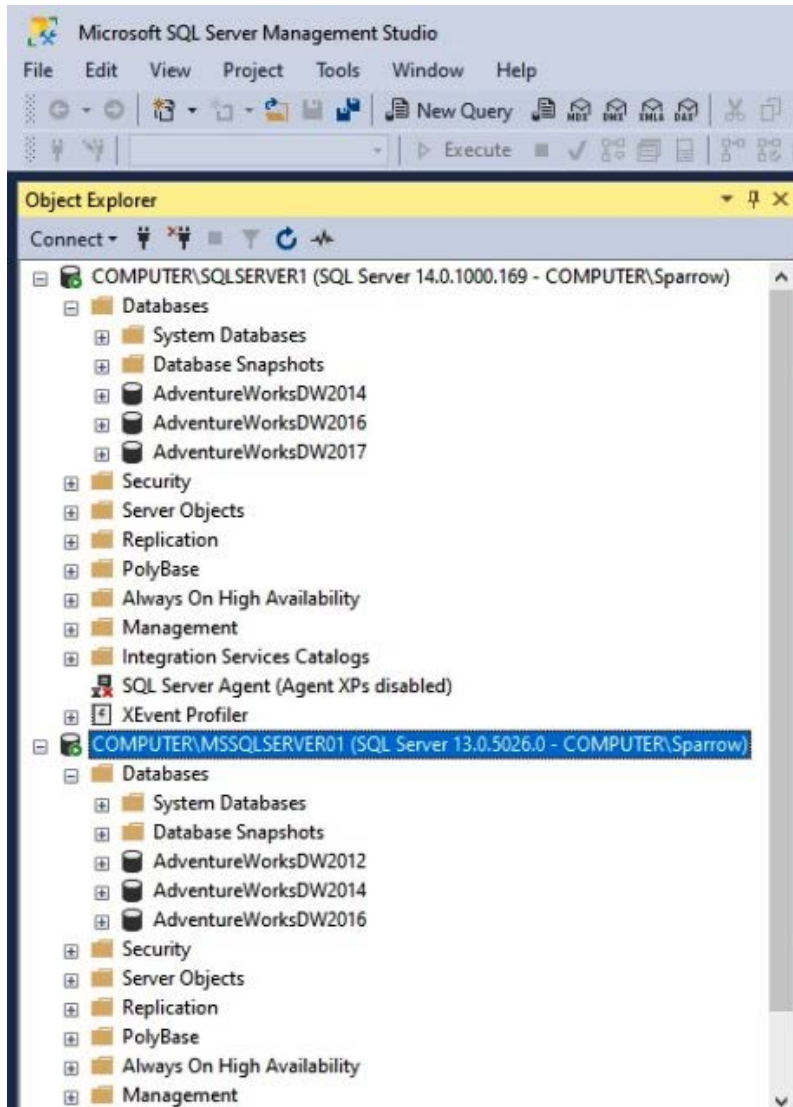


Figure 10.19 – The AdventureWorks DW databases

You will notice from the preceding screenshot that I have used the same steps to restore the other versions of AdventureWorks. My one bit of advice is that if you want another user to be able to access the database, then right-click on **Users** in the **Security** folder to add another user. All our instances and databases are now set up. The next step is to connect to them via Power BI Desktop.

To connect to the SQL server from Power BI Desktop, select **SQL Server** from **Get Data**. The trick is to remember what you called your server name.

Refer to *figure 10.20* to remember what you called your server. In Power BI, type in `./servername`, and then click **OK**:

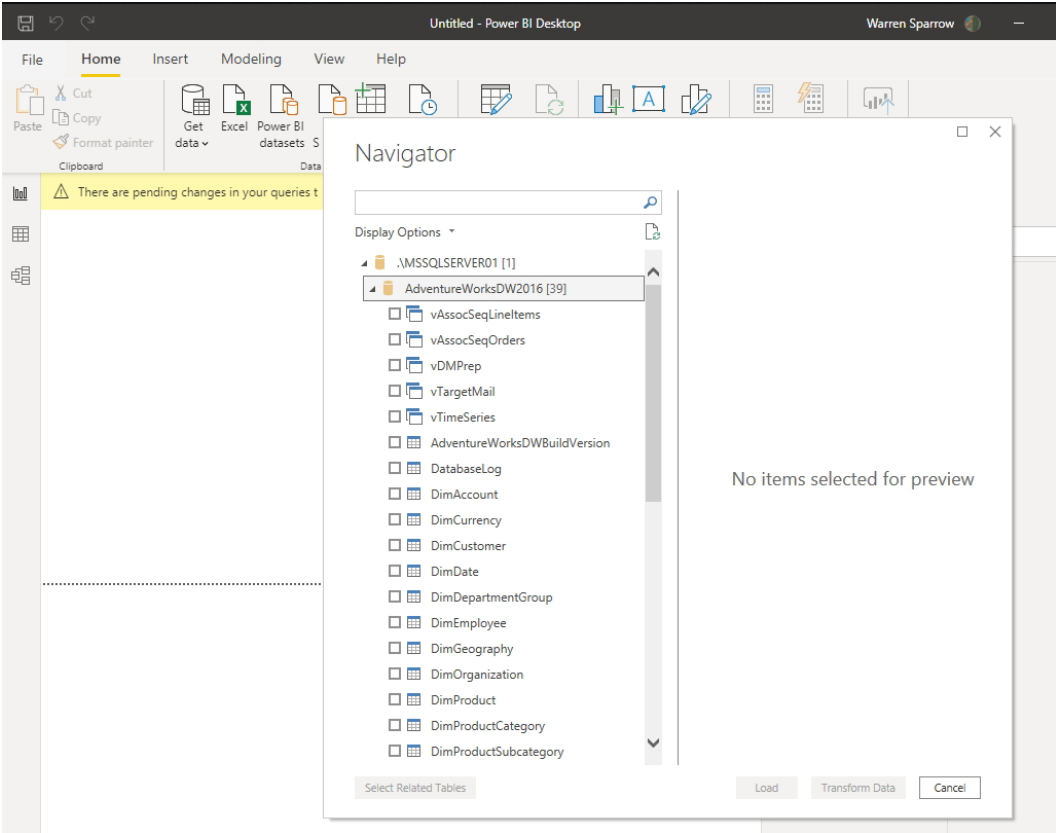


Figure 10.20 – Connected SQL

You will notice that in the preceding screenshot, I created two instances so that I can demonstrate switching between servers and databases for this exercise; but once again, if you only have the one instance, this will still work.

Now that we have the AdventureWorks database and a SQL server with instances, we can now connect to them and look at how we can use the different query parameters.

Using parameters

In this section, we will look at the Power BI Desktop query parameters, which provide a kind of way in which we can filter data. There were some developments and updates made to Power BI Desktop in 2016 that improved the ability to create parameters and use them in various ways. The most common ways that we can reference parameters are through data sources, filter rows, keep rows, and remove and replace rows. It is also possible to load the parameters into the data model so that we can reference them from measures, calculated columns, tables, and reports.

Parameterizing a data source

In this section, we will look at how we can connect different data sources that have been defined in query parameters to load different columns or connections to data sources. One of the great things is that certain pieces of software, such as Salesforce objects, SharePoint, and Power BI Desktop, allow you to use parameters when defining your connection properties. This means that you can have one parameter for the SQL server instance and another parameter for the target database.

Parameters are independent of datasets, which means that you can create the parameters before or after you have created or added your dataset. When we created a parameter in *Chapter 4, Connecting to Various Data Sources Using Get & Transform*, you will remember that we had to define the parameter and we set the initial values in the Power Query editor. After creating the parameters, they are listed in the **Queries** pane, which is where we can update and configure the parameter settings later.

Pretend that you have different customers who are using the same database structure, but they might be using different instances of SQL Server and they would typically have different database names. Using query parameters, we can switch between the different data sources and then publish reports to the different Power BI services. Let's see how to do this:

1. Open the Power Query editor and click on **Manage Parameters** from the **Home** ribbon.

2. In the **Name** textbox, type `SqLsrvInstance`, and then type a description of your choice into the **Description** textbox. The next bit of the setup might be a little bit different, depending on how many instances you have, but you will have to make sure that one of the instances that you are using is a real SQL Server name. You will need to add a `. \` character before your SQL Server name:

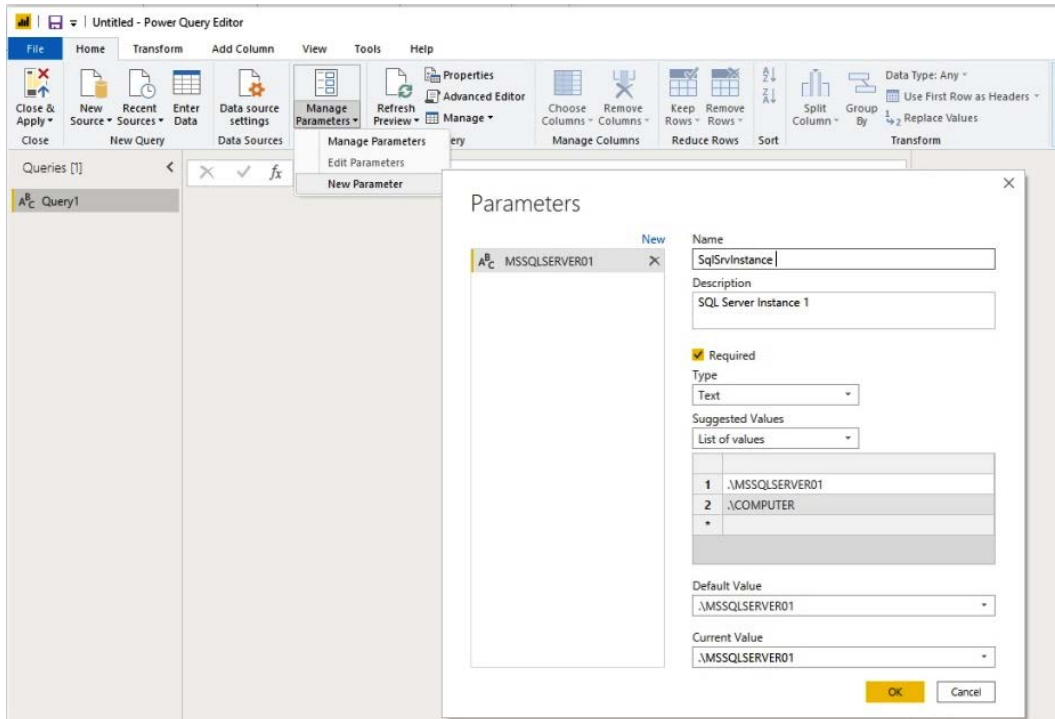


Figure 10.21 – Setting the parameters

3. For **Type**, select **Text**, and then select **List of values** from the **List of values** drop-down list box. A grid will appear, and every time you press *Enter*, it will create another line for you to type in.
4. Type in the SQL Server instance and make this the default value and current value.
5. Once you click on **OK**, this will close the **Parameters** dialog box. Add this parameter to the **Queries** pane. You will see the current value in parentheses:

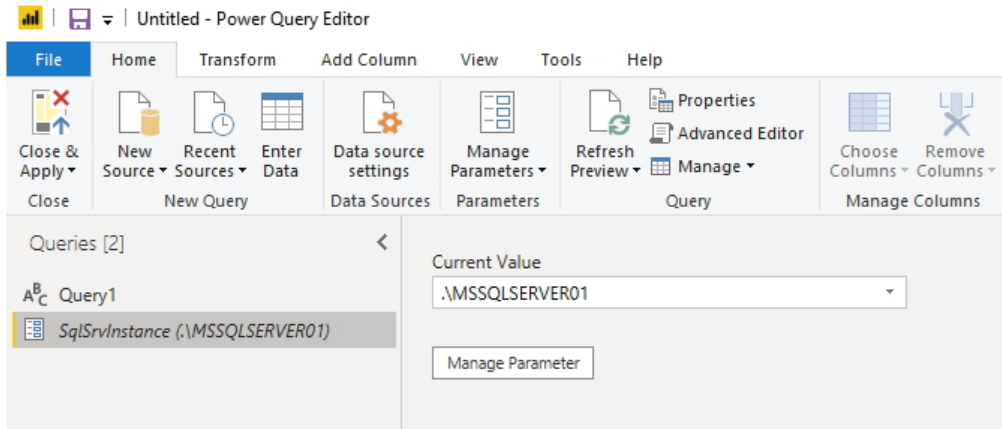


Figure 10.22 – Instance in the Queries pane

We are going to do exactly the same thing, but this time we are going to connect to the different databases that we have. This will then create the parameters that we need for the database that we require. Use the information from the following screenshot to create the **Database** parameter:

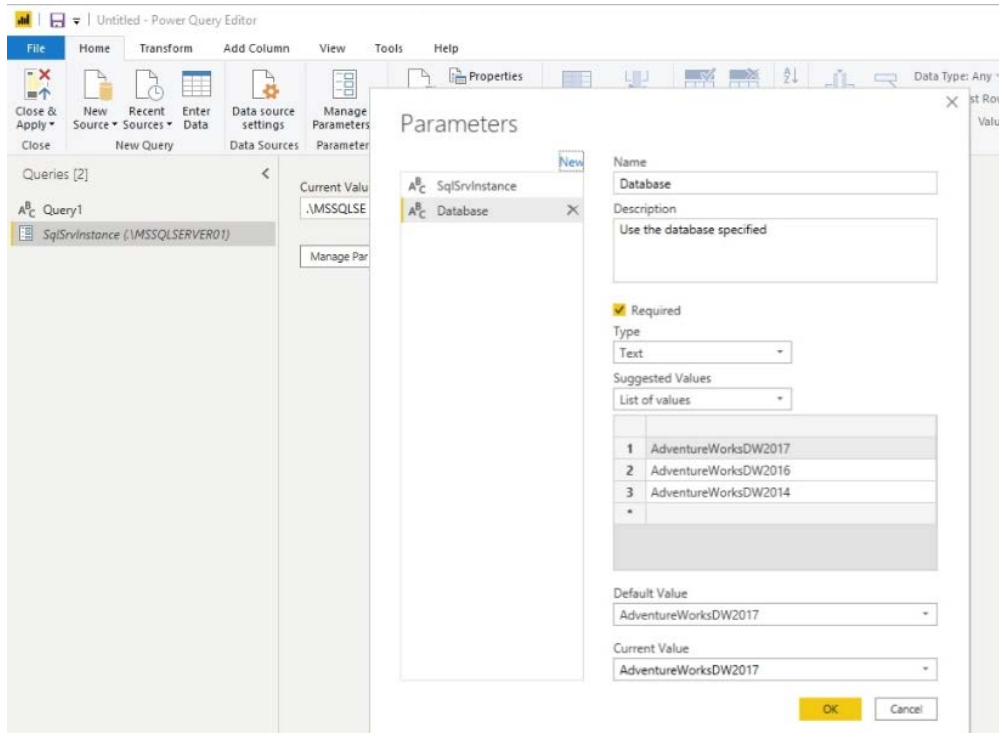


Figure 10.23 – The AdventureWorks database's Database parameter

6. Click on **OK**, and then on **Close & Apply**. We have now created the connection parameters and we can connect to the SQL Server instance to retrieve the database that we need.
7. We will use T-SQL to run this query. We will need to check that the **Require user approval for new native database queries** property is disabled for this to work:

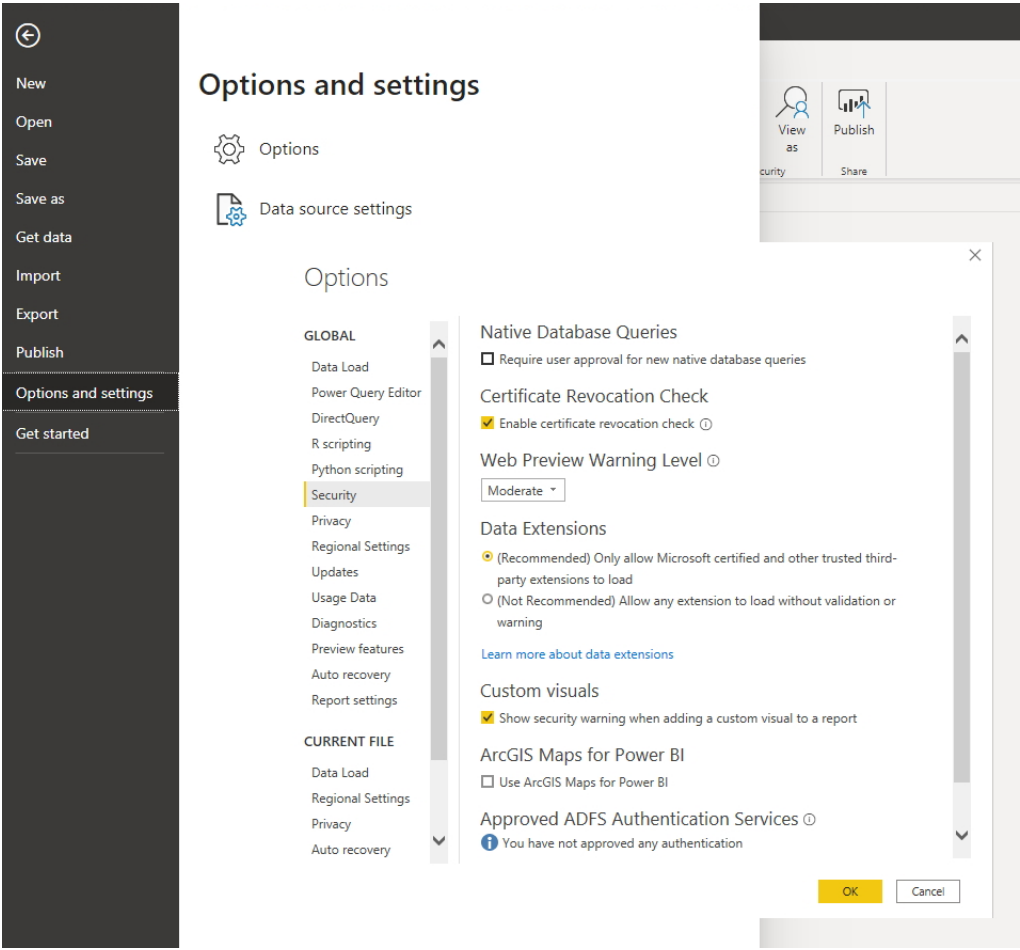


Figure 10.24 – Disabling native database queries

8. In Power BI Desktop, select **Options** and **Settings** from the **File** menu, and then click on **Security**.

9. Back in Power Bi Desktop, select **SQL Server Database** from the **Get Data** tab, before clicking on **Connect**.
10. Choose **SqlSrvInstance** from the **Server** drop-down list box and choose the **Database** option for the **Database** parameter:

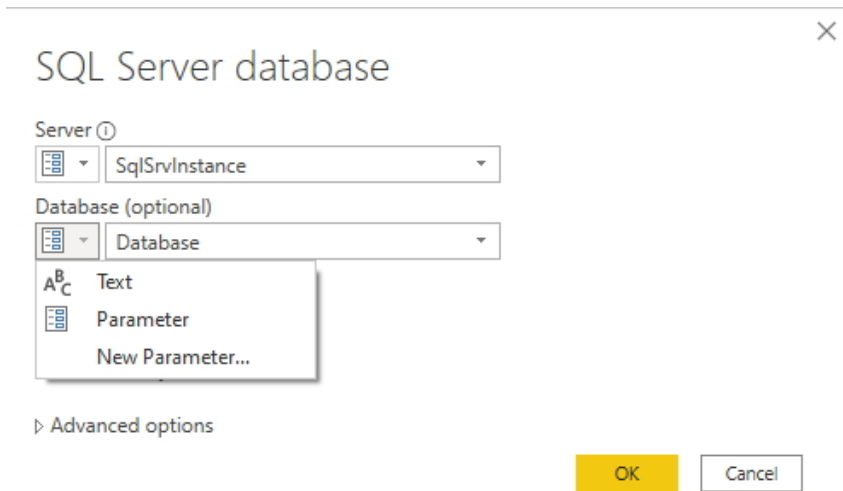


Figure 10.25 – The Server parameter

11. Click on the **Advanced options** arrow and paste the following T-SQL statement into the **SQL statement** box:

```
SELECT h.SalesPersonID AS RepID,
       CONCAT(p.LastName, ' ', p.FirstName) AS FullName,
       CAST(SUM(h.SubTotal) AS INT) AS SalesAmounts
FROM Sales.SalesOrderHeader h INNER JOIN Person.Person p
  ON h.SalesPersonID = p.BusinessEntityID
WHERE h.SalesPersonID IS NOT NULL
      AND YEAR(h.OrderDate) = 2012
GROUP BY h.SalesPersonID, p.FirstName, p.LastName
ORDER BY FullName ASC;
```

12. Click **OK**, and if everything is working correctly, you should get a preview that looks similar to the following:

□ ×

SqlSrvInstance: Database

ReplID	FullName	SalesAmounts
285	Abbas, Syed	151257
287	Alberts, Amy	560092
280	Ansman-Wolfe, Pamela	963421
275	Blythe, Michael	3985375
283	Campbell, David	1351422
277	Carson, Jillian	3396776
281	Ito, Shu	2387256
274	Jiang, Stephen	431089
284	Mensa-Annan, Tete	1269909
276	Mitchell, Linda	4111295
289	Pak, Jae	4106064
279	Reiter, Tsvi	2188083
282	Saraiva, José	1870884
286	Tsofilas, Lynn	836055
288	Valdez, Rachel	1245459
278	Vargas, Garrett	1389837
290	Varkey Chudukatil, Ranjit	2646078

Load
Transform Data
Cancel

Figure 10.26 – Preview of the script

If you look at the top of the preview, you will notice our two parameters—`SqlSrvInstance` and `Database`. So, this means everything is connected to the correct default parameters.

Click on **Load**, and this will load to the dataset. Before continuing, you might want to rename this to something more appropriate by either right-clicking on the **Fields** panel on the right or by clicking on the ellipses (...) on the right-hand side and then renaming it:

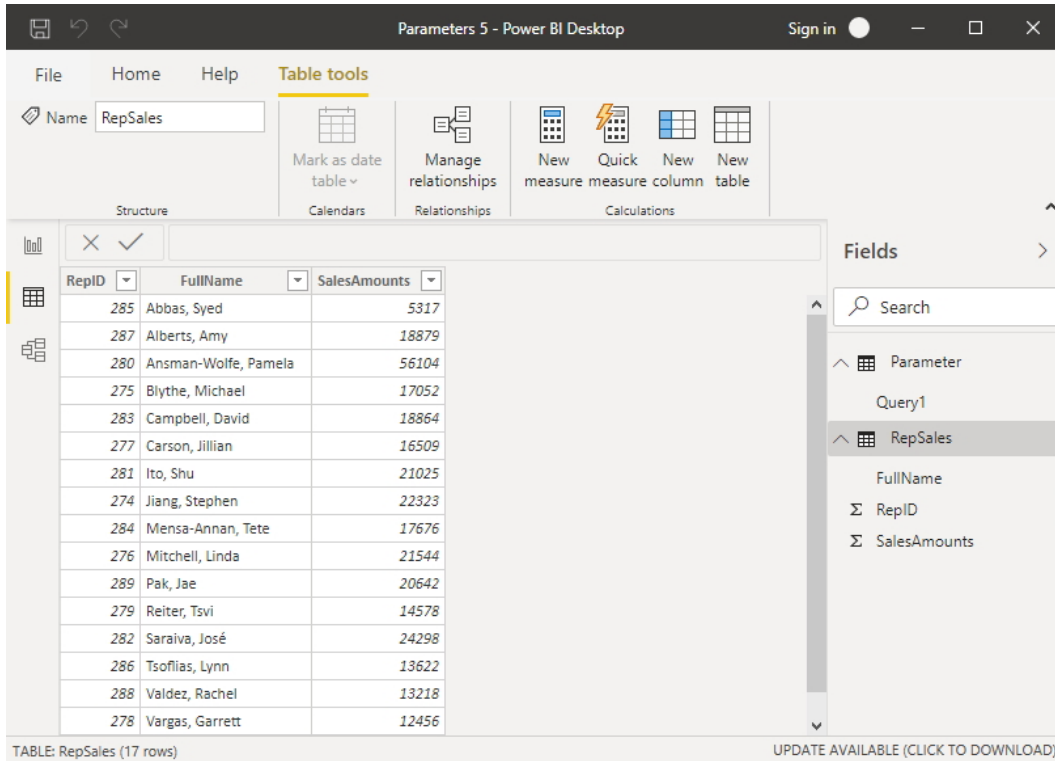


Figure 10.27 – The completed source parameter

The beauty behind this is that we have created two different parameters that we already set up for different SQL Server instances, as well as different databases. We can use the same parameters for different datasets, which means you can use the same connection information every time you have a dataset that uses the same data source.

If we look at the M statement that was generated when we used the T-SQL statement earlier, notice how easily the parameters have been referenced:

```
= Sql.Database(SqlSrvInstance, Database,
```

Although we have created two parameters, we still have **Filtered Row** in **APPLIED STEPS** in the preceding scenario. In the next section, we are going to look at how we can convert a date into a parameter.

Adding parameters to filter data

We can add in additional parameters so that it is possible to filter according to more than one thing. Part of the code that we used earlier is `AND YEAR(h.OrderDate) = 2012#`. What we can do is replace 2012 with a parameter, which will allow us to change the year with a new parameter. Go through the same steps as before (under *Parameterizing a data source*), but this time, create a parameter called `YearSales` and use a range of 2011 to 2014 for the list of values:

Parameters

New

ABC SqlSrvInstance

ABC Database

ABC YearSales X

Name

YearSales

Description

Year of the sale

☒ Required

Type

Text

Suggested Values

List of values

1	2011
2	2012
3	2013
4	2014
5	2015
6	

Default Value

2011

Current Value

2011

OK

Cancel

Figure 10.28 – The completed source parameter

Now that we have created the new parameter, we will need to change 2012 to " & YearSales % " (with the quotation marks) in the M code:

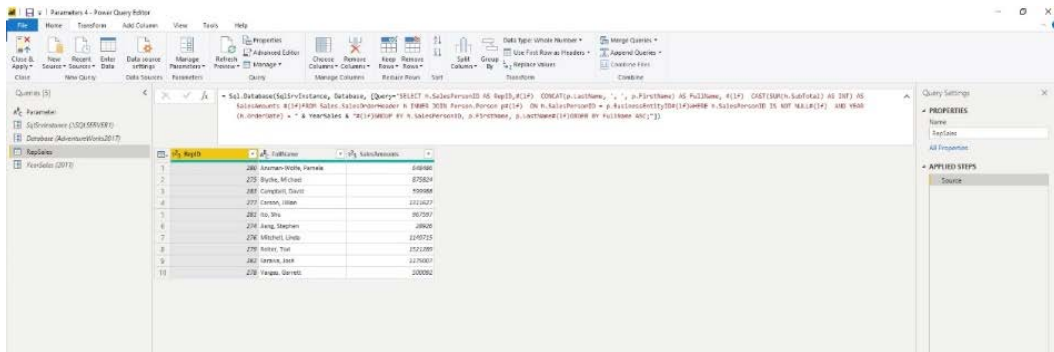


Figure 10.29 – Changing the year M code to a parameter

It is possible to see how well the new parameter works by selecting a different year and then going back to the RepSales parameter and viewing the sample. You will notice that each time you change the year, the figures are different. You might also notice that when you select the 2015 year, there is no data for this and the preview is blank:

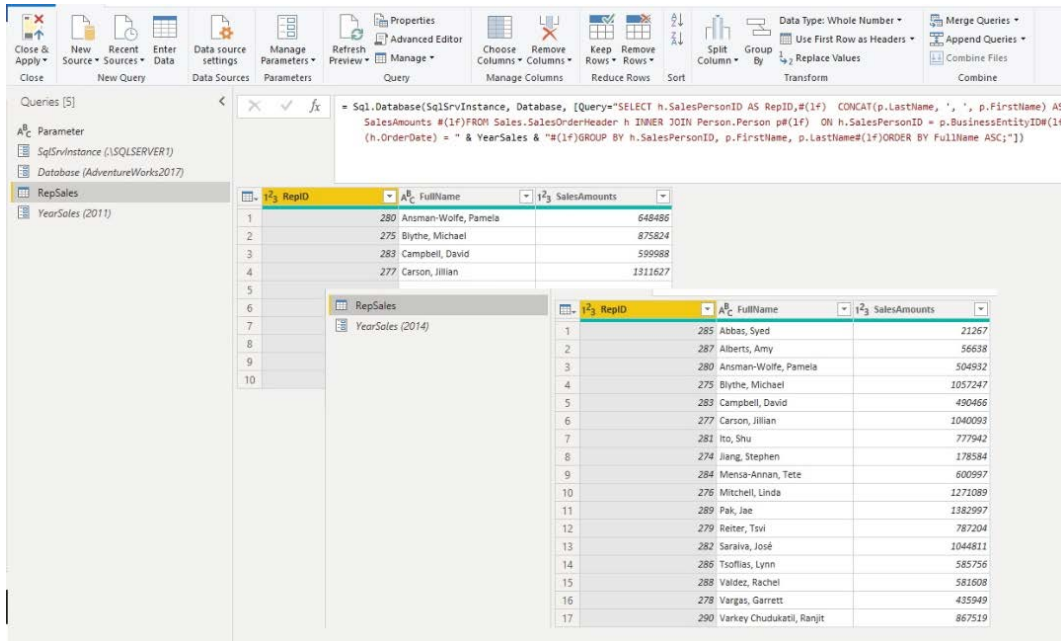


Figure 10.30 – Selecting different years

It is possible to expand on this and do more than just change the text. We can use parameters and change mathematical formulas as well, which we will look at in the next section.

Adding parameters to control statement logic

Up until now, all of the parameters that we have created change text from something to something else. It is possible to make the text do additional things, such as carry out mathematical equations. For example, we can change SUM to AVG, and this will then carry out a different equation. Although it is possible to use these in just about any mathematical equation, I am going to demonstrate SUM, AVG, MIN, and MAX

Create another parameter and call it MathsAgg with the values of SUM(h.SubTotal), AVG(h.SubTotal), MAX(h.SubTotal), and MIN(h.SubTotal):

The screenshot shows a 'Parameters' dialog box with a list of parameters on the left and configuration fields on the right. The 'MathsAgg' parameter is selected in the list. The configuration fields include:

- Name:** MathsAgg
- Description:** Change the maths aggregate to work out different sales amounts.
- Required:** ☒
- Type:** Text
- Suggested Values:** List of values
- Default Value:** SUM(h.SubTotal)
- Current Value:** SUM(h.SubTotal)

The 'List of values' section contains a table with the following data:

1	SUM(h.SubTotal)
2	AVG(h.SubTotal)
3	MAX(h.SubTotal)
4	MIN(h.SubTotal)
*	

At the bottom right, there are 'OK' and 'Cancel' buttons.

Figure 10.31 – The completed MathsAgg parameter

In the database, there might be various columns with figures that we could apply this to. For example, there might be a `DiscountAmounts` column and we could then use the `MIN(h.DiscountAmounts)` formula to work out the minimum amount for this column. We can have different parameters for each column, but only if the different datasets can support this.

We will once again need to update the `M` statement by replacing `SUM(h.SubTotal)` with `" & MathsAgg & "` (including the quotation marks):

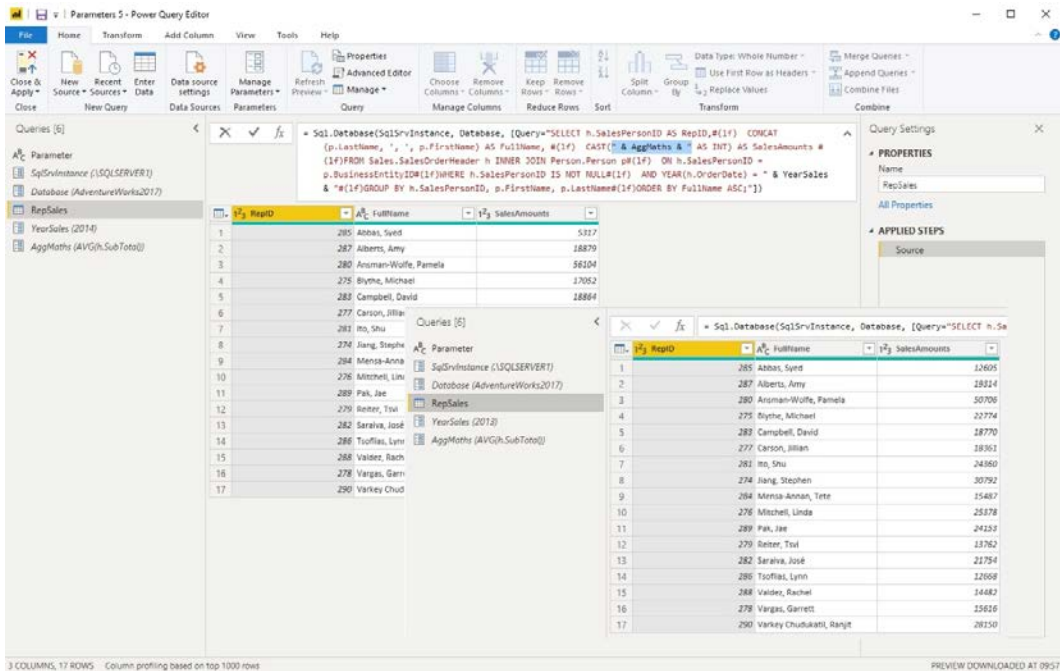


Figure 10.32 – The completed MathsAgg M code

Once again, if your dataset allows for this, you could use the same parameter that you created here and apply it to a different parameter that you created before—for example, the `DiscountAmounts` column that we mentioned earlier.

We have now used different mathematical aggregate functions and used them in a parameter, but in the next section, we will look at how we can use other built-in functions to create parameters to order things.

Adding parameters to order objects

We can order columns in Excel and Power BI using either the ascending or descending formula. We can use functions such as ordering on parameters to order various columns. With this, it is possible to use various columns as our values, which means that we can not only choose whether we would like something to be in ascending or descending order, but we can also have the different column names to choose from.

As we did earlier in the preceding section, we will create another parameter and call this one `OrderResults`. We will create four different values in our list—`FullName ASC`, `FullName DESC`, `SalesAmounts ASC`, and `SalesAmounts DESC`:

Parameters ×

New

ABC

SqlSrvInstance

ABC

Database

ABC

YearSales

ABC

AggMaths

ABC

OrderResults

×

Name

OrderResults

Description

Order specified columns in ascending or descending order

☒ Required

Type

Text

Suggested Values

List of values

1	FullName ASC
2	FullName DESC
3	SalesAmounts ASC
4	SalesAmounts DESC
*	

Default Value

FullName ASC

Current Value

FullName DESC

OK

Cancel

Figure 10.33 – The OrderResults parameter

Once the parameters have been created, change the M statement by changing `FullName ASC` to `" & OrderRes & "` (including the quotation marks):

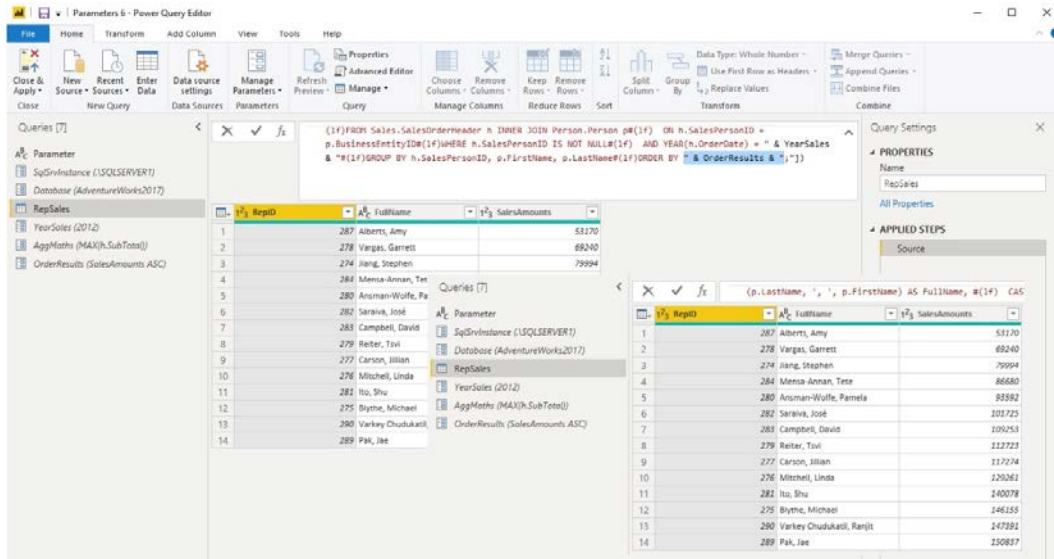


Figure 10.34 – The completed OrderResults parameter

In this chapter, we used parameters to filter different scenarios. In earlier chapters, we predominately used **APPLIED STEPS** to filter the data that we required. One of the problems with **APPLIED STEPS** is that everything is set in stone, and once you have the steps in order, it is not always that easy to change things around. Parameters, however, allow you to change multiple things without **APPLIED STEPS** as you are adding a type of filter, which is applied to the datasets and is then applied.

The one bit of advice that I do have at this point is that when you are creating your templates, every time you create a new parameter, you should apply and save the changes in a different filename. For example, in this section, we have created five different types of parameters, and I have named my file Chapter10.Parameter1, Chapter10.Parameter2, and so on. This was done for two reasons:

- I have a completed file for each parameter that I could use as an example.
- If I broke the M code at any time, I could always revert to the previous file.

The one drawback to using parameters is that up until this point, we have opened the Power BI Query Editor to change the parameters that we needed, which is time-consuming, but we can do this directly in the **Data** view. Let's look at that next.

Using parameters in the Data view

From the **Data** view, it is possible to change all of the parameters that we have created without having to go into the Query Editor, which makes things quicker and easier.

Click on **Edit Parameters** under **Transform data** in the **Home** ribbon, which will bring up the **Enter Parameters** window:

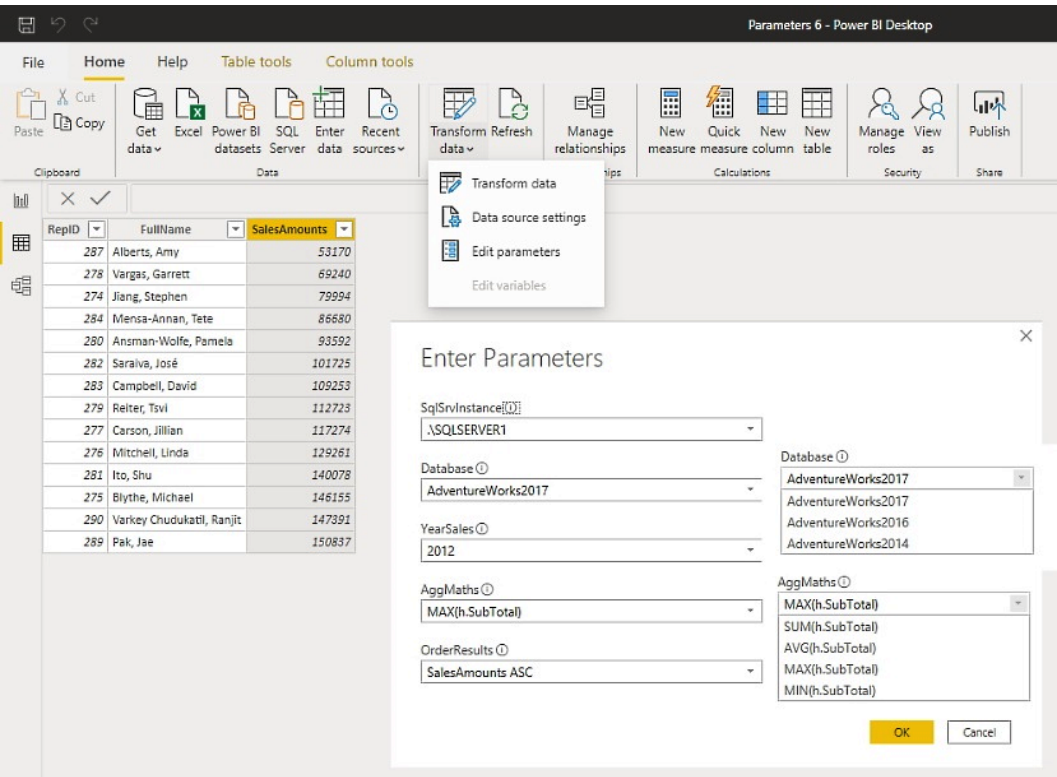


Figure 10.35 – The parameters from Data

The window itself is self-explanatory, and you can use the drop-down list boxes next to each parameter to choose the relevant filters that you would like. Once you are finished, click on **OK**, which will automatically filter according to the parameters that you have chosen. It is also possible to change the parameters while you are in **Report** view, which changes your visualizations automatically.

Once your parameters are set up, using the **Parameter** window makes it much quicker to filter your requirements without having to open the Power Query Editor.

Summary

Looking back at this chapter, we started off by looking at the differences between the Excel and Power BI languages and how concatenation is similar but uses different languages. The ampersand operator (&) was used to show how we can join different strings, dates, and objects before examining how this was different, but similar, to using `Text.From` and `Text.Combine`.

The setting up of a SQL server is a bit technical and normally, a technician would set this up in a medium to large corporation, although this is changing in the current climate with the use of scalable online platforms that allow you to purchase only what you require, which makes this more affordable for smaller organizations. My advice when installing is to make sure you concentrate on the different steps, as if you change a setting such as the authentication, you might not be able to access your databases.

Lastly, we investigated parameters. We took an intensive look at setting up parameters and accessing different data sources, as well as using parameters to filter data from previous years, using parameters to order columns in ascending and descending order. In the final part of this chapter, we looked at how we can set all of the parameter settings in Power BI's Data view without having to go into Query editor.

The next chapter concentrates on custom functions and delves into the custom functions of M.

11

Creating a Basic Custom Function

In *Chapter 9, Working with M*, we introduced the M Power Query language and learned how to use and write the M syntax, including steps to reveal a list of functions and definitions in Power Query.

This chapter will take you through the steps to create functions manually using the M functional language in Power Query, as well as how to create a date and time column using functions.

In this chapter, we're going to cover the following main topics:

- Creating a function manually using M
- Creating a `DateTime` column using three M functions

Technical requirements

You should be familiar with the content that we covered in *Chapter 9, Working with M*, and *Chapter 10, Examples of M Usage*, before delving into the topics covered here. It goes without saying that you should be proficient at importing various data sources into Excel or Power BI and be comfortable with the Power Query interface, being able to navigate it with ease. You need an internet connection to download the relevant files from GitHub. Also, the code files for this chapter can be found at the following link:

<https://github.com/PacktPublishing/Learn-Power-Query>

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=arwkny-WhIk&list=PLcLcwrwLel186O_GJEZs47WaZXwZjwTN83&index=12&t=0s.

Creating a function manually using M

Just like Excel, Power Query (and Power BI) has many functions that you can use to prepare or transform data. These are, of course, updated regularly by developers. When we use icons and set options in Power Query, the program works hard behind the scenes to generate the code to provide you with results to any functions you may apply to the data.

We can display a list of M functions using the `#shared` code in the Power Query formula bar, as explained in *Chapter 9, Working with M*, in the *Using #shared to return library functions* section. We can use this to build our data queries, but sometimes, we need to construct personalized functions to make our lives easier and less complicated. By less complicated, we mean being able to address repetitive tasks with one action instead of multiple actions or construct them so that they can be applied to many different queries or arguments. We will use a manual method of creating a custom function by constructing a step manually in the advanced editor to transform a query into a function.

In this section, we will create a function that can be applied to any file you need and that you can produce a set of data from. Imagine we work with monthly data; for instance, say we receive sales figures each month for our company's sales representatives and we need to export this data regularly to update a yearly projection. We can get Power Query to do this for us by creating a custom function. The values, of course, will be different in each monthly workbook depending on the scenario. This is because we use each scenario to create a function with one parameter to export.

Here, we will learn how to edit a query to change the file path, and then duplicate the query and work with Advanced Editor to convert code into a parameter to turn it into a custom function. We will then test and invoke the function by connecting to a folder and then display the results in a table:

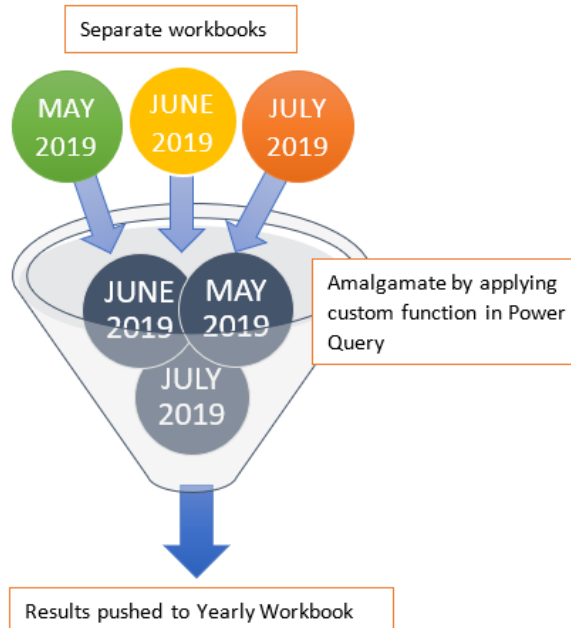


Figure 11.1 – Graphical representation showing how multiple results are transformed into one output table using a single parameter

For this example, we need a Power BI file called `ChoklatoFlakSales.pbix`, which includes the `MaySales` query.

You can, however, start from scratch and import an Excel workbook named `May-ChoklatoFlakSales` into Power BI if you prefer. Excel workbooks have exactly the same structure and format. For this example, we have used the `May-ChoklatoFlakSales.xlsx`, `June-ChoklatoFlakSales.xlsx`, and `July-ChoklatoFlakSales.xlsx` workbooks.

Let's get started. We will break our scenario down into three main steps.

Changing the file path of the query to a local path

As you will be using code files for this example, the path to the exercise files needs to be updated to your local computer location. This is to ensure that when opening the existing Power BI file (or the completed exercise file, for that matter) containing the query example, the software will try and load the tables from the new path. Let's see how to go about this:

Note

You can also load the workbook directly to Power BI from the GitHub path, should you prefer to do so.

1. Open Power BI Desktop, and then launch the file called `ChoklatoFlakSales.pbix`.
2. Within the Power BI file, we can see the `MaySales` query already in Power BI.
3. We need to transform our data using Power Query. Action Power Query by selecting **Home | Transform Data**.
4. Click on the gear icon next to **Source** in the **APPLIED STEPS** pane.
5. You will see that there is an error displayed on the main window as the file path is not recognized, and so the query will not load:

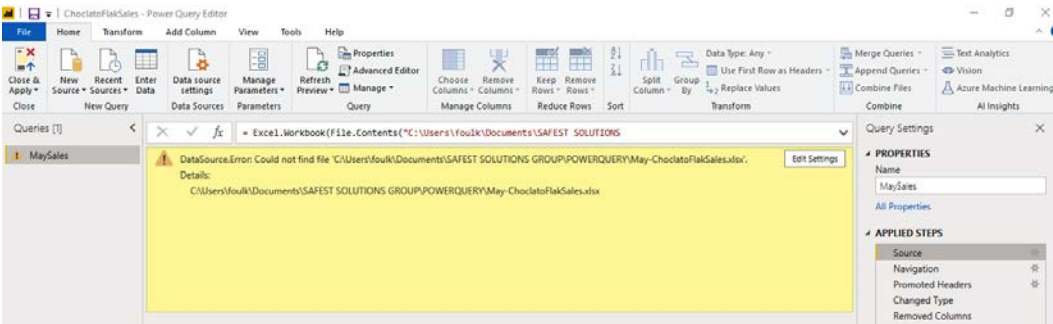


Figure 11.2 – Displayed error due to incorrect file path on query settings in Power Query

6. Click on **Browse...** to change the file path. Navigate to the location on the computer where the `May-ChoklatoFlaksales.xlsx` workbook resides. Select the file to update the path, and then click on **Open**. Click on the **OK** command to change the path:

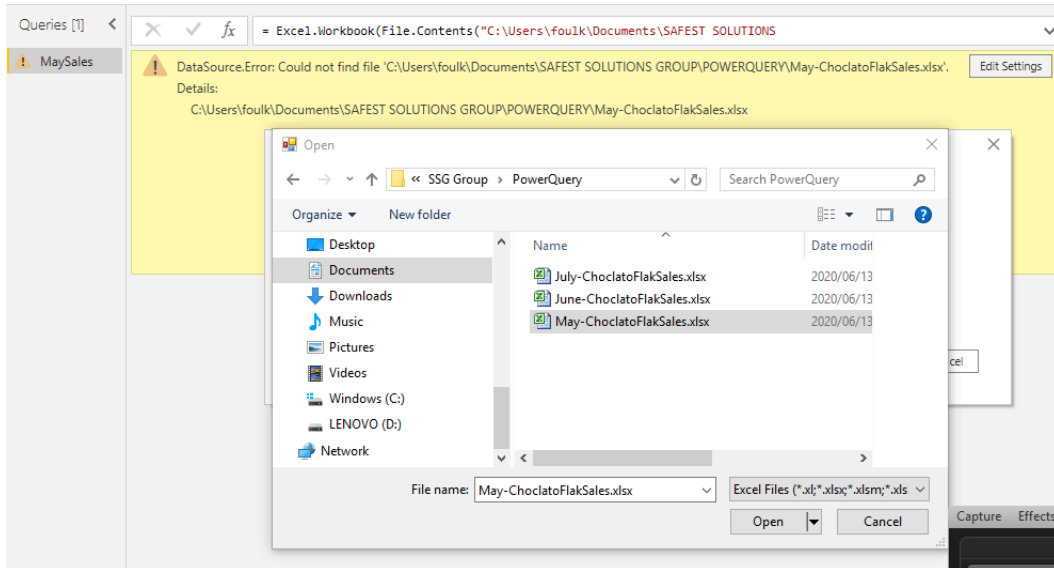


Figure 11.3 – Updating the file path in Power Query

7. The query is now updated in Power Query and the query will load.

Next, let's see how to create a function from a query manually.

Creating the function manually

We now need to transform our query so that it becomes a function, after which we will assign a parameter to the query function. So, what is a parameter? Parameters were explained in *Chapter 10, Examples of M Usage*, in the *Using parameters* section.

Let's see how to transform the query:

1. We will firstly duplicate the query called MaySales in Power Query.
2. Right-click on the MaySales query and choose **Duplicate** from the drop-down menu provided.
3. A copy of the query is now visible under the existing MaySales query, named MaySales (2).

4. Open **Advanced Editor** to edit the code for the query by selecting **View | Advanced Editor**:

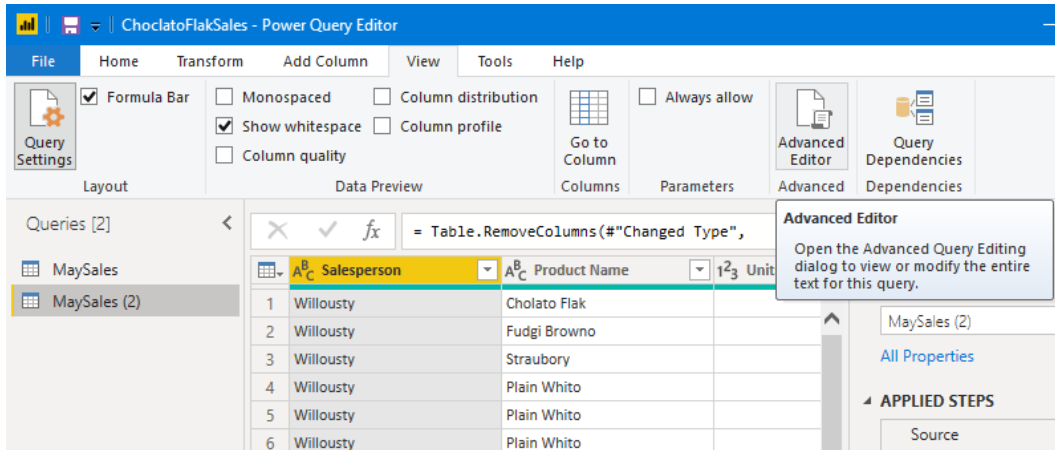


Figure 11.4 – Accessing Advanced Editor in Power Query

5. We now need to select part of the code to edit into a parameter so that we are able to run this code on any data source within the source folder. The following screenshot identifies the portion of the code that we will need to change via the highlighted code. The highlighted portion is currently the source path location of the data source we imported into Power BI Desktop:

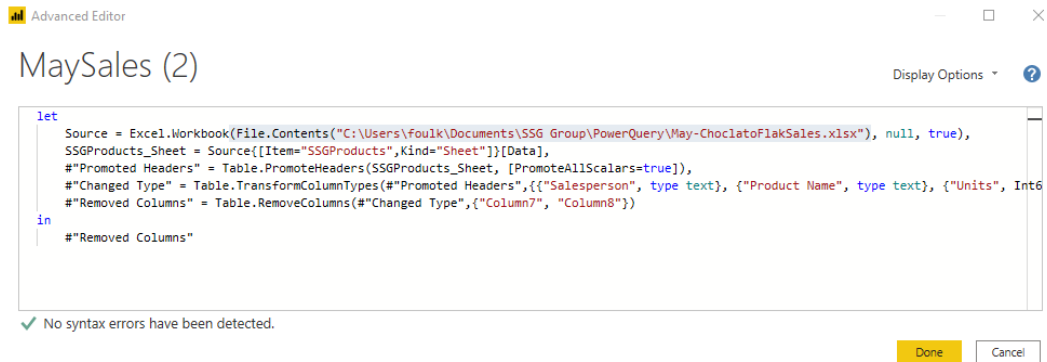


Figure 11.5 – The highlighted code that we will turn into a parameter query using Advanced Editor

6. To make the change to the source path, we will add the following line of code above the **let** syntax:

```
(FileBinary as binary) as table =>
```

This is shown in the following screenshot:



Figure 11.6 – The Advanced Editor view edited to include the parameter code above the let call

- Click on **Done** to change the highlighted code into a parameter query. Notice that because we edited the code, the MaySales (2) query has turned into a parameter query. We can identify the parameter query thanks to the **fx** icon located to the left of the query name:

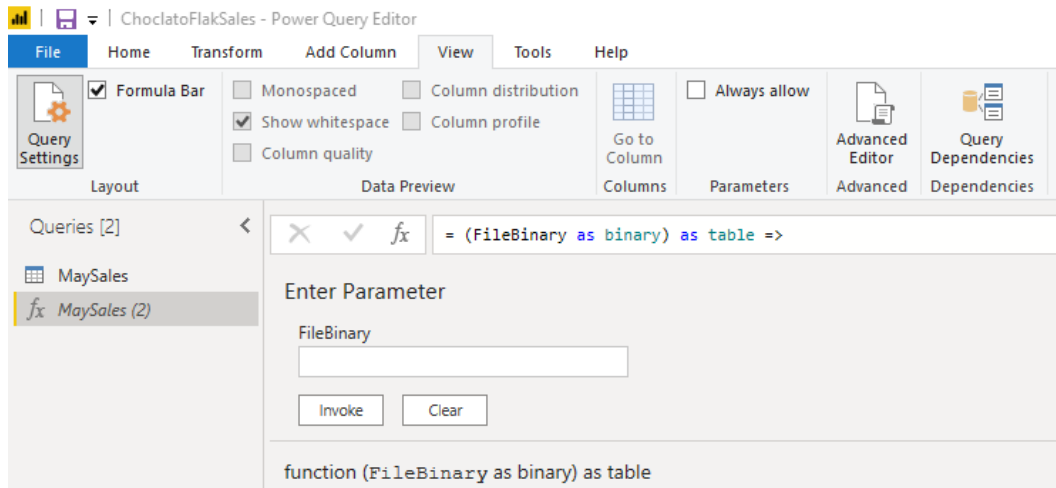


Figure 11.7 – The parameter query applied to the query function

In this section, we learned how to edit a query using Advanced Editor to create a parameter query. Now, let's test the parameter function that we have created in the next section.

Testing the parameter function

Here, we need to ensure that the parameter query works for us by linking it to a source folder so that it can do its job. Remember that all the files in the folder need to be of the same file type, otherwise you may encounter an error. Let's see how we test the function:

1. Click on **Home** | **New Source** | **More...** | **Folder** | **Connect**:

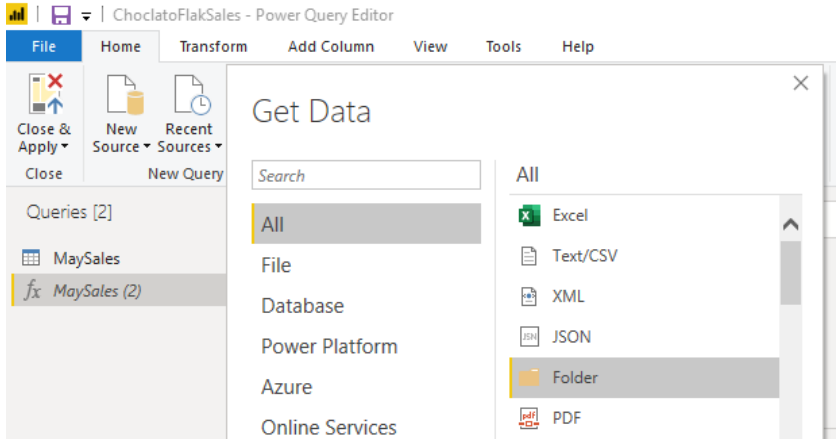


Figure 11.8 – Creating a new folder connection

2. Then, click on **Browse...** to find the file path you wish to connect to:

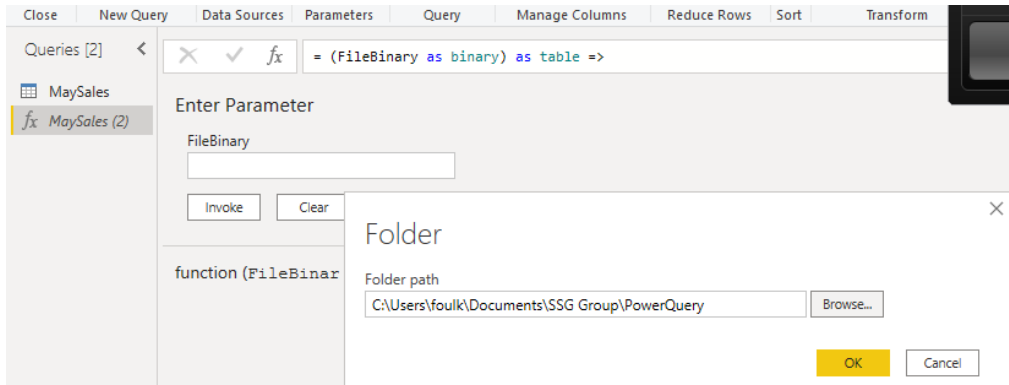


Figure 11.9 – Selecting the folder path

3. Click on **OK** to accept the folder.

- Then, select **Transform Data** to add a new query to the Power Query interface:

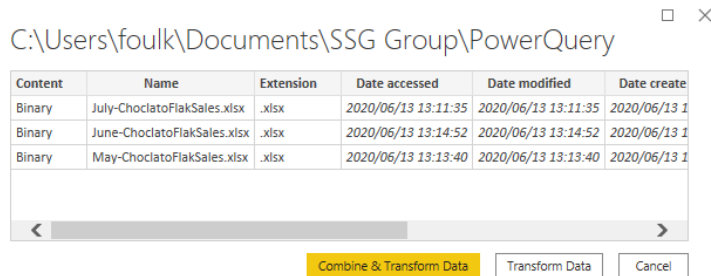


Figure 11.10 – Choosing Transform Data to add the folder connection to Power Query

We will now use the **Invoke Custom Function** option to apply our function to the files that we have added to the folder, as selected in the previous step. Essentially, what we are getting here is a new column with the parameter query, which will invoke the function.

- Click on **Add Column | Invoke Custom Function**.
- Provide a name for the new column in the **New column name** field. For this example, we have used `SalesInvoke`. Choose **MaySales (2)** as the function query. Leave the **FileBinary** option as **Content**, as shown:

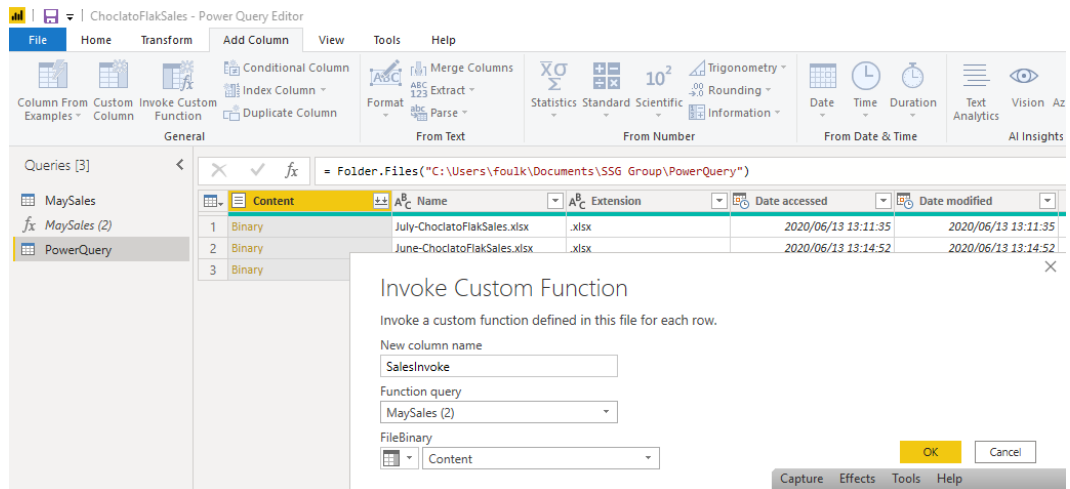


Figure 11.11 – Invoking a custom function

- Click on **OK** to continue.

8. The new column, SalesInvoice, is added to the query:

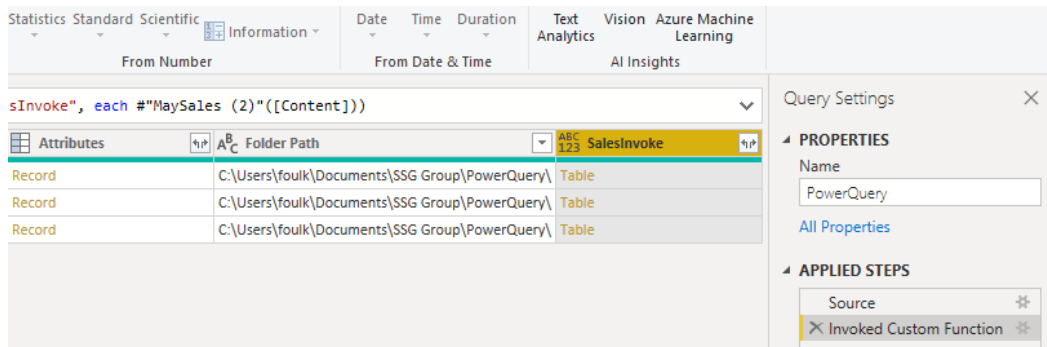


Figure 11.12 – The new column is added to the query

9. Click on the **SalesInvoice** column to view the results for each of the source files:

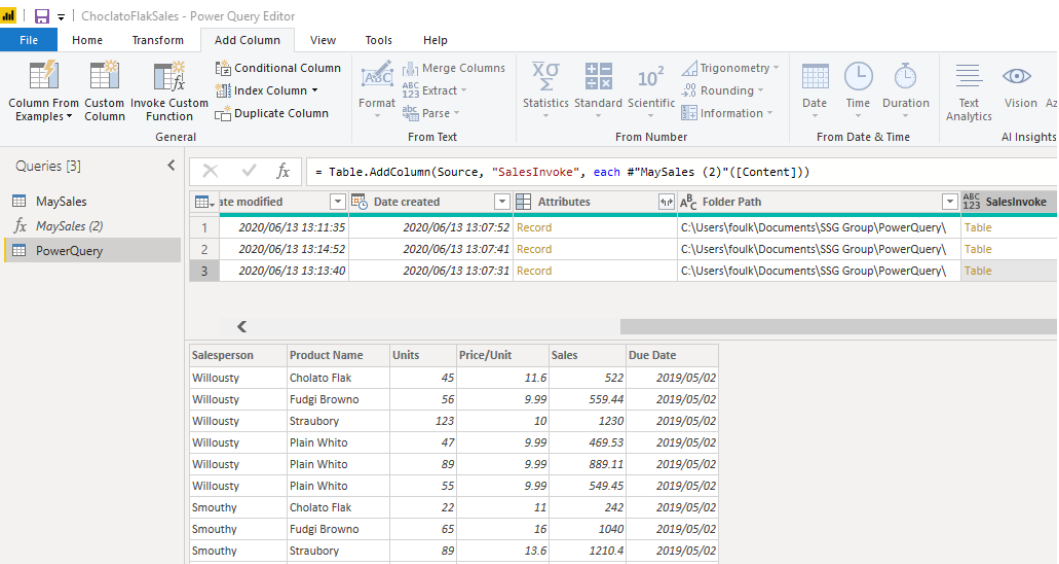


Figure 11.13 – Clicking on the whitespace in the last column will display the content of the file in the window below

10. Please note that if you happen to have file formats within the source folder that are not the same as the file type you are invoking, you could encounter an error. For example, we are invoking an Excel file format in this example. If I had .csv files in this folder, or any other file type, then I would receive errors in certain cells because of the file type difference. Remove the files from the folder before invoking the function.

11. Remove any columns you do not need for the custom function. Observe the following screenshot:

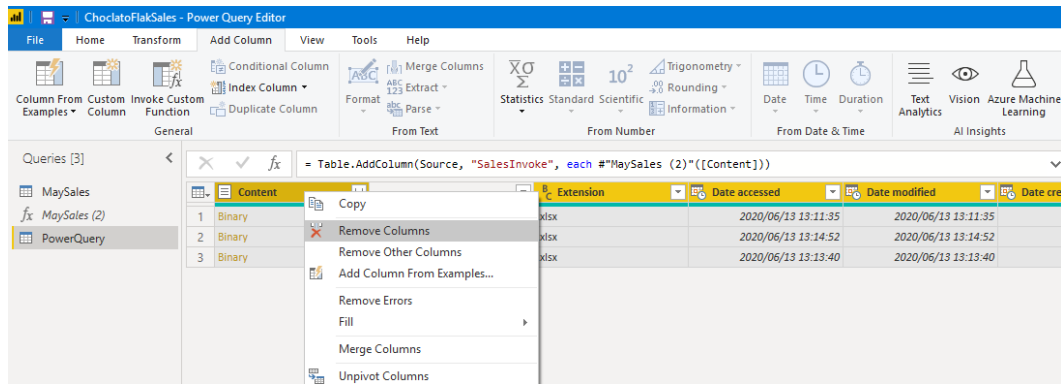


Figure 11.14 – Removing unnecessary columns from the query

In this example, I have removed all the columns except Name and SalesInvoice. You should now be left with the following columns in the query:

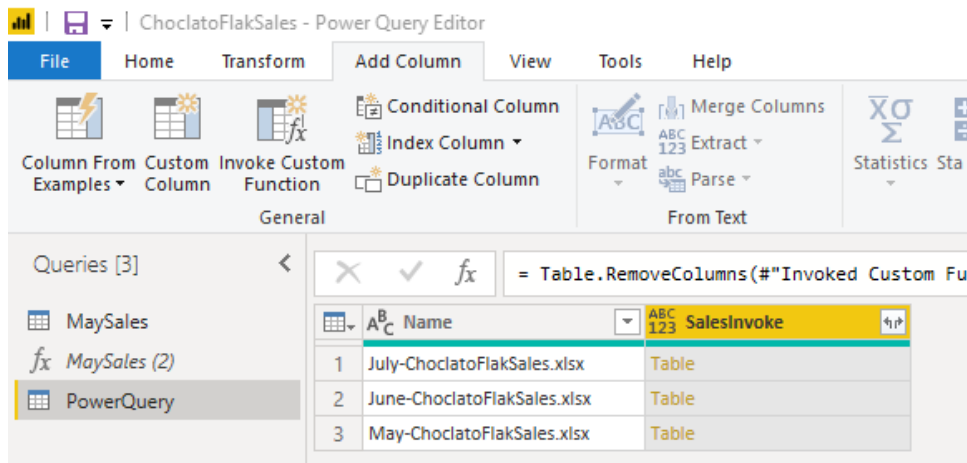


Figure 11.15 – The columns that remain after removing them from the query

12. The final step in this process is to append all the data from the Excel workbooks into a single workbook so that we see the May, June, and July sales in one combined result:

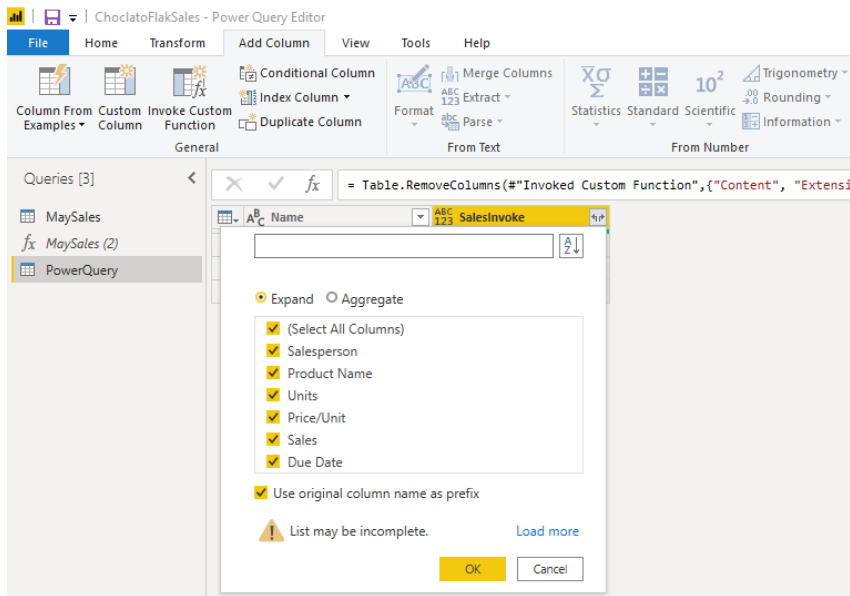


Figure 11.16 – Clicking on the double composite arrows (expand) will show the append options

13. Click on **OK** to view the result.

The beauty of this process is that any new datasets added to the folder path will be appended to the table. All you need to do is refresh the data connection.

14. Add the August -ChoklatoFlakSales.xlsx workbook to the folder path and watch the magic happen:

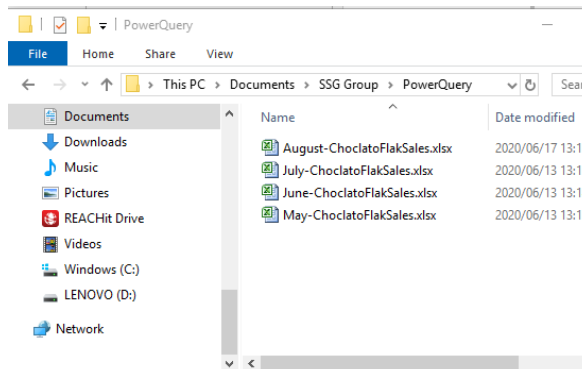


Figure 11.17 – The August workbook is added to the folder

15. Click on the **Refresh All** icon on the **Home** tab of the Power Query interface to refresh the data. You will now see that the August sales have been added to the dataset:

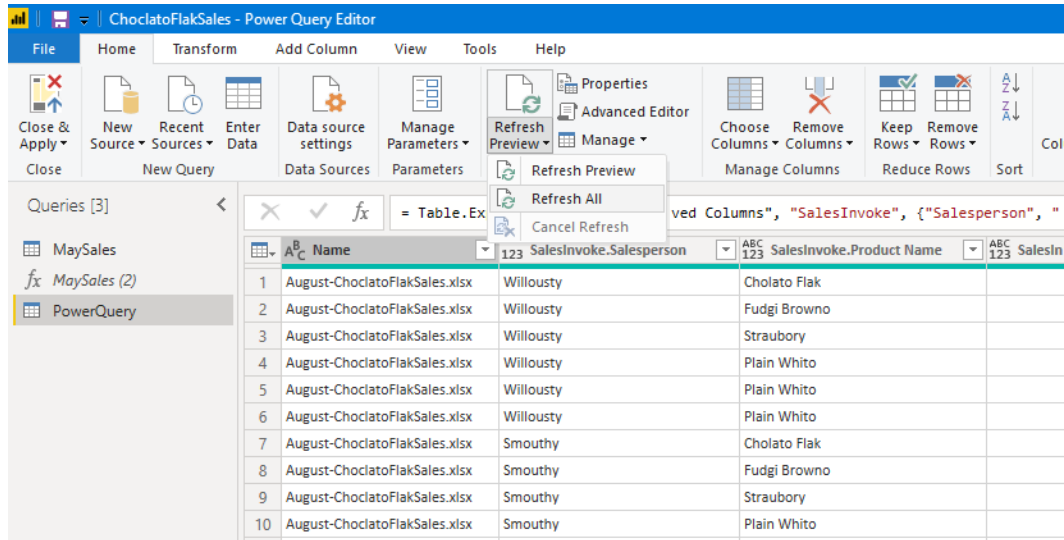


Figure 11.18 – Clicking on the Refresh All icon to pull in the new dataset

It is important to note that there are a number of methods to achieve the result in the preceding example. The steps you have followed so far give you a good indication of the magic that Power Query can provide when you need to update a workbook by appending datasets. It is necessary to understand this process using the manual method, as used here. This will benefit you more than clicking on a single button and not knowing how the application works behind the scenes, using functions and parameters. The shorter method uses the **Combine Files** icon on the **Home** tab:

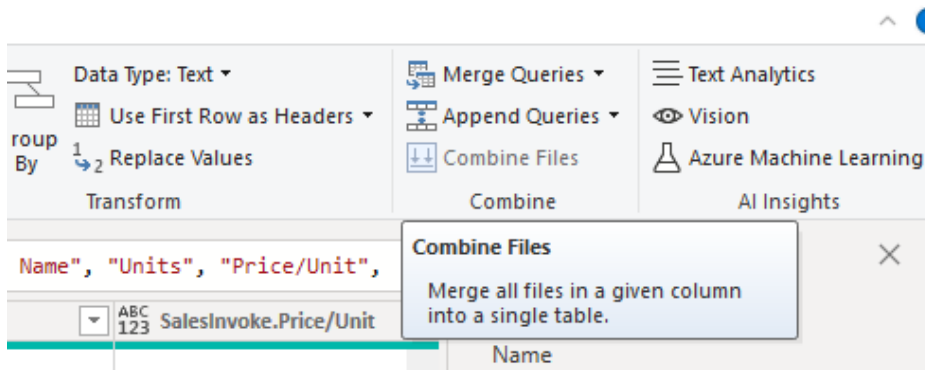


Figure 11.19 – The Combine Files method

If you are an experienced user of M, you will be able to create the code for all of the preceding steps without using the interface as we have done in this example. I prefer to use the manual method so that I understand what I am doing more thoroughly than having to switch between code and testing all the time.

You can also use the **Manage Parameters** icon in the **Home** tab to create a binary parameter, after which you could create a reference by right-clicking on the binary file. The next step would be to create the custom function by right-clicking on the created parameter query and then selecting **Create Function**.

The functions you want to perform to transform data are then actioned manually, after which you would use **Invoke Custom Function**. So, the steps are very similar; they just depend on the method that you prefer.

Having understood this, we will move on to the next section to create a date/time column using three M functions.

Creating a date/time column using three M functions

In this section, we will learn how to create a function manually by creating a `DateTime` column in Power Query using three M functions, namely, `List.Datetimes`, `Duration.TotalDays`, and `Time.LocalNow()`.

A comprehensive reference for all the date/time functions in Power Query can be found at <https://docs.microsoft.com/en-us/powerquery-m/date-functions>.

The great thing about this Microsoft documentation is that you can copy code directly from the website and paste it into your Power Query Advanced Editor window or the formula bar. That way, you do not need to remember the code and can save a lot of time when constructing M code. Locate the code you wish to use and then click on the **Copy** button at the top right of the code:

Example

Create a list of 10 values starting from 5 minutes before New Year's Day (`#datetime(2011, 12, 31, 23, 55, 0)`) incrementing by 1 minute (`#duration(0, 0, 1, 0)`).

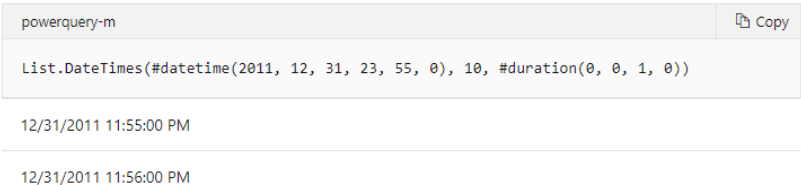
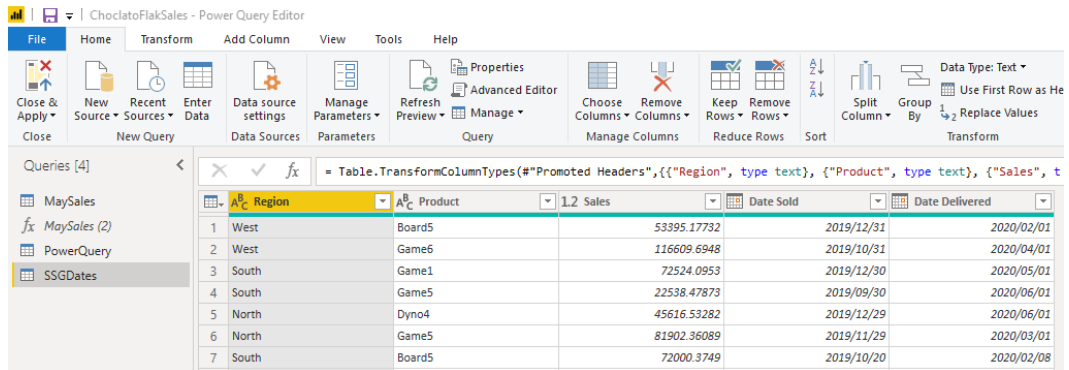


Figure 11.20 – Copying M code from the docs.microsoft.com website

Before we delve into the M date functions, let's look at a really easy method to subtract dates in Power Query without having to remember any code.

If you want to find the number of days between two dates in Power Query, you can use the `subtract_dates` function. This function is readily available from the **Add Column** tab in Power Query. Perform the following steps to calculate the number of days between two dates:

1. In the `SSGDates` query, we have a table listing products, as well as the `Date Sold`, and `Date Delivered` columns. We want to subtract the two dates from each other to find the number of days between the sold and delivered dates. The first step is to select the columns to subtract. The order of the date columns is important when selecting. If we select **Date Sold** before **Date Delivered**, the formula will calculate a negative number due to the `Date Sold` date being before `Date Delivered`. If you want a positive integer, then select **Date Delivered** before selecting the **Date Sold** column:



	Region	Product	Sales	Date Sold	Date Delivered
1	West	Board5	53395.17732	2019/12/31	2020/02/01
2	West	Game6	116609.6948	2019/10/31	2020/04/01
3	South	Game1	72524.0953	2019/12/30	2020/05/01
4	South	Game5	22538.47873	2019/09/30	2020/06/01
5	North	Dyno4	45616.53282	2019/12/29	2020/06/01
6	North	Game5	81902.36089	2019/11/29	2020/03/01
7	South	Board5	72000.3749	2019/10/20	2020/02/08

Figure 11.21 – The `SSGDates` query

2. So, first select the **Date Delivered** column, and then **Date Sold**. Click on **Add Column | Date | Subtract Days**:

The screenshot shows the Power Query Editor interface. The 'Date' menu is open, and 'Subtract Days' is selected. The data table has columns: Region, Product, Sales, and Date Sold. The formula bar shows: `= Table.TransformColumnTypes("#Promoted Headers",{{"Region", type text`

	Region	Product	Sales	Date Sold
1	West	Board5	53395.17732	
2	West	Game6	116609.6948	
3	South	Game1	72524.0953	
4	South	Game5	22538.47873	
5	North	Dyno4	45616.53282	
6	North	Game5	81902.36089	
7	South	Board5	72000.3749	
8	North	Game10	99227.6778	
9	West	Dyno1	93520.54806	
10	South	Game5	77445.82665	2020/09/29
11	South	Dyno5	38690.90909	2020/11/09
12	South	Game2	81105.31419	2019/12/09

Figure 11.22 – Calculating the number of days between dates using the Subtract Days function

3. The result is entered in a separate column, showing the number of days between the two dates:

The screenshot shows the Power Query Editor interface. The formula bar shows: `= Table.RenameColumns("#Inserted Date Subtraction",{{"Subtraction", "Number Days"}})`. The data table has columns: Product, Sales, Date Sold, Date Delivered, and Number Days. The formula bar shows: `= Table.RenameColumns("#Inserted Date Subtraction",{{"Subtraction", "Number Days"}})`

Product	Sales	Date Sold	Date Delivered	Number Days
Board5	53395.17732	2019/12/31	2020/02/01	32
Game6	116609.6948	2019/10/31	2020/04/01	153
Game1	72524.0953	2019/12/30	2020/05/01	123
Game5	22538.47873	2019/09/30	2020/06/01	245
Dyno4	45616.53282	2019/12/29	2020/06/01	155
Game5	81902.36089	2019/11/29	2020/03/01	93
Board5	72000.3749	2019/10/20	2020/02/08	111
Game10	99227.6778	2019/10/01	2019/12/01	61
Dyno1	93520.54806	2019/11/15	2020/03/23	129
Game5	77445.82665	2020/09/29	2020/04/04	-178

Figure 11.23 – The number of days is generated in a separate column

- Let's view the M code for this function. Click on **View | Advanced Editor** to see the code. This is what we get:

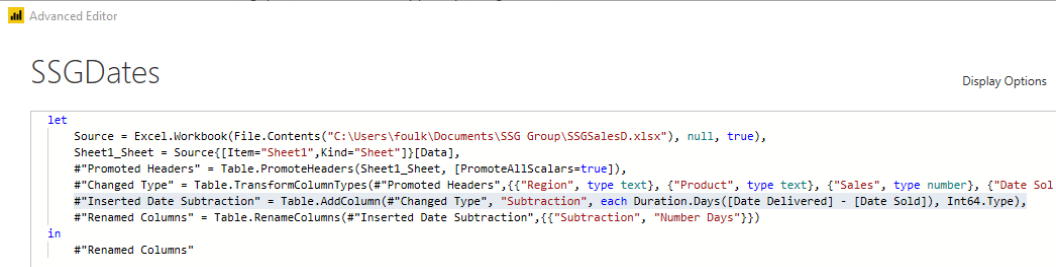


Figure 11.24 – The M code for the Subtract Days function

In the preceding example, we looked at one method to subtract days from existing date columns. For the next part of this section, we will get to know the `List.DatesTimes` M function. This function allows you to generate a list of dates or times based on your input.

The first step is to calculate the number of days between two dates:

- Create a new query named `NumbDays`.
- Click on **View | Advanced Editor**.
- Change the name of the `Source` code text to `NumbDays`. This will be our M formula function name, which we can use at a later stage in our `List.DatesTimes` formula:

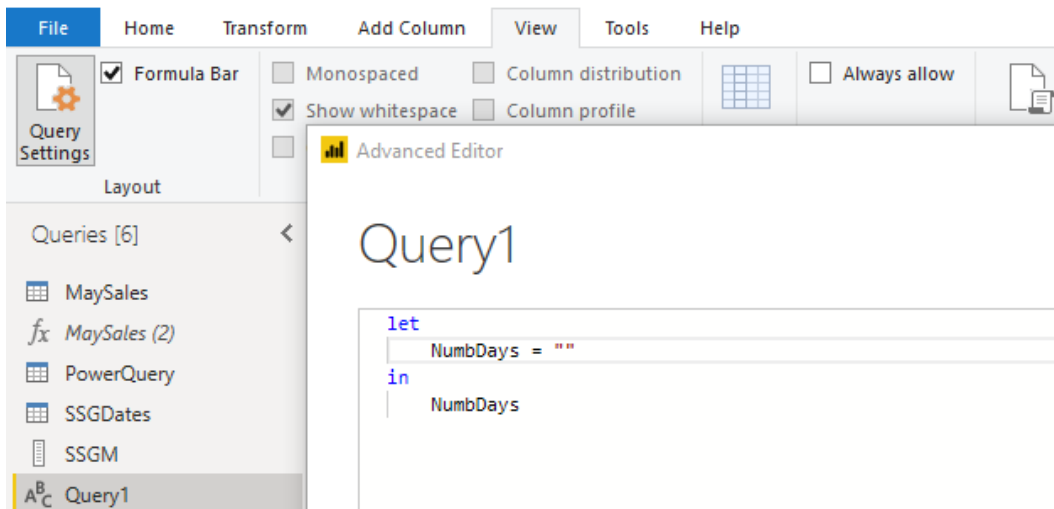


Figure 11.25 – Renaming the source code `NumbDays`

4. We will use the following calculation to subtract the two dates:

```
= #datetime(2019,01,01,00,00,00) -  
#datetime(2020,06,17,00,00,00)
```

5. Click on **Done** to view the number of days between the two dates:

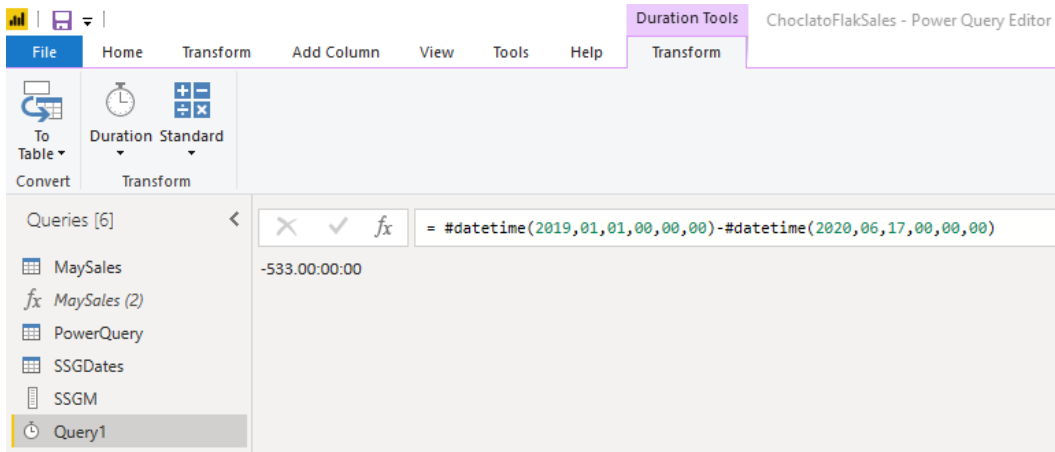


Figure 11.26 – Subtracting the dates to obtain the number of days between them

6. As we have a negative value for the number of days, simply switch the M code around to produce a positive value:

```
= #datetime(2020,06,17,00,00,00) -  
#datetime(2019,01,01,00,00,00)
```

7. Add a step by clicking on the **fx** icon in the formula bar, as follows:

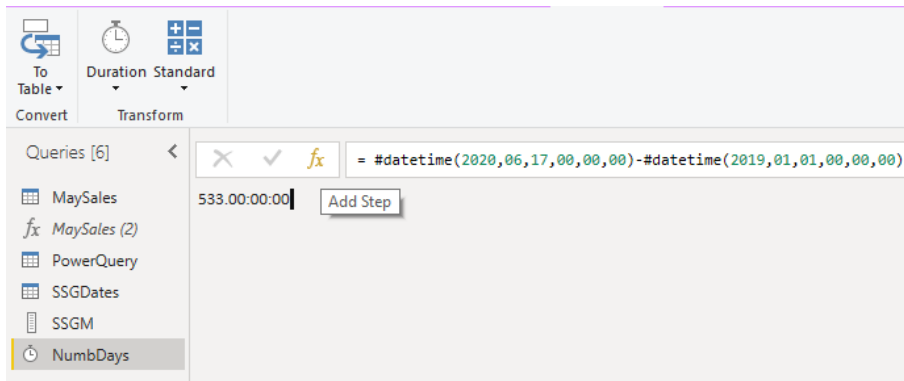


Figure 11.27 – Creating a step

8. When we construct M code, we can type the code into the **Advanced Editor** window or in the formula bar. Enter the following formula into the formula bar:

```
List.Datetimes(#datetime(2019, 01, 01, 00, 00, 0), 10,
#duration(0, 0, 1, 0))
```

This code will generate a date and time list of 10 records from January 1, 2019 with a 1-second interval:

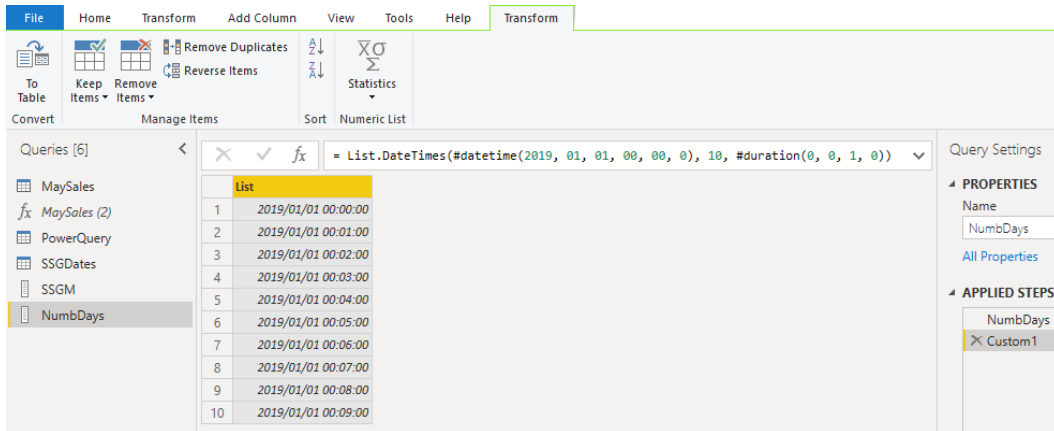


Figure 11.28 – The List.Datetimes M code

Here, we have used the second operator, but you can do this for days, hours, minutes, seconds, and so on.

9. We will edit the M code so that the list reflects every hour and remove the minute operator by entering 0:

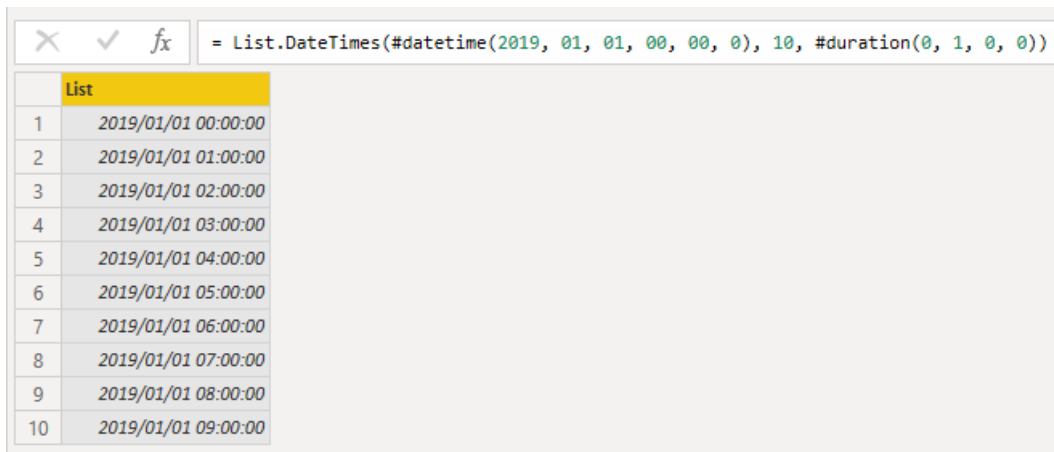
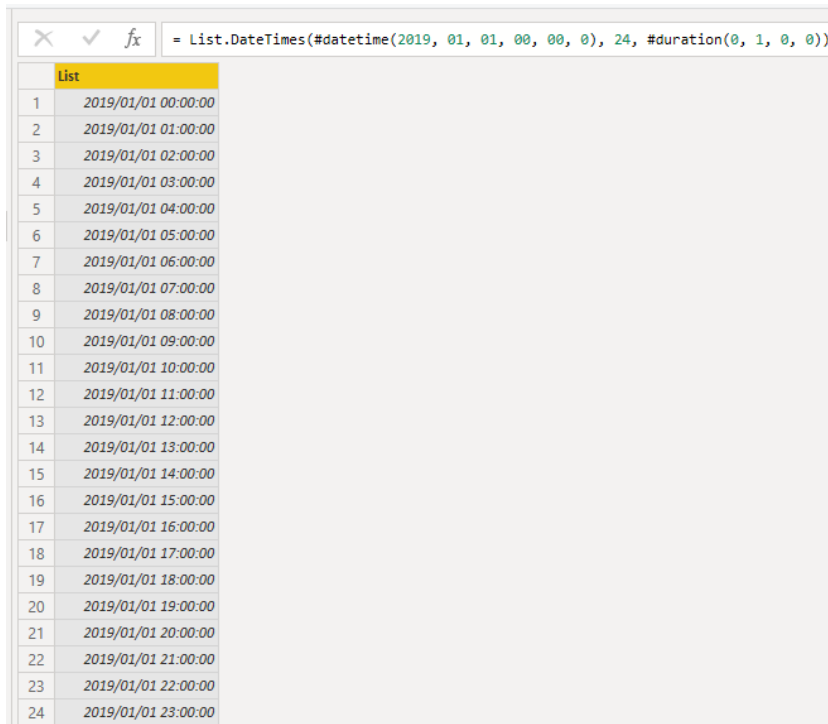


Figure 11.29 – The M code edited to reflect the increment by day, and not minutes

10. Let's update the number of lines to 24 to meet the requirements for the number of hours in a full day:

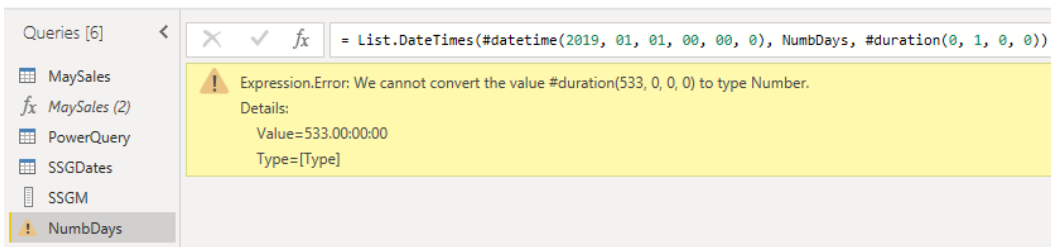


The screenshot shows the Power Query editor with a formula bar containing the M code: `= List.DateTimes(#datetime(2019, 01, 01, 00, 00, 0), 24, #duration(0, 1, 0, 0))`. Below the formula bar is a table with 24 rows, each representing an hour of the day on January 1, 2019. The first row is highlighted in yellow.

	List
1	2019/01/01 00:00:00
2	2019/01/01 01:00:00
3	2019/01/01 02:00:00
4	2019/01/01 03:00:00
5	2019/01/01 04:00:00
6	2019/01/01 05:00:00
7	2019/01/01 06:00:00
8	2019/01/01 07:00:00
9	2019/01/01 08:00:00
10	2019/01/01 09:00:00
11	2019/01/01 10:00:00
12	2019/01/01 11:00:00
13	2019/01/01 12:00:00
14	2019/01/01 13:00:00
15	2019/01/01 14:00:00
16	2019/01/01 15:00:00
17	2019/01/01 16:00:00
18	2019/01/01 17:00:00
19	2019/01/01 18:00:00
20	2019/01/01 19:00:00
21	2019/01/01 20:00:00
22	2019/01/01 21:00:00
23	2019/01/01 22:00:00
24	2019/01/01 23:00:00

Figure 11.30 – The 24 rows in the list reflecting the number of hours in a day

11. Go back to the NumbDays step, where you will see the number of days between the two dates. For this example, you will see 533 days between the two dates. We will use the NumbDays variable instead of the 24 days in the M code to reflect these 533 days. Enter NumbDays into the formula in place of 24. You will see an error occur immediately. The reason for this is that the variable entered is not actually a number, so we need to tweak this a bit:



The screenshot shows the Power Query editor with the formula bar containing the M code: `= List.DateTimes(#datetime(2019, 01, 01, 00, 00, 0), NumbDays, #duration(0, 1, 0, 0))`. The left sidebar shows a list of queries, with 'NumbDays' selected. A yellow error message box is displayed, indicating that the variable 'NumbDays' cannot be converted to a number.

Queries [6]

- MaySales
- MaySales (2)
- PowerQuery
- SSGDates
- SSGM
- NumbDays

Expression.Error: We cannot convert the value #duration(533, 0, 0, 0) to type Number.
 Details:
 Value=#duration(533, 0, 0, 0)
 Type=[Type]

Figure 11.31 – The edited M code to reflect the NumbDays variable

12. This is where the `Duration.TotalDays` M function comes into play. Go back to the **NumbDays** step and edit the M code to read the following:

```
=Duration.TotalDays(#datetime(2020,06,17,00,00,00) -  
#datetime(2019,01,01,00,00,00))
```

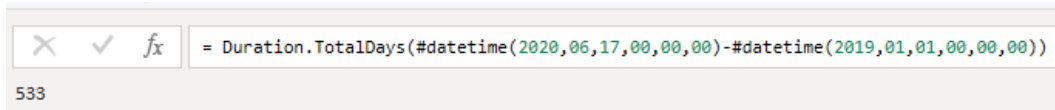


Figure 11.32 – The variable is now a number by adding the `Duration.TotalDays` M code

13. If you click on the **Custom1** step, you will now see that the 533 records have updated.
14. The next step is to turn the values into a data table. Click on the **To Table** icon at the top left of the ribbon:

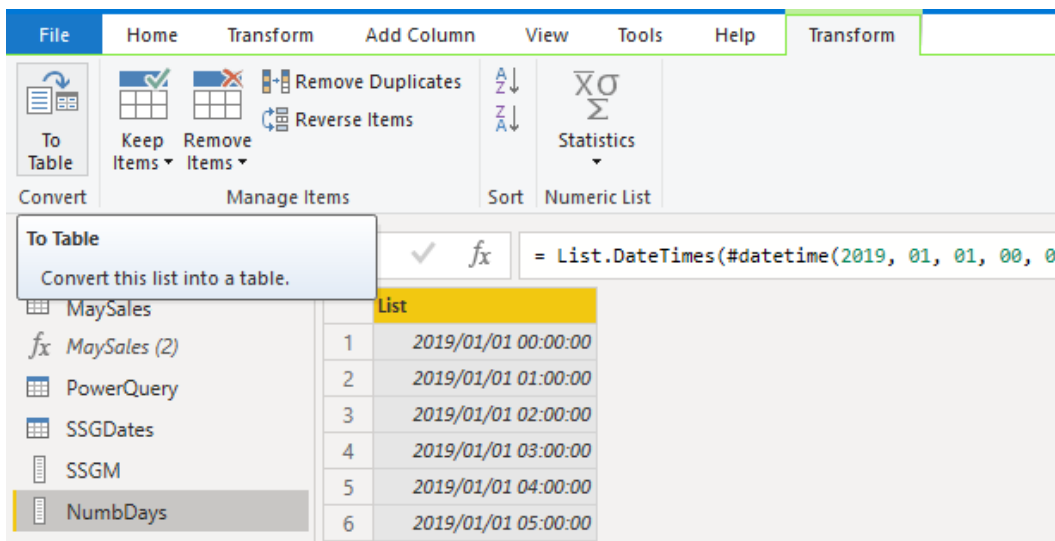


Figure 11.33 – Creating a table from the date values

15. Click on **OK** to select the defaults and convert them into a table.

16. Sort the table into ascending order to check that it shows January 1, 2019, and then sort it into descending order. If you find that some days are missing when you have sorted it into descending order, you need to tweak the M code a little further. This is because we don't have the required 25 lines per day. We need to multiply the first step by 24 so that all the hours in the day are covered:



Figure 11.34 – Editing the M code by multiplying by 24 to reflect the required number of hours in each day

17. If we now move back to view the **Sorted Rows** step, we will see that the calendar is still incorrect as it should be counting from 00 in the code. We now need to edit the code further to include `DateTime.LocalNow()`, which will return the date and time of the system's current date and time:

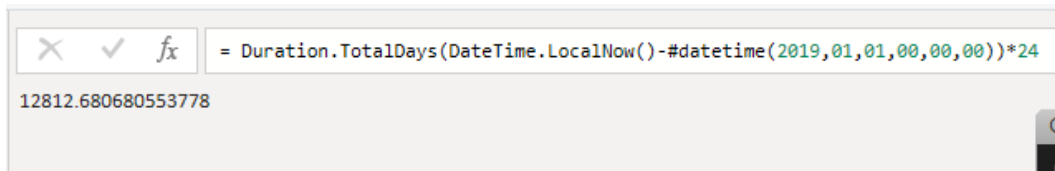
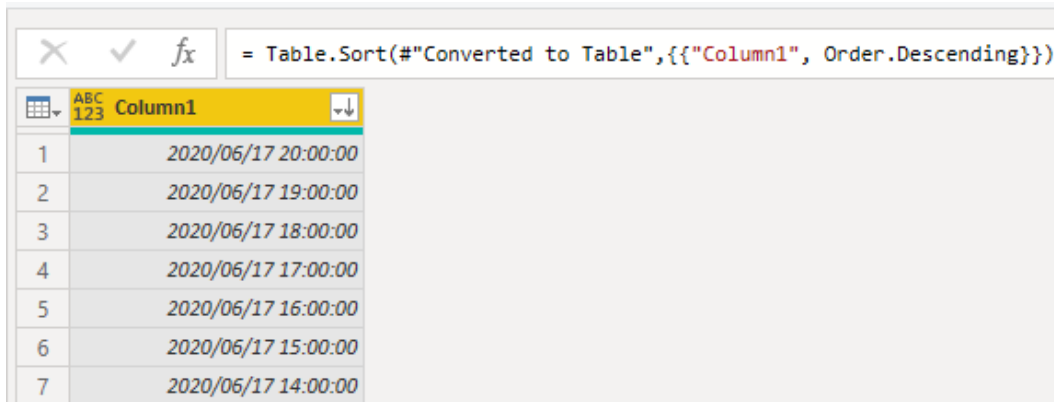


Figure 11.35 – The `DateTime.LocalNow` function added to the M code

18. Click on the **Sorted Rows** step to view the change to the data:



```
= Table.Sort(#"Converted to Table",{{"Column1", Order.Descending}})
```

	Column1
1	2020/06/17 20:00:00
2	2020/06/17 19:00:00
3	2020/06/17 18:00:00
4	2020/06/17 17:00:00
5	2020/06/17 16:00:00
6	2020/06/17 15:00:00
7	2020/06/17 14:00:00

Figure 11.36 – DateTime.LocalNow updated to the system's current date and time

In this section, we mastered working with dates in Power Query using the `List . DateTime`, `Duration . TotalDays`, and `Time . LocalNow ()` M functions to create a date and time column. Have a browse through the link provided at the start of the section to broaden your knowledge about the other date and time functions that you can apply to datasets. We hope that the three functions included in this topic have sparked your interest and that you are already thinking of scenarios where you could apply them to your datasets.

Summary

In this chapter, you mastered changing a file path to a different folder location on your computer, and you should now be able to create an M function manually by editing code using the Advanced Editor window to create a parameter function.

You should be able to test a parameter query and create a new column to invoke a custom function. We covered Power Query date and time functions using the `List . DateTime`, `Duration . TotalDays`, and `Time . LocalNow ()` M functions to create a date and time column, and now we should be able to tweak the data using all three of the functions to edit M code.

In the next chapter, we'll cover the differences between DAX and M. You will understand the differences between the two languages by going through some examples and learning to create calculated measures.

12

Differences Between DAX and M

M is the mash-up functional language of Power Query. Its formal name is Mashup or Power Query Formula Language, and it is used to query many data sources, while **Data Analysis Expression (DAX)** allows functions, much like Excel, to work on data stored in tables.

In this chapter, you will learn the differences between the two languages as we will look at the properties of both DAX and M. We will discuss the DAX syntax to understand how formulas are constructed, as well as look at how to add a DAX formula in Excel. Toward the end of the chapter, you will learn how to create a calculated column to display the result of a formula in a new column, and create a measure to calculate aggregates.

We will cover the following main topics in this chapter:

- Learning about the DAX and M functionality
- Constructing DAX syntax
- Creating a calculated column
- Creating calculated measures

Technical requirements

It would be useful for to go through *Chapter 9, Working with M*, *Chapter 10, Examples of M Usage*, and *Chapter 11, Creating a Basic Custom Function*, prior to starting this chapter. The previous chapters contain valuable examples and discussions regarding the M language. You'll find all the relevant code files used in this chapter at <https://github.com/PacktPublishing/Learn-Power-Query>.

For Code in Action video of this chapter please visit the following link: https://www.youtube.com/watch?v=up4CCRSNj7I&list=PLeLcvrwLe186O_GJEZs47WaZXwZjwTN83&index=13&t=0s.

Learning about the DAX and M functionality

In this section, you will learn the differences between DAX and M and find out when each one is used, as well as their functions.

DAX is mainly used in data transformations in Power BI dashboards where you would need to gain business analysis from existing data models. DAX is powerful when required to produce, for instance, growth percentage analysis across a list of products for specific date ranges, or to analyze market trends. Together with Power BI, DAX can powerfully assist with real-world business challenges to create meaningful, interactive business dashboards for reporting and decision-making. It is a formula language; it is not considered a programming language as it is structured using custom calculations (in fields and columns). As a formula language, DAX comprises an assembly of the following:

- Operators
- Constants
- Functions

Now, M code is used in the Power Query editor. Each time you create a transformation, expressions are generated automatically. To view or edit any M code with the Power Query interface, you need to visit either the **Advanced Editor** window or the formula bar. We have already discussed and worked with M code in the other chapters of this section of the book.

In the following table, we have highlighted the main differences between the two language platforms as a summary:

Properties	M	DAX
Language used in...	Power Query, Power BI, Analysis Services, and Excel	SQL Server Analysis Services Tabular, Power BI, and Power Pivot (Excel)
Stands for...	Mashup or Power Query formula language	Data analysis expression language
Developed by...	Microsoft	Microsoft
What is it capable of?	Connecting data sources The mashup and transformation of data	Selecting, joining, merging, and filtering data in a dynamic way, as well as creating calculated columns, measures, and tables Doesn't use programming blocks
How does it work?	With a step-by-step, functional, powerful, case-sensitive programming language structure.	With expressions just like Excel functions. It is a formula language, not a programming language.
Advantages	Easy to use with transformational steps. Uses <code>let</code> expressions and <code>in</code> statements.	Very powerful, with many of the same functions as Excel. Therefore, it is easy for Excel users to understand. Used for the analysis of business queries.
IntelliSense	M IntelliSense	It has a suggestions feature
Resource	https://docs.microsoft.com/en-us/powerquery-m/	https://docs.microsoft.com/en-us/power-bi/transform-model/desktop-quickstart-learn-dax-basics
Function reference list	Built-in #shared library with support https://docs.microsoft.com/en-us/powerquery-m/power-query-m-function-reference	https://docs.microsoft.com/en-us/dax/dax-function-reference

Table 12.1 – Differences between DAX and M

Now that we have learned about the different properties of DAX and M, let's investigate DAX a little further.

Constructing DAX syntax

Every language you come across will have a syntax and a structure. Syntax refers to all the elements that you use to construct a formula. A formula usually consists of functions. In *Chapter 9, Working with M*, in the *Understanding the M syntax and writing with M* section, you learned all about the M syntax. In this topic, you will learn about the DAX table and column name syntax.

It is important to note, especially if you are only just beginning to learn about DAX, that it is broken up into two parts:

DAX	DAX tool to use
DAX queries	Analysis Services, DAX Studio, and SQL Management Studio
DAX formulas	Power BI, Excel, and Power Pivot

Table 12.2 – DAX tools breakdown

You need to understand these tools as you will need to use the correct tool for the DAX code you are using. Often, we search for reference code and copy the code to adapt to our scenarios, so be mindful of this as you will most likely end up with an error and waste time trying to find out why the code does not work.

DAX functions and Excel functions are much the same in terms of behavior and type—for example, their array and lookup functions. The main difference between the two is the syntax. Excel uses cell references or ranges whereas DAX references a table or column in the formula. It should be noted that you cannot blend Excel functions and DAX functions in one formula. DAX will always require relationships to exist between tables when performing lookups.

Let's take a look at the DAX syntax elements. The following is an example of a basic DAX formula created in Power BI:

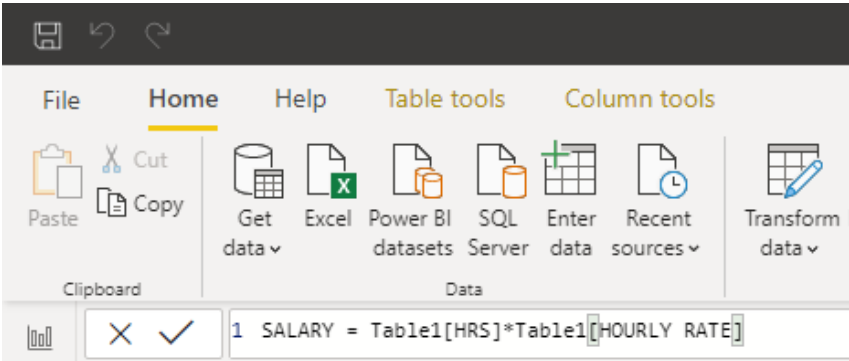


Figure 12.1 – Power BI Desktop DAX calculations

Note that not all constructions of formulas will be the same, as some formulas do not contain certain elements, such as functions.

The following table explains each part of the DAX syntax:

Parameter	Explanation
Name	When entering DAX syntax, you always start with the name of the element you are creating first. In the Calculated column, this will become the new column name (heading).
=	Then, the = sign operator is added to indicate the start of a formula, just like in Excel.
Functions	Any function name that is inserted, such as SUM, will contain a parenthesis directly after it and needs a closing-argument parenthesis after the formula is complete—for example, SUM (Table1 [HRS])
Parentheses	Parentheses (brackets) are used to surround an expression that contains more than one argument.
Referenced table	This is a referenced table; in the case of our example, Table1 [HRS] , where Table1 is the table we are referring to.
Referenced column	This is a referenced column; in the case of our example, Table1 [HRS] , where HRS is the column within the table that we are referencing.
Expression	Some functions, such as CALCULATE, require an expression. An expression could be a measure. You will learn all about measures next.

Table 12.3 – A breakdown of DAX formula syntax

Note

If you do not enter the correct syntax, a syntax error will be returned in the column.

In the preceding table, we looked at the DAX syntax and how to construct a DAX formula. Let's see where we formulate DAX syntax in the next topic.

Constructing DAX formulas in Excel

To create a DAX formula in Excel, you use the **Power Pivot** window. The following explanation assumes you know how to add Excel data to Power Pivot. This was explained in *Chapter 2, Power Pivot Basics, Inadequacies, and Data Management*. There are two ways to go about it:

- Click on the Power Pivot formula bar and start constructing the formula, as follows:

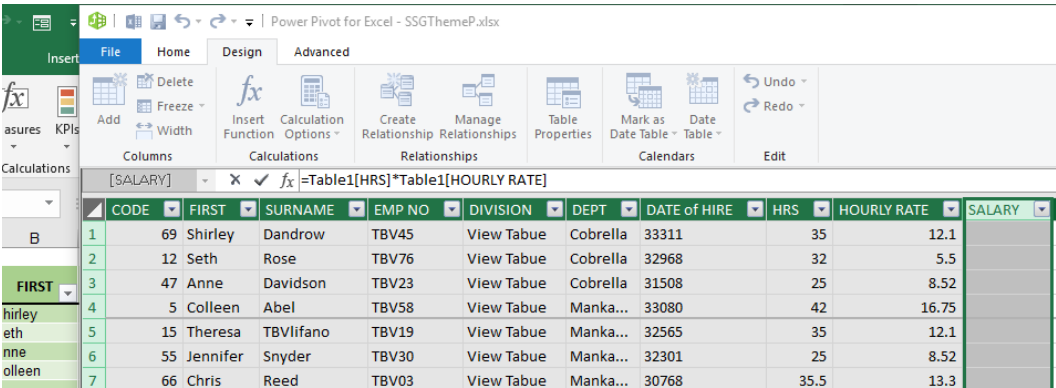


Figure 12.2 – The Power Pivot DAX formula construction

- Use the **Design** tab in the Power Pivot ribbon to access the **fx (Insert Function)** icon to select a function to use within the formula.

We learned where to type a DAX formula in Excel in this section. The next section will address how formula help is available when constructing a DAX formula.

Using IntelliSense

IntelliSense provides the user with a list of functions or parameters to help when constructing a DAX formula. This is the same as when we enter a function in Excel. The help tool is available to give us tips and offer help with functions by indicating what we need to make the formula work. This is the same for DAX formulas. By entering characters such as parentheses, quotes, or square or curly brackets, IntelliSense automatically creates the closing element for these characters.

IntelliSense also offers different types of elements to choose from by providing a drop-down list of elements. To the left of these elements, you will see an icon. The cube, triangle, square, and circle icons depict different things. The cube signals all the native M functions and the triangle, circle, and square are for all the variables, steps, parameters, constants, query names, and so on.

IntelliSense also highlights keywords and offers parameter hints; help is available at your fingertips:

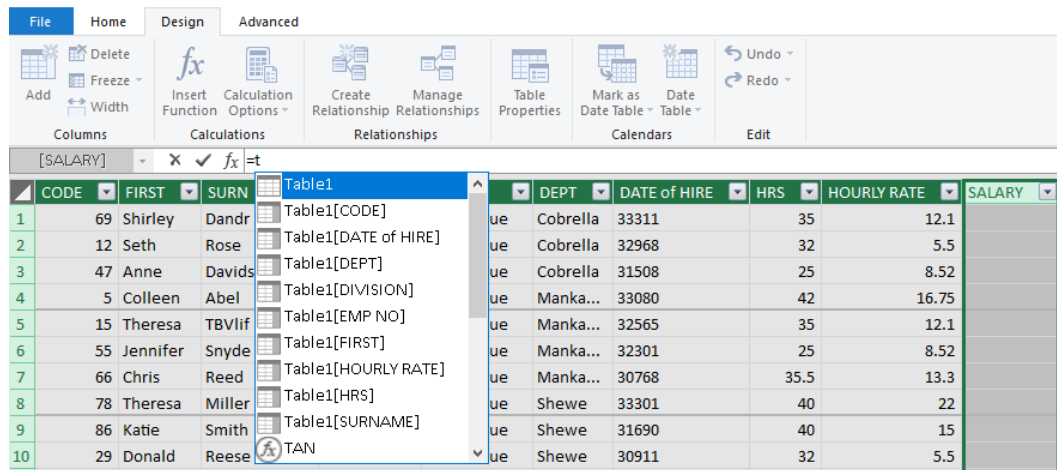


Figure 12.3 – A DAX IntelliSense example using the formula bar in Power Pivot

Here is an example of IntelliSense in Power Query using M, showing the cube element to help with code construction:

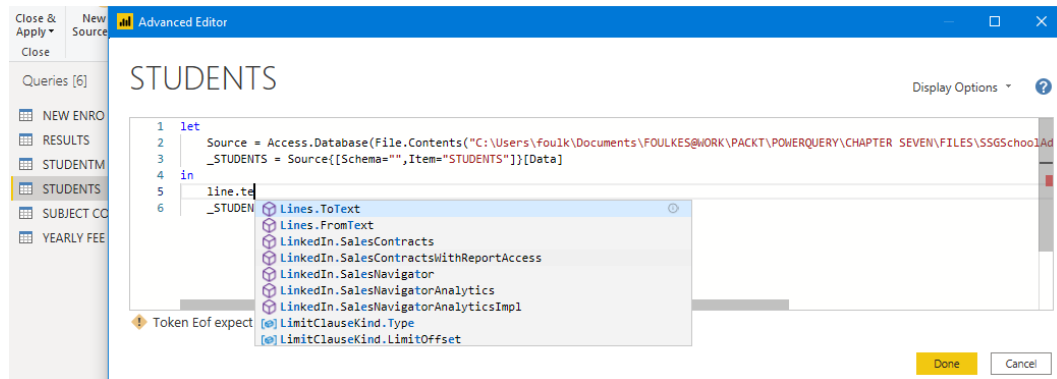


Figure 12.4 – An M code IntelliSense example using the Advanced Editor window in Power Query

Now that you understand DAX syntax and construction and know how IntelliSense can support formula construction, we will introduce you to DAX formula types.

Creating a DAX formula

There are a number of formulas that you would create in Power BI using DAX. These are all visible on the Power BI ribbon under the **Calculations** group—namely, **New measure**, **Quick measure**, **New column**, and **New table**, as shown:

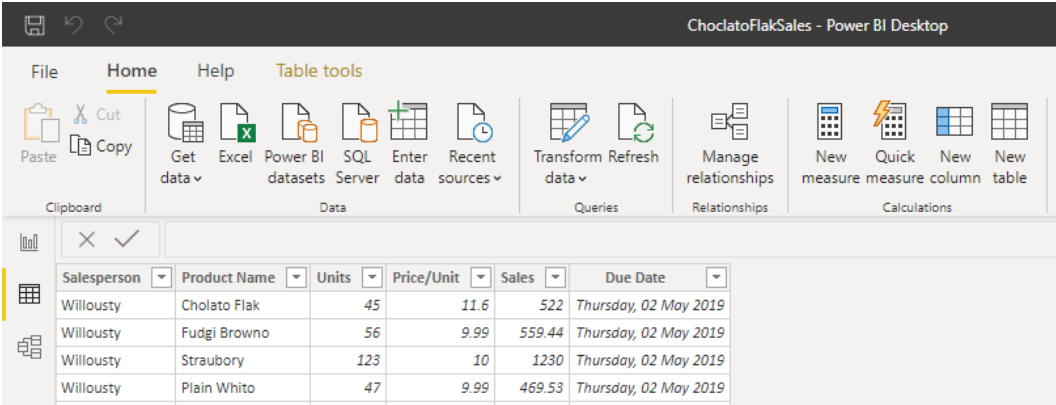


Figure 12.5 – The Calculations group under the Home tab in Power BI Desktop to access the DAX features

Here is an explanation of the three main types of DAX formulas. The following calculations described are accessible via the **Calculations** group under the **Home** tab. Here, you will see **New measure**, **Quick measure**, **New column**, and **New table**:

- **Calculated columns:** These are used to perform calculations on rows automatically by adding a column or columns from different tables.
- **Quick measures:** These are built-in measure templates, which are already constructed for you and can be added without prior knowledge of coding.
- **Calculated measures:** These do not take up any physical memory and the results are more dynamic due to their adaptability. This type of calculation always requires an aggregator (function).
- **Calculated tables:** With this formula, you can create new tables and data.

Now that you have an overview of the different types of DAX calculations available, we will look at some of them in detail.

Understanding the DAX formula and storage engine

DAX has a formula engine and a storage engine, which are responsible for running DAX queries within Power BI. In the following table, we have outlined the responsibility of each engine and a summary of the main advantages and disadvantages of each:

Engine Type	Responsibility	Advantages and Disadvantages
Formula (FE)	<ul style="list-style-type: none"> • Reads queries • Interprets queries • Requests data from the storage engine • Executes queries 	<ul style="list-style-type: none"> • Single-threaded: Can only process one thing at a time, so is a bit slow • Solves query problems
Storage (SE)	<ul style="list-style-type: none"> • Takes care of all the data by compressing and analyzing it, as well as ensuring the speedy accessibility of data and management of simple queries (such as SUM, COUNT, and so on) • A database that runs on top of Power Query and Power BI called Veripaq 	<ul style="list-style-type: none"> • Multi-threaded: Processing is extremely fast as it can do more than one action at a time. • Cannot solve any complex queries and needs the formula engine to solve the query. • Has difficulty processing huge datasets that include row-by-row calculation (a calculated column). Would need to send calculations back and forth to the formula engine to perform calculations row by row in this case. • Is extremely fast at producing a result from a filtered table with a measure applied, for example, so the correct formulation of DAX formulas is crucial for speedy action. An example is using a measure instead of a calculated column.

Now that you have an understanding of what tasks the storage and formula engine perform in DAX, let's discuss some examples in the following sections.

Creating a calculated column

In this section, we will create a calculated column in Power BI Desktop to slice or filter a value or calculation on every row in a table. Let's run through the logic of what creating a calculated column means.

A calculated column is a new column that is added to an existing table. For each row of the table, the DAX formula is calculated immediately, just like using the autofill handle in Excel to fill in a formula. Be mindful of the fact that when using calculated columns, the result of the calculation is always stored in memory, unless of course it is reloaded or released when exiting or opening up Excel/Power BI. This will cause the table to be refreshed, which forces the column to recalculate. Let's see how this works:

1. Open Power BI. For this example, we will be using the `SSGThemePark.pbix` file.
2. You can create the DAX formula in any view, but I prefer the table view so that I can identify the data and see how the data is structured in the columns. It makes it easier when you can see column headings when working with formulas. The view you choose to create the DAX formula in is purely based on personal preference.
3. Click on the **New column** icon in the **Calculations** group of the **Home** or **Table tools** tab. A new column is inserted to the right of the existing dataset, with a heading named `Column`. Notice that the formula bar is also active at this point, waiting for user input:

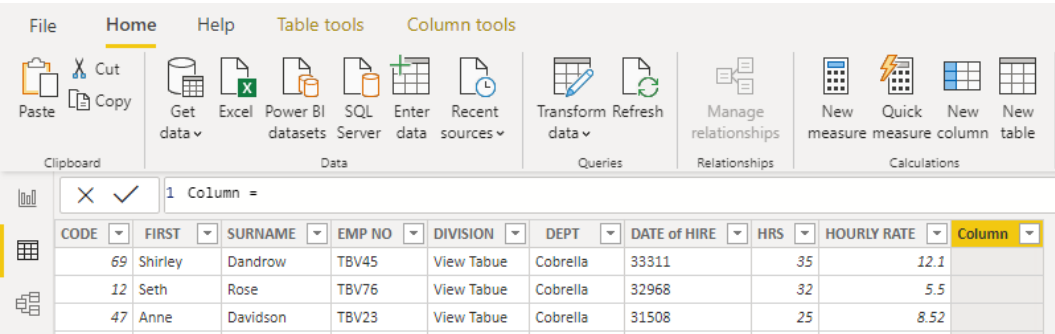


Figure 12.6 – A new column is added to the existing dataset and the formula bar is active

4. To construct the DAX formula, simply type it into the formula bar. The formula bar is situated directly above the table headings. We will use an easy example to calculate the salary for each employee based on the hours worked multiplied by the hourly rate.

- The formula will be entered as follows. Change the Column text to SALARY so that the column heading is relevant to the calculation being performed on the column. After the = sign, start typing the name of the column that you would like to use; in this case, it will be the HRS column. Notice that once you start typing, the program automatically populates suggestions for you. Also, notice how the icons differ in the drop-down list according to the elements populated. In the following drop-down list, we can see function icons and table icons. Double-click on **Table1[HRS]** to include it in the formula:

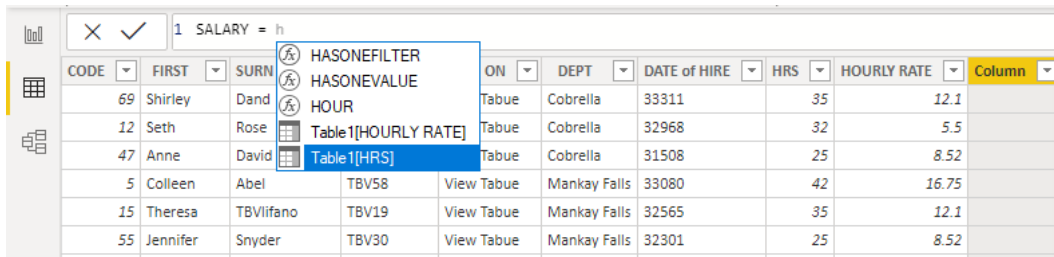


Figure 12.7 – The list is filtered as you type the name of the column you wish to use in your formula

- Add the multiplication operator (*), then type HOURLY RATE to end the DAX formula:

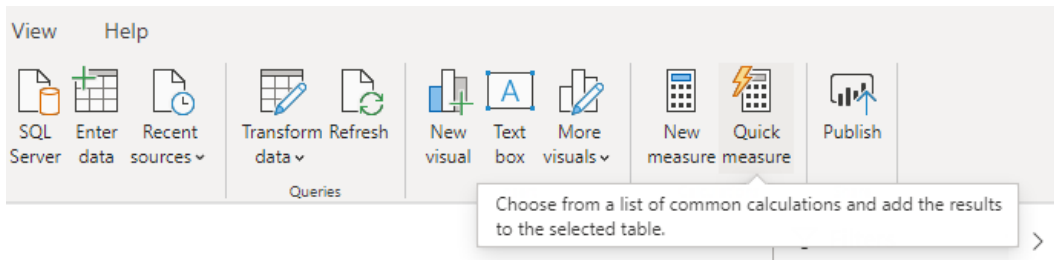


Figure 12.8 – The DAX formula is entered to calculate row values

- This will now update the SALARY column, as shown in the preceding screenshot.

You have now learned how to create a calculated column using DAX formula construction. We will now look at the next type of DAX calculation—using measures.

Creating calculated measures

In this section, we will learn when to use the calculated measure feature and how to create a new measure by adding a function or expression. Let's look at some of the things we need to know before creating a calculated measure.

All measures need to contain a function. A measure cannot work without the table column you are creating the measure on having a function (`sum`, `min`, `count`, and so on) within the formula—this is the difference between a calculated column and a calculated measure. This is called an *aggregator* (function), and without an aggregator, it is called a *naked column* in the programming world. IntelliSense is very useful here as it will complete your code for you.

The beauty of measures is that they are only calculated when they are accessed, and so they don't use up your memory. They also allow many different outputs to be produced by just changing the filter criteria of the existing measure. Measures are created in the **Report** or **Data** view in Power BI.

A measure can be applied across tables from one query to another with ease. It adapts just the same as an Excel formula would in a workbook when referencing cells. We can move measures without losing any functionality within the measure. When we create calculated columns, the data is stored in the xVelocity engine, so calculated columns take up more storage in your database. The virtual memory used in calculated columns is much smaller, however, when interacting through reports. Measures are faster to load as they are not stored in memory.

Before creating a calculated measure from scratch, let's have a look at the **Quick measure** feature, which provides built-in, readily created formula templates for use.

Using quick measures

We can use measures that are built for us, or we can create our own measure constructions using DAX formulas. If you are a beginner, the best method is to use the **Quick measure** feature, which returns a list of common calculations for you to choose from and apply to your dataset. This will help you to understand DAX formulas and become familiar with the DAX structure.

Take the following steps to test this measure:

1. Open the dataset you wish to perform the measure on. We will use `SalesData.pbix` for this example.
2. From the **Home** tab, locate the **Quick measure** icon and select it, or right-click on an existing table and select **Quick measure**:

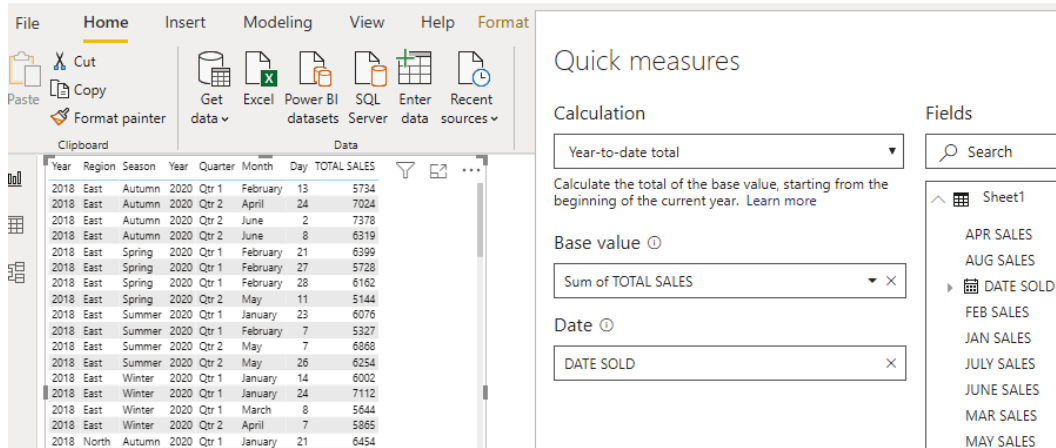


Figure 12.9 – The Quick measure option in Power BI

3. The **Quick measures** dialog box will populate, where you can select from a list of calculations to apply to your table.
4. Select **Year-to-date total** for this example.
5. Drag the fields from the table that you wish to use in the calculation. For this example, we will drag the **Sum of TOTAL SALES** field from the **Fields** list into the space provided under the **Base value** setting:

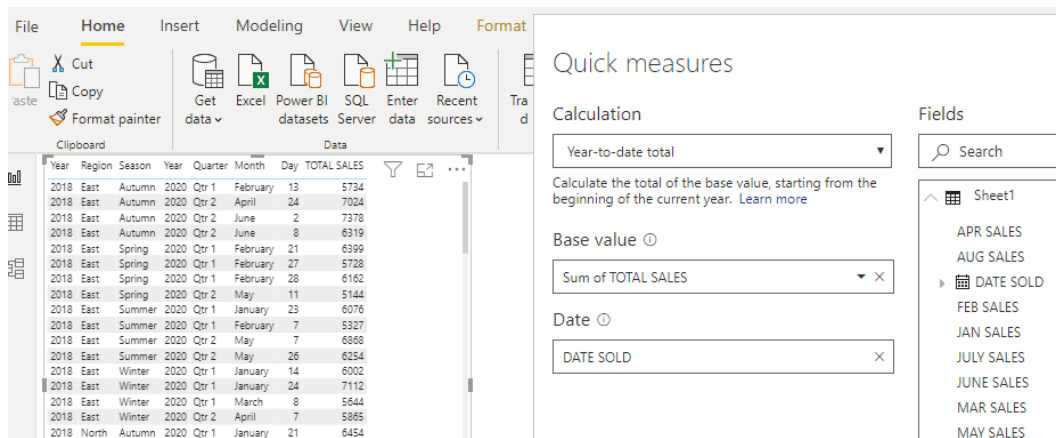


Figure 12.10 – Quick measure calculations

6. Notice how you can change the calculation type by clicking on the drop-down arrow to the right of the box, as in the following screenshot:

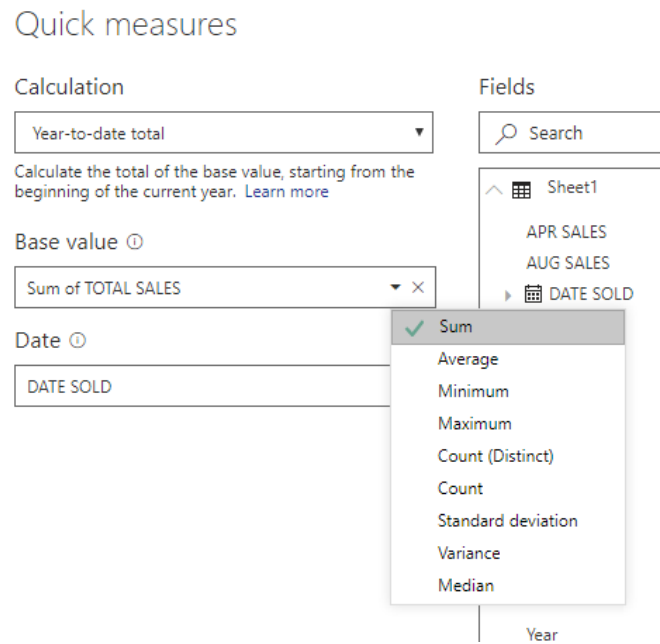


Figure 12.11 – Adding a field to the Base value field or selecting a function from the list

7. Drag the **DATE SOLD** field from the **Fields** list into the space provided under the **Date** setting.
8. Click on **OK** to let Power BI work out the DAX function for you.

The measure is added to the **Fields** list pane:

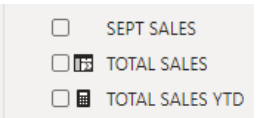


Figure 12.12 – A measure called TOTAL SALES YTD added to Field list

9. Drag the **TOTAL SALES YTD** measure onto the existing table dashboard. The table is updated with the new column reflecting the **YTD**:

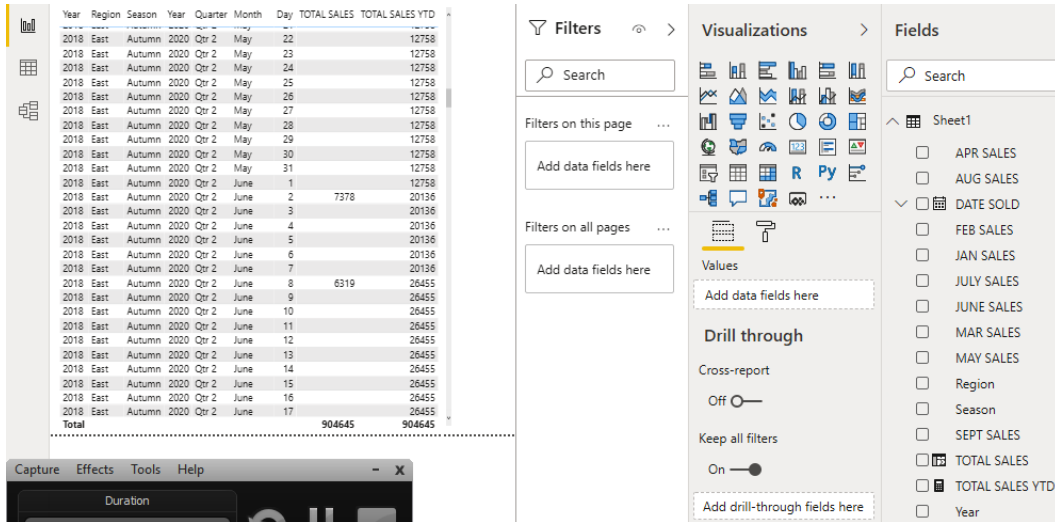


Figure 12.13 – The updated dashboard

- Click in the formula bar to view the DAX formula that was created by running through the previous few steps in the user interface:

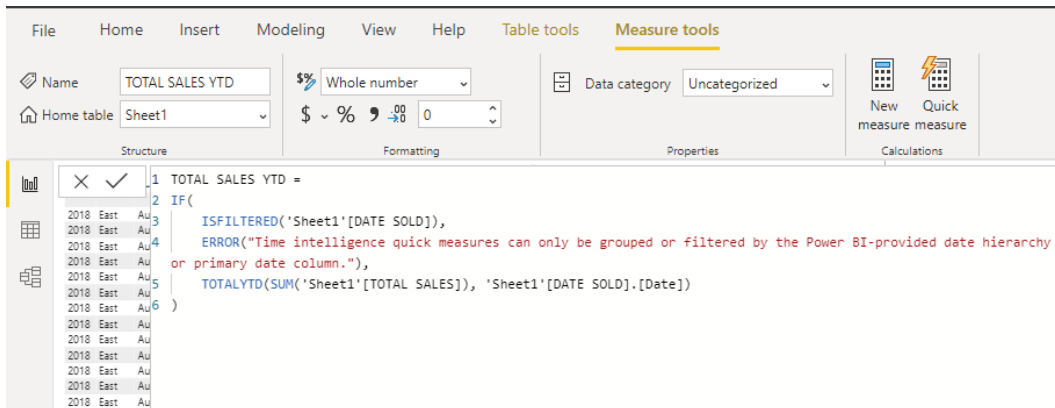


Figure 12.14 – The DAX formula for TOTAL SALES YTD

After learning about the **Quick measure** feature, we can now investigate further by constructing our own DAX measure using code.

Formulating a DAX measure from scratch

We will continue with the example code file from the preceding section to learn about the `CALCULATE` and `FILTER` DAX formulas.

As we will be calculating the sum of the sales by quarter, we will need the `Filter` function to divide the quarter into categories. After we have done this, the formula will calculate the total sales according to the quarter categories. The reason for using this measure is that a quarter can have new salary information refreshed at any time, thereby increasing the number of rows.

Let's create a new measure to add to our report dashboard:

1. Using the example file from the previous example, `SalesData.pbix`, we will add the `TOTAL SALES` field to the dashboard as a **Card** visualization type (just next to the existing table). Use the **Format** icon to change the **Data label** setting to **None** to display the total sales for the entire dataset. This is done to show you that there is no filter applied to this field:

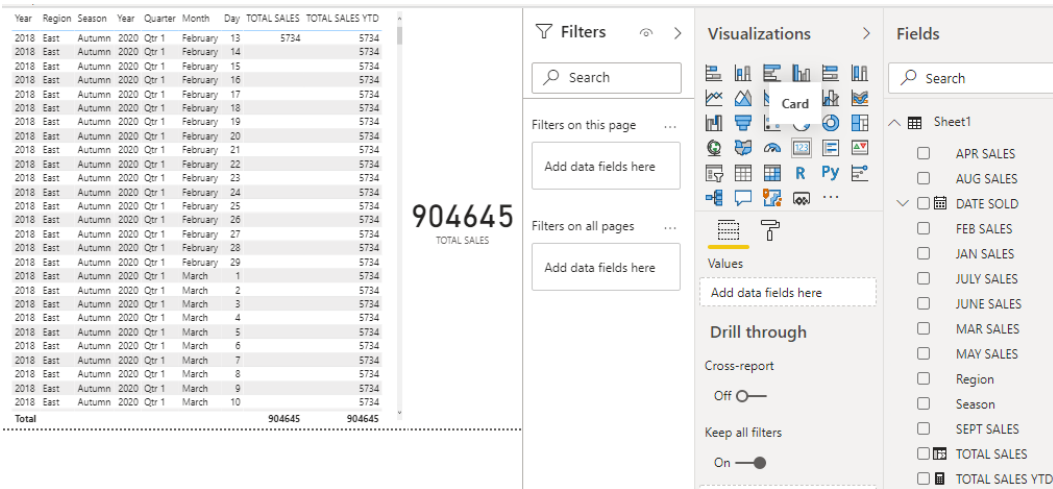


Figure 12.15 – The `TOTAL SALES` field added to the dashboard as a Card visualization

2. Resize the total sales card so that it does not take up the entire screen.
3. Add the **Year**, **Region**, and **TOTAL SALES** fields from the **Fields** list into the white space of the dashboard to create a filter. This is done to show you that we can create filters without DAX formulas, too:

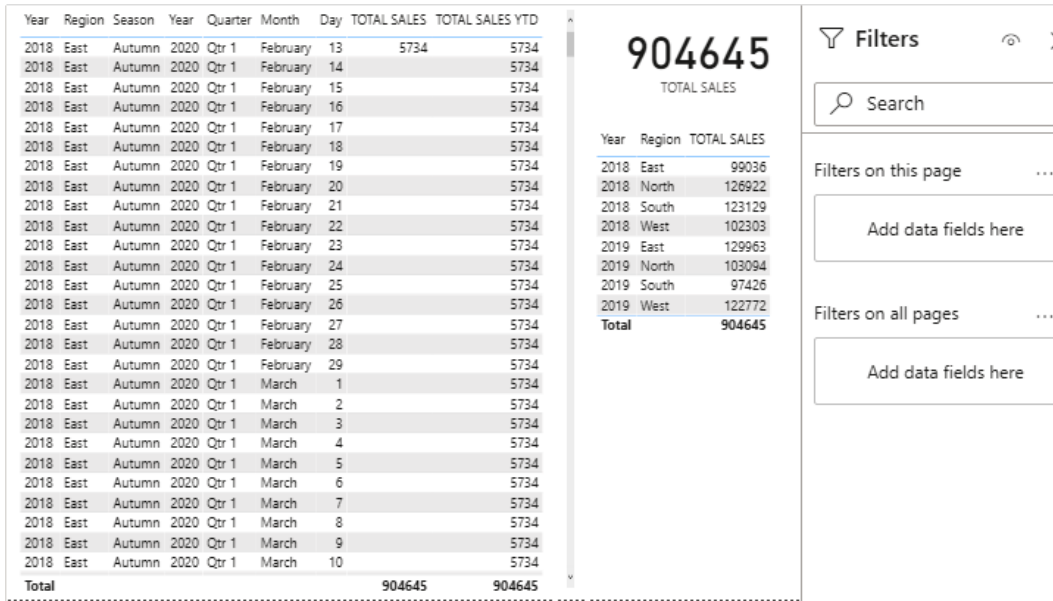


Figure 12.16 – Year, Region, and TOTAL SALES added to the dashboard

- Next, we will look at how to work out the total sales per season. Make sure you have clicked on the **Reports** icon in the navigation pane.
- Drag the **Season** and **TOTAL SALES** fields onto the dashboard. This automatically creates a filter on the **Season** column for you. Although this is easier than creating a measure, it would be better practice to do this using a measure as you would need to use measures as expressions when building a DAX formula:

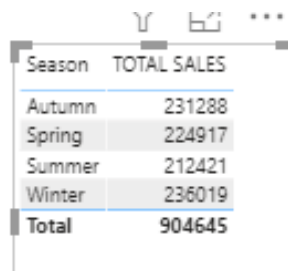
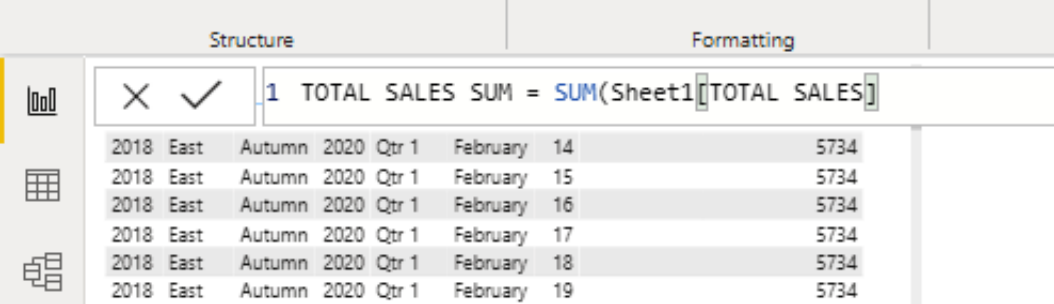


Figure 12.17 – Season and TOTAL SALES on the dashboard

Let's do the same thing here, but by using a measure instead.

- Select the table you want to use for the measure before creating the measure; otherwise, it will select the first table by default. Note that this can be changed later, but it is easier if you select the table beforehand.

7. In the Power BI interface, select **New measure**. We will be typing the following code into the formula bar in order to work out the total sales for each season using a filter and a calculation:



The screenshot shows the Power BI interface with the 'Structure' and 'Formatting' tabs at the top. The formula bar contains the code: `1 TOTAL SALES SUM = SUM(Sheet1[TOTAL SALES])`. Below the formula bar, a table is displayed with the following data:

2018	East	Autumn	2020	Qtr 1	February	14	5734
2018	East	Autumn	2020	Qtr 1	February	15	5734
2018	East	Autumn	2020	Qtr 1	February	16	5734
2018	East	Autumn	2020	Qtr 1	February	17	5734
2018	East	Autumn	2020	Qtr 1	February	18	5734
2018	East	Autumn	2020	Qtr 1	February	19	5734

Figure 12.18 – A TOTAL SALES SUM formula

Note that if you do not have an aggregator (function) in your code, then you will not be able to create a measure—IntelliSense will watch you while you work and will not offer you the field name in order to go further with the code if you omit the function. If you don't see the field populate in the auto-complete pane when constructing your code, then it is probably because you do not have the correct conditions in place to create the measure.

When you start typing the code, you will be offered suggestions (through IntelliSense) based on your input. To navigate through the list offered, simply use the down arrow on the keyboard and then the *Tab* key to select the element. Alternatively, just double-click on an element to add it to the code.

To add a new line in your code, press *Shift + Enter* on the keyboard. So, now you know a few things about entering code.

8. If you now drag the **TOTAL SALES SUM** measure from the **Fields** pane onto the table, you will notice that the values are exactly the same. The underlying difference is that the measure will remain the same, whereas the **TOTAL SALES** column could be renamed or removed at some stage and cause problems with our formula in the future.
9. We can remove the **TOTAL SALES** column from our visualization now as we have put the measure in place.

10. Now, we will use the `CALCULATE` function to calculate the sum of our quarters.
When using the `CALCULATE` function, we need to have a measure—we cannot use the table and column references.
11. Type the measure name, then `=`, followed by the `CALCULATE` function, which will require an expression (a measure) within parentheses. Refer to the following screenshot to see how this should look:

The screenshot shows the DAX formula bar with the measure definition: `1 CALQ = CALCULATE([TOTAL SALES SUM])`. Below the formula bar, a table of data is displayed, showing sales figures for various quarters and regions.

Year	Region	Autumn	2020	Qtr 1	February	14	5734
2018	East	Autumn	2020	Qtr 1	February	15	5734
2018	East	Autumn	2020	Qtr 1	February	16	5734
2018	East	Autumn	2020	Qtr 1	February	17	5734
2018	East	Autumn	2020	Qtr 1	February	18	5734
2018	East	Autumn	2020	Qtr 1	February	19	5734
2018	East	Autumn	2020	Qtr 1	February	20	5734

Figure 12.19 – Measure code to calculate total sales

12. Drag the **CALQ** measure from the **Fields** list onto the table dashboard:

The screenshot shows the Power BI interface with the **CALQ** measure added to the dashboard table. The table displays sales data for various regions and seasons, with the **CALQ** measure calculated for each row.

Year	Region	TOTAL SALES
2019	South	97426
2018	East	99036
2018	West	102303
2019	North	103094
2019	West	122772
2018	South	123129
2018	North	126922
2019	East	129963
Total		904645

Season	TOTAL SALES SUM	CALQ
Autumn	231 288.00	231 288.00
Spring	224 917.00	224 917.00
Summer	212 421.00	212 421.00
Winter	236 019.00	236 019.00
Total	904 645.00	904 645.00

Figure 12.20 – The CALQ measure added to the dashboard table

13. CALCULATE is really powerful when you add the FILTER function to it. We will now combine the FILTER function with the CALCULATE function to filter the Season field for Spring only:

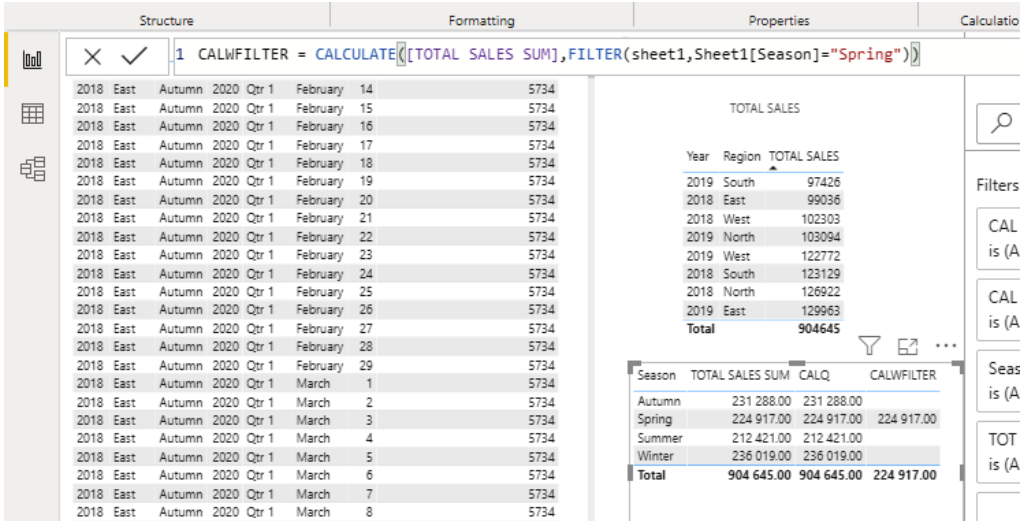


Figure 12.21 – The CALCULATE filter for Spring

14. Now, drag **CALWFILTER** from the **Fields** list onto the table dashboard. Notice how only Spring has a value now.
15. Add the **Year** field to the table dashboard:

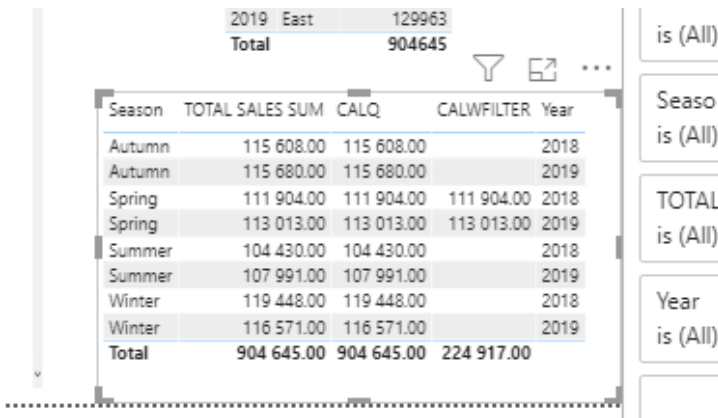


Figure 12.22 – Year added to the table dashboard

16. The filter now works for 2018 and 2019. You can use the sort drop-down arrow on the Year column to sort it in descending or ascending order to see the results.

I hope this gives you some idea of what is possible with DAX formula measures. Next, we will move on to organizing measures.

Organizing measures

Measures are identifiable by a calculator icon to the left of the measure name in the Power BI interface:

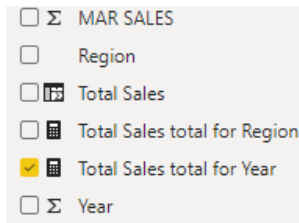


Figure 12.23 – Measure identified by a calculator icon

We can place all our created measures into a display folder in the folder list pane. All of our measures can be stored in one folder, even if they are associated with different tables:

1. Click on the **Model** icon in the navigation pane.
2. Select the measure you wish to move into a display folder:

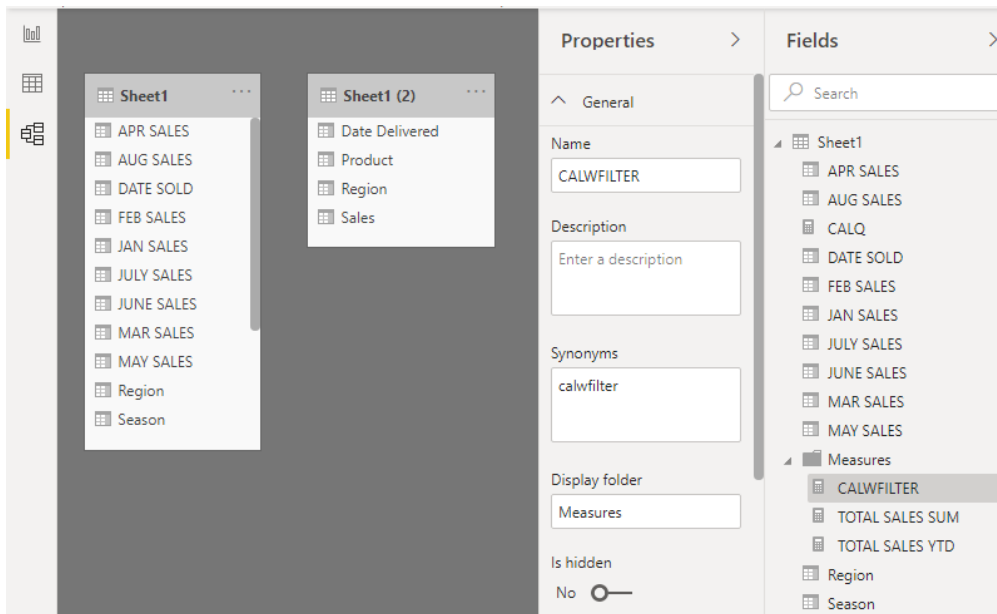


Figure 12.24 – Display folder creation in the Model view

3. The **Properties** pane will open to the left of the **Fields** list.
4. In the **Display folder** area, type the name of the folder you wish to place the measure into.

If you want to create a subfolder, then type a backslash (\) after the display folder name entered into the area provided—for instance, `Measures\Sales`.

You have now learned how to organize your folder structure in the **Fields** list pane.

Summary

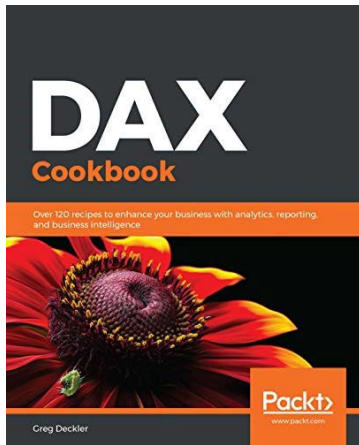
In this final chapter of the book, you have learned all about the differences between DAX and M functionality. You will now be confident in identifying the different parts of the DAX formula syntax structure and should be knowledgeable about where to construct DAX formula in Excel, Power Query, and Power BI.

We also touched on IntelliSense, which aids in DAX formula construction, and we know about the three main types of DAX formula (which are calculated columns, calculated measures, and calculated tables). You can create a calculated column, as well as quick measures and calculated measures. You can also create folders in which to store measures in the **Field** list of the Power BI interface. Measures will help you construct an expression formula to calculate a result specific to a purpose, dynamically and in real-time.

We hope that you now have a wealth of knowledge after reading this book and that it has instilled a desire to learn more about this extremely powerful, in-depth application named Power Query.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

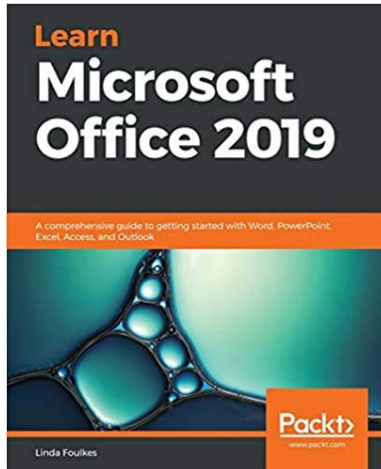


DAX Cookbook

Greg Deckler

ISBN: 978-1-83921-707-4

- Understand how to create common calculations for dates, time, and duration
- Create key performance indicators (KPIs) and other business calculations
- Develop general DAX calculations that deal with text and numbers
- Discover new ideas and time-saving techniques for better calculations and models
- Perform advanced DAX calculations for solving statistical measures and other mathematical formulas
- Handle errors in DAX and learn how to debug DAX calculations
- Understand how to optimize your data models



Learn Microsoft Office 2019

Linda Foulkes

ISBN: 978-1-83921-725-8

- Use PowerPoint 2019 effectively to create engaging presentations
- Gain working knowledge of Excel formulas and functions
- Collaborate using Word 2019 tools, and create and format tables and professional documents
- Organize emails, calendars, meetings, contacts, and tasks with Outlook 2019
- Store information for reference, reporting, and analysis using Access 2019
- Discover new functionalities such as Translator, Read Aloud, Scalable Vector Graphics (SVG), and data analysis tools that are useful for working professionals

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

Symbols

#shared libraries

- lists, creating 304

- number data types, creating 302, 303

- records, creating 305

- relevant data, searching 307, 308

- table data types, creating 306

- text data types, creating 301, 302

- using, to return library functions 301

A

Advanced Editor window

- comment, adding to 74

- comment, adding with formula bar 75

- comment, keeping visible in

 - formula bar code 75

- using 67-73

ampersand operator (&) 321

And/Or conditions

- used, for filtering table data 153-155

Android

- Power Query Office 12

APPLIED STEPS

- documenting 61, 62

- multiple steps, deleting 60

- renaming 61, 62

- working with 60

automatic page refresh

- about 245, 246

- reference link 245

- setting up 245

B

basic power query

- creating 76-79

C

calculated column

- about 382

- creating 384, 385

calculated measures

- about 382

- creating 385

calculated tables 382

chart

- selecting 272-279

comma-separated value (CSV) 236

concatenate formula

- using, for merging 314-320

- conditional column
 - creating, with the if...then...
 - else statement 150-153
- Create, Retrieve, Update,
 - Delete (CRUD) 96
- CSV file
 - importing, with M language 308-310

D

- dashboard
 - publishing 279-282
 - saving 279-282
 - sharing 279-291
- dashboard, best practices
 - about 292
 - audience 292
 - clutter 292
 - color 292
 - size 292
- data
 - adding, to data model with
 - Power BI 267-271
 - collecting, and connecting
 - with Power BI 252-258
 - connecting, from relational database 111
 - connecting, from table or range 99-103
 - connecting, through Power BI 114-117
 - connecting, to web 103-111
 - connecting, through Excel's Get
 - & Transform 112, 113
 - custom connections 117-119
 - refreshing 144-147
 - turning, with pivot tool 140-144
 - turning, with unpivot tool 140-144
- Data Analysis Expression (DAX) 375

- Data Analysis Expression
 - (DAX) functionality
 - versus M functionality 376, 377
- database management systems
 - (DBMS) 98
- databases
 - about 96-98
 - types 96-98
- data, custom connections
 - connecting, from folder 124-132
 - connecting, from Workbook 120-123
- dataflow refresh
 - reference link 246
- data model
 - about 4
 - data, adding with Power BI 267-271
 - pivot table, adding 34-36
 - reference link 4
- data profiling, tips
 - about 89-91
 - column distribution 94
 - column profile 92
 - column quality 92, 93
- dataset
 - selecting 272-279
 - types 226
- data source
 - parameterizing 335-341
- Data source settings
 - exploring 133
 - exploring, from Excel 133-135
- data table
 - previewing 57-59
- data type conversions 321, 322
- Data view
 - parameters, using 348

data visualization
 selecting 272-279

date/time column
 creating, with M functions 364-373

DAX formula
 about 383
 constructing, in Excel 380
 constructing, with IntelliSense 380, 381
 creating 382
 types 382

DAX formulas, types
 calculated columns 382
 calculated measures 382
 calculated tables 382
 quick measures 382

DAX measure
 formulating, from scratch 389-394
 organizing 395, 396

DAX storage engine
 about 383
 advantages 383
 disadvantages 383

DAX syntax
 constructing 377-379

E

Excel
 Data source settings, exploring
 from 133-135
 DAX formulas, constructing in 380
 pivot table, creating in 30-33
 Power Query, launching with 18

Excel files
 combining 258-266

Excel's Get & Transform
 data, connecting through 112, 113

F

function
 creating, manually 355, 356
 creating, manually with M 352, 353

G

Go to Column feature
 using 66, 67

I

IF function
 writing, in Power Query 184-190

Import storage mode setting
 selecting 233

incremental refresh
 about 244
 advantages 245
 reference link 245

index functions 201-213

IntelliSense
 used, for constructing DAX
 formulas 380, 381

L

library functions
 returning, with #shared libraries 301

lists
 creating 304

Load To... options
 default custom load settings,
 modifying 80-83
 discovering 79
 queries, loading to worksheet
 manually 83-88

M

M

- used, for creating function
 - manually 352, 353

Mac

- Power Query Office for 13

Mashup (M) 298

merging

- with concatenate formula 314-320

M functionality

- versus DAX functionality 376, 377

M functions

- used, for creating date/time
 - column 364-373

Microsoft SQL Server Analysis Services

- execution, investigating 234, 235

M language

- about 298
- used, for importing CSV file 308-310
- writing 299-301

modular functions

- about 201
- beginning with 201-204

modulo 201

M syntax 299-301

multiple files

- appending 214-219

multiple tabs

- appending 220-222

N

navigation pane

- about 54
- editing, with query options 55
- queries, grouping 55, 56
- used, for adding new query 54, 55

number data types

- creating 302, 303

O

OneDrive refresh

- performing 241-243
- viewing 241, 242

online analytical processing (OLAP) 98

online transactional processing (OLTP) 98

P

parameter function

- testing 358-364

parameters

- adding, to control statement
 - logic 344, 345
- adding, to filter data 342, 343
- adding, to order objects 346, 347
- using 335
- using, in Data View 348

parameter table

- creating, for queries 191-195
- data source, modifying 196-200

PivotChart

- creating 248-252

pivot table

- adding, to data model 34-36
- creating 23-30
- creating, in Excel 30-33
- Power Pivot, creating 30
- relationships, creating 37-39

pivot tool

- used, for turning data 140-144

Power BI

- about 6

- data, connecting through 114-117
- launching 18
- Microsoft SQL Server Analysis Services
 - execution, investigating 234, 235
- on Mac/Apple 10
- on Windows 6
- Power Query, accessing from 18-20
- refresh types 235, 236
- stored imported data, viewing 234
- using, for collect and connect
 - data 252-258
- using, to add data to data
 - model 267-271
- versions 6
- Power BI App, on iPhone
 - download link 10
- Power BI Desktop
 - download link 7
 - versus Power BI Free 6-10
- Power BI Desktop Storage mode setting
 - viewing 227-230
- Power BI online
 - reference link 237
- Power Pivot
 - about 4, 5
 - activating 16, 17
 - calculated fields 48
 - creating 30, 248-252
 - launching 18
 - limitations 47
 - Microsoft Office versions 48
 - multiple items, selecting 47
 - preview 48
- Power Pivot Office 2010 6
- Power Pivot Office 2013 5
- Power Pivot Office 2016 5
- Power Pivot Office 2019 (office 365) 5

- Power Query
 - about 11, 39
 - accessing, from Power BI 18-20
 - activating 16, 17
 - calculated column, creating 39-43
 - calculated field, creating 44, 45
 - features 11, 12
 - IF function, writing 184-190
 - installing, in Office 2010 15
 - installing, in Office 2013 15
 - launching 18
 - launching, within Excel 18
 - Office 2019 (Office 365), versus
 - Office 2016 13, 14
 - Power Pivot table, creating 46, 47
- Power Query add-in from
 - download link 15
- Power Query Office
 - for Android 12
 - for Mac 13
 - for Windows 12
 - versions 12
- Power Query window
 - data table, previewing 57-59
 - elements 52, 53
 - main ribbon 53
 - tabs 53

Q

- queries
 - parameter table, creating 191-195
- Queries list 54
- query
 - file path, changing to local path 354, 355
- Query Settings pane
 - about 59
 - query properties, modifying 59

quick measures
 about 382
 using 386-389

R

records
 creating 305
refresh types, Power BI
 about 235, 236
 automatic page refresh 245, 246
 dataflow refresh 246
 incremental refresh 244, 245
 OneDrive connection,
 refreshing 236-240
 OneDrive refresh, performing 241, 242
 OneDrive refresh, viewing 241-243
 scheduled refresh 243, 244
relational database
 data, connecting from 111
relevant data
 searching 307

S

scheduled refresh
 setting 243, 244
SQL server
 setting up 322-328
SQL Server 2016 Developer edition 323
SQL Server Developer edition
 about 322
 download link 323
SQL Server Management Studio (SSMS)
 installing 329-334
star schema 97
storage modes
 about 226

 setting 226
 viewing 30, 31
structured column 169
Structured Query Language (SQL) 111

T

table data
 age, extracting from data 177, 178
 altering, with column and
 row tools, using 147
 appending 172-174
 appending tool, using 167
 column, removing 147, 148
 columns, extracting 178, 179
 columns, merging with
 combine 167-170
 columns, splitting 164-167
 conditional column, creating with the
 if...then...else statement 150-153
 duplicate rows, removing 161, 162
 dynamic multiple criterion
 filters, creating 158-160
 extract column features, using 179-181
 extraction tools, using 177
 filtering, with And/Or
 conditions 153-155
 grouping 175-177
 index column, using 149, 150
 merging tool, using 167
 null values, replacing 162, 163
 single criteria filters, creating 156-158
 text and values, merging into
 column 170, 171
 top or bottom rows, removing 148
 working, with header row 163, 164
table data types
 creating 306, 307

text data types

creating 301, 302

Transact-SQL (T-SQL) 323

U

unpivot tool

used, for turning data 140-144

V

View settings

font, modifying 63, 64

formula bar, viewing 63

Go to Column feature, using 66, 67

investigating 62

query relationships, showing 64-66

W

web

data, connecting to 103-111

Windows

Power Query Office 12

